Context-Aware API for Android Devices

Chris Smith
Supervisor: Naranker Dulay

Abstract

In recent years location-based services have seen a dramatic increase in adoption, and all modern smartphone platforms have integrated services to facilitate the creation location-aware applications. Such applications enhance the experience of users, using location to modify content or alter the behaviour of the application to better suit the circumstances.

The product of this project is a *context*-aware API for the Android platform. This allows applications to augment the already available location data with extra *context* about the user's situation - primarily their current activity. It also develops an algorithm for recognising *places* which are relevant to the user, and monitoring which activities are performed in *journeys* between those places, thus enabling predictions of the user's likely destination based on their activity.

Research into other methods of annotating context was conducted, and it was found that most potential sources of context information either produced little or no information, or were too battery-draining to perform in a real world environment with current technology. Much effort was placed into optimising the API to have as small effect on battery life as possible.

A context-aware API was successfully produced, along with a collection of applications which use the API in order to demonstrate its features or provide example use cases.

Contents

List of Figures

List of Tables

Part I. Introduction

1 Proposal

Objective: to create an API for android applications to query the user's [probable] current activity, and to consider and implement possible uses for this API in existing applications. The user's activity would be determined based on available sensor and ambient data (e.g. time, location, orientation/movement of device, background noises, camera image, in-range bluetooth devices, etc), previous behaviour of the user, and possibly behaviour of other users which has been shared between devices.

Motivation: Activity-awareness would be a major step forward in making mobile devices better able to adapt to what the user wants to do with them. The latest generation of mobile phones have made location-aware applications quite ubiquitous, and a lot of these could be further enhanced by making them activity aware. For example, an application which lists businesses in a certain area could not only know the search area (by merit of being location-aware), but could also make an educated guess at what you're looking for (e.g. the activity API may suggest the user is likely to be going to lunch, so the application could initially show nearby eating establishments instead of requiring the user to search for them).

Challenges/issues: primary challenge is developing an algorithm which can make reasonable estimates as to the user's activity (or attempting and then justifying why such an algorithm is not feasible, and investigating requirements or alternatives), and would form the bulk of the project. Sub-challenges within this include: researching/implementing machine learning techniques so the algorithm can take previous behaviour into account, processing data from 'messy' inputs such as mic/camera, and designing an API that would enable third-party app developers to easily make their applications activity aware.

Approach: data from sensors would need to be processed (e.g. mic input processed into a figure for ambient noise level in dB). The combination of this processed data would then need to be fed into an algorithm (possibly a neural network) to determine likelihood of various activities. There would need to be some mechanism for users to correct or train the system (at least initially), and it's possible that this data could then be shared to other users of the api/application.

2 Aims and Motivation

The primary aim of this project is to create an application for the Android platform that can sense the user's context in some fashion. This application will have a public interface which will allow other applications written by third party developers to read and receive updates about the user's context.

The ease of access to location aware services in modern smartphone platforms has lead to a surge in the number of applications which improve their utility or behaviour by integrating location information. It stands to reason that if additional context information were available, developers would be able to take advantage of this and further improve the utility of their applications. This, in turn, would increase the productivity of the end-user.

As discussed inII, a large amount of existing research has been done on context-aware devices, and specifically on activity-aware systems. There have also been some limited implementations for smartphones. Unfortunately, the end product of most of this research is not suitable for deployment or use in practical, every-day circumstances. This project aims to produce a working prototype which can be used on a day-to-day basis on an Android smartphone without significantly degrading performance.

3 Issues and challenges

One of the main challenges for this project will be accomplishing accurate and useful context-awareness without significantly hindering the battery life or performance of a typical device. Existing algorithms tend to be extremely verbose, sometimes performing upwards of thousands of calculations per classification; on a mobile device this is likely to severely cripple battery life.

The problem of battery life affects all areas of the project - from how often data is collected, how the data is then analysed, and which sources of potential data are consulted. A large amount of time will need to be spent analysing the various potential data sources and establishing whether or not the cost in consulting them is worth the reduction in battery lifetime and any gain in the reliability or accuracy of context information.

The aim of the application is to provide the context data to third-party applications, so another challenge will be designing an appropriate interface which will allow applications to query and receive updates about the user's context. Consideration will have to be given as to any security measures (such as access control) which may need to be applied in order to protect user privacy.

4 Structure

Part II summarises some of the current work in the field of context-aware systems and activity inference. Part III explains in detail the techniques developed to classify user activity from accelerometer data, applications developed to facilitate this, and analyses data collected from users to determine the accuracy of the activity classification algorithms. Parts IV and V deal with potential context information from other sensors such as microphones and cameras, and the contextual value of the user's location, and how interesting places can be inferred.

Part VI introduces the Context Analyser, the primary output of this proejct, and its suite of related applications. The project is evaluated in part VII and conclusions drawn and future work considered in part VIII.

Part II. Background

Context- or Activity-Aware devices is an area currently under lots of research. There are many and varied applications of activity-aware devices, ranging from personal fitness and healthcare to training factory workers or merely playing music.

While this research is going on, there has been a huge expansion in the ownership, use and power of mobile telephones. Mobile telephones are so ubiquitous and now come with such a large sensor platform that they are the obvious choice for implementing activity-aware technologies for use in day-to-day life.

This project aims to make a context-aware API available on an open mobile platform, which will enable developers to start adding context-aware functionality to their applications without the extremely large overhead of writing a logger and classifier themselves, or re-engineering the application to use an existing context-aware framework if one is available.

5 Applications

There are many documented applications of activity-aware systems, and current research efforts which bring the technology to mobile telephones will only serve to lengthen this list.

The canonical example for activity-awareness, especially on mobile telephones, is modeling the user's "interruptibility"[29, 23]. This allows the software to know whether it's appropriate (or "polite") to disturb the user, and can advise the user's contacts when they are busy. It can also be used to create a "smart answering machine" [15] which can selectively direct calls straight to an answering machine if the user is engaged in an "uninterpretable" activity and the call does not appear to be important. These allow the user's mobile telephone to better approximate human behaviour - when approaching someone in person it is normally quite easy to determine whether it would be polite or necessary to disturb them, based on their demeanour, activity, and the urgency of your request; when picking up the telephone it is not possible at all without assistance from an activity-aware system.

The current implementations of these ideas have several problems, however. The more interesting research [15] requires a static camera fixed in an office to observe user behaviour, instead of implementing it directly on a telephone, which obviously constrains its usefulness. Of the two solutions actually targeted at mobile telephones, one[29] requires bulky custom hardware which the user must carry on their belt, and the other [23]does not expose an API to other applications and only surfaces the context-aware functionality in two small applications whose focus is on social interaction rather than improving the user's experience of the telephone locally. This project will aim to bring the ideas of these to generic hardware (an Android mobile telephone), and to provide an API which other applications can harness.

One use particularly suitable for mobile phones is dynamic adaptation of the device's settings based on the user's current activity and context[26]. When a user is walking the device can dynamically increase the font size to make it easier to read with an unsteady hand, and correspondingly decrease it when the user is stationary. In a similar fashion, the brightness of the backlight can be altered based on the ambient light level, and the ringer volume altered according to the noise level. Unfortunately this research did not progress beyond a feasibility study and was implemented on a Nokia 6110, which is severely outdated by today's standards.

Another popular area for activity-aware systems is in healthcare. Such systems can be used to monitor vulnerable people as they go about day to day activities to ensure that they're not in trouble - several systems[30, 19] can be used to monitor elderly persons and summon help if it is detected that they have fallen. Another healthcare application[32] allows nurses to remotely monitor the activities of their patients in a hospital ward, allowing them to respond to problems and keep up-to-date with their patients' well-being while not physically present. Activity-aware applications have also been used to try to encourage users to be more healthy; one novel application records the day-to-day fitness activities a user performs and uses this as a basis for a virtual "garden" that blossoms or wilts according to how much the user works out in a week[8].

As well as monitoring activities which the user is familiar with, activity-aware systems can also be used to assist users in learning new activities. One application[31] monitors the activities of trainee workers in a car manufacturing plant, and helps to provide a link between theoretical classroom-based training and practical work. The activity-aware system can offer advice to the workers that's directly related to the current task they're performing, and can even monitor their activities for compliance with procedures and give them a score afterwards.

While the research into healthcare and training applications present novel uses of activity-aware systems, the applications themselves are not really applicable to a mobile device or the scope of this project. The research does, however, describe the techniques used in those applications for activity classification and should prove useful in that respect.

Other areas of research include making activity-aware suggestions to the user [5], or issuing reminders or alerts based on the user's activity [25]. One example of the latter is an activity-aware system that detects when the user is making coffee, and plays a sound on a remote computer to alert thirsty coworkers to the fact. Sound isn't only limited to alerts, however: the XPOD[9] project is an activity-aware music player, which tailors the music being played to the user's current activity based on their past ratings. This type of activity-aware device presents a much greater level of personalisation than previously possible, and making this type of customisation available to mobile telephone users and application developers will surely result in many new applications.

6 Inferring activity

There are three general phases in most context-aware systems: a sensing component, which reads or receives raw sensor data relating to the user's environment or activity; a feature extraction component, which analysis the sensor data and identifies a set of features from that data; and a classification component, which uses the extracted features to reason about the user's activity[7]. Each of these components will be expanded on below. Depending on the method of classification, some initial or continuous training may be required, and this is also considered below.

6.1 Sensors and devices

At the basis of activity-recognition are the physical hardware sensors. The most commonly used sensor is the accelerometer, which outputs the acceleration of the sensor[1] along a certain axis. There is extensive research on using accelerometers to classify activities such as walking [18, 11] (including whether or not the subject is walking on flat ground or up and down stairs [6]), running [11], falling [18], sitting [18, 11], cycling [11], etc.

Smartphones also come equipped with a microphone and GSM stack (prerequisites for a telephone conversation!), and commonly a camera, geolocation API (usually backed by GPS) and Bluetooth stack. With the exception of the latter two, these types of sensors are not particularly well explored for their use in context-aware systems at present. It is easy to reason how each would be useful - a microphone can reveal the ambient noise, which could indicate the difference between sitting in a library and a bar; the camera likewise can reveal the lighting conditions (if the device is not in a pocket or bag). The GSM stack can provide rough location information and also a signal strength to one or more cell towers; the signal strength will vary both with the user's proximity to the cell tower and the environment around them - being inside will degrade the signal more than being in open air, for example - so may provide vital clues to a context-aware system. One aspect of this project will be to research how the microphone, camera and GSM stack can be used to enhance existing activity classification algorithms.

Current research on location information and Bluetooth device proximity is summarised in section 8 (p17) and 16 (p30) respectively.

6.2 Feature detection

It is not possible to reason directly about raw sensor inputs, so the next step in inferring activities is to extract useful *features* from the raw input. Features are usually mathematical properties of the input data, such as the difference between the minimum and maximum data point in a given time frame. Most classifiers use an extremely large number of features - [13] detect 562 different features from their inputs.

Some of the more commonly used features in activity-recognition systems are: mean, standard deviation, energy, entropy, correlation between axis, and discrete FFT coefficients[16]. Obviously, not all features are of equal value. FFT coefficients are generally very good indicators of activity, but the ideal coefficients and window sizes vary depending on the exact activity that is being

[1] and thus the device to which it's attached, and therefore the person using the device

detected. Likewise, the choice of other features to give the best recognition rate varies depending on the activity being detected[16].

As the sensor data is received continuously, it needs to be partitioned somehow before features are extracted. Most implementations use a sliding window approach with a 50% overlap between windows[3]. A window size of 10 seconds with a 50% overlap would result in one set of features being computed every 5 seconds. The window size is normally selected to correspond to a pre-defined number of samples to enable fast computation of certain features - most notably FFTs[3].

One challenge will be determining a set of features that are robust enough to perform activity analysis on, but are sufficiently inexpensive to calculate continually on a mobile device, where CPU speed is limited and excessive usage results in undesirable higher battery consumption.

6.3 Training

In order to meaningfully classify and label activities, some kind of training generally needs to be performed beforehand. The choice of classifier affects how much offline analysis has to be done on the training set, and whether or not it can be adapted at run-time.

One might expect that training would best be performed in a controlled environment, to reduce external influences on the user, but subjects in a laboratory setting are much more self-conscious about their movements, and this manifests itself in the data collected. Walking in a laboratory tends to produce acceleration data showing a consistent gait cycle which can be split into distinct phases, whereas walking in an uncontrolled setting produces data showing large fluctuations in gait phases and length. This means that classifiers trained on laboratory data may achieve a much lower accuracy when deployed in natural conditions[3].

6.4 Classification

The classification step involves feeding the features for frame into some kind of machine learning algorithm which can, using training data[2], determine which activity the feature-set most like represents. There are many different algorithms that can be used to perform the classification, some of which are discussed below.

6.4.1 Decision trees

Decision trees are possibly one of the simplest approaches possible[15]. A tree is constructed such that each node contains a test function, with branches for each possible discrete outcome of the function. This allows data to be classified with a “divide and conquer” approach. While high accuracy is possible in some circumstances[15], there are several drawbacks to decision trees: a plain decision

[2] and any offline analysis made of that data

tree has no way to model uncertainty - in an activity-aware system there will always be a degree of uncertainty as to the classification, and being able to measure this is an important tool. They also have an inductive bias which leads to a preference for the most general solution, and in most cases this generalisation causes many false results[28].

Decision trees require the structure of the tree and the test functions for each node to be determined during training. They do not lend themselves to minor on-the-fly modifications or new activities that are not part of the training set.

6.4.2 Neural networks

Neural networks are based on an extremely simplified model of the brain. The network consists of layers of neurons, and each neuron performs a simple arithmetic operation on its inputs. This normally consists of taking each of its inputs, multiplying it by a weight, and then summing all of the weighted inputs together; the resulting figure then becomes the neuron's output, and the input to one or more nodes in the next layer.

A network consists of a layer of input neurons, a layer containing one or more output neurons, and one or more layers of "hidden" neurons in between. The number of "hidden" layers, and the number of neurons within those layers must be chosen before training of the network begins. The training process will then determine the weights for each link in the network. The choice of number of layers poses a problem when designing a network, as too small a number can cripple the power of the network, but too large can cause it to be too expensive to evaluate and can possibly lead to it memorising erroneous data[9].

Neural networks, however, do provide good accuracy and could potentially (although not easily) be modified on-the-fly to cope with new activities.

6.4.3 Genetic algorithms

Genetic algorithms use the principle of natural selection to 'evolve' a solution to a problem. A set of random solutions are created, and a pre-defined fitness function is used to determine their relative worth. The best solutions are then combined together to produce the next generation of solutions, in a manner roughly analogous to reproduction in animals. Small "mutations" are also introduced into each generation to counter the effect of local maxima being reached.

Genetic algorithms can be combined with other techniques such as neural networks - the weights in the neural network can be "evolved" using genetic algorithms to create a neural network which is good as satisfying the fitness function.

The drawback of genetic algorithms is the need for a fitness function - the network will only ever be as good as the fitness function, and if you have a way to define what makes a good network you could in most cases hardcode the solution instead of evolving a network to satisfy it.

6.4.4 Instance-based learning

Instance-based learning (IBL)[34] algorithms are a class of "lazy" algorithms. They perform classification based on previously observed instances that have already been classified. There is no training required for IBLs, they're extremely adept at adapting to new scenarios, and they have a very low error rate[9] which makes them ideal for activity-recognition.

One particular type of IBL algorithm which is frequently seen in activity-aware research is the K-Nearest Neighbour (KNN) algorithm[12]. With the KNN algorithm, each sample is treated as a vector, and the distance[3] between the sample and the existing instances is calculated. The sample is then classified according to the classification of the majority of its k nearest neighbours.

One drawback of IBLs is that each new instance tends to be remembered for future use, which eventually results in large amounts of memory consumption and complexity when comparing distances of new samples. This can be partially overcome by only storing instances which would affect the classification of new samples[34].

The KNN algorithm can be easily extended to support dynamic classification of new types of activities - if a sample is not within a certain distance of sufficient other samples, it can be classified as a new type of activity.

6.4.5 Conclusion

There are numerous machine learning algorithms available and suitable for use in activity classification tasks. There has been a lot of research into their use, and all of the algorithms discussed have produced good results. Because of the lack of need for any training, however, the K-Nearest Neighbour algorithm appears to be the most promising for a mobile device. Any algorithm that needs explicit training prior to classification would almost certainly require a desktop application or a remote service to analyse the data, as it typically requires large amounts of memory and expensive computations. This either makes the application extremely cumbersome for the user (they have to connect their phone to a computer, transfer a file, obtain and run a separate application, then transfer some file back), or puts a large resource burden onto the distributor (having to remotely analyse all of the data from all users would require dedicated hardware for any more than a few users).

7 Mobile telephones

It's hard to overstate the ubiquity of mobile telephones at present. In 2003, over a billion mobile telephones were sold - six times as many as the number of personal computers[10]. In 2007, this same figure describes the number of cameraphones sold[24], clearly representing a substantial growth in sales and advancements in the technology. In fact, mobile telephones are the fastest adopted

[3] the euclidean distance is usually used, but any metric will suffice

technology in human history[10]. This ubiquity, coupled with the fact that mobile telephones are comfortably carried around on a daily basis by most of their users, makes them a very attractive alternative to more traditional platforms used for activity-aware research, which typically involved bulky or inconvenient apparatus that was expensive to manufacture[27] and made users very self-conscious.

7.1 iPhone

There have been several published works related to activity-recognition on the iPhone. The similarity between iPhone and Android platforms means that many of the concepts developed for or used on the iPhone are applicable to both.

7.1.1 iLearn

iLearn[27] is a suite of three tools - iLog, iModel, and iClassify - which together allow for real-time classification of low-level activities. iLog is run on the user's iPhone and allows the user to specify which activity they will be performing. The application then records raw sensor data from the iPhone's three-axis accelerometer and 124 features computed from this data in real-time. The data is then stored on the device, annotated with the selected activity.

The training data collected by iLog is then transferred to a desktop computer where iModel uses a Naïve Bayesian Network (NBN) to create a model which can be used to classify future input. The choice of NBNs was based on their ability to classify a set of trial data correctly, and the low computational cost of classifying data once the model has been generated.

Once the model has been created, it is transferred back to the device where it is used by iClassify. This provides an API for other applications, and allows them to register for a callback which it publishes the user's current activity to every second.

Unfortunately, neither the source code nor the API are published. The inability to run background processes on the iPhone suggests that any "API" would have to be more like a framework where the third-party developer has to re-engineer their application to use the iClassify application as a base. This is undesirable as it makes it extremely difficult to adapt existing applications to use the activity-aware API, and is a very cumbersome way of providing what could be a very minor piece of functionality for the application.

7.1.2 Evaluation

[20] present an evaluation of the iPhone for use in "people-centric sensing applications". One of the major drawbacks highlighted is that the iPhone does not support applications which run in the background. This means that any application wishing to perform continuous real-time activity detection would need to run as a foreground process, preventing the user from using the device for any other function.

The research also shows that the computational compatibility of the iPhone is more than sufficient to perform the necessary calculations for a typical activity-recognising application, which suggests that any modern smart phone would be capable of these.

7.1.3 Multitasking

Since the evaluation presented above was written, Apple have announced a new version of the iPhone OS which supports pseudo multi-tasking. However, this form of multi-tasking only allows a limited set of predefined functions to be performed, such as playing audio or monitoring location. This still prevents continuous real-time activity detection from taking place in a useful fashion.

Further, because of the homogeneous hardware and memory model employed by the iPhone, applications typically make assumptions about the amount of free memory that will be available to them. This prevents any real implementation of multi-tasking, as background apps would infringe on this fixed amount of free memory. It is therefore unlikely that future updates will enable true, uninhibited multi-tasking, as it would break compatibility with a lot of existing applications.

7.2 Android

While the Android platform is relatively new, it is rapidly gaining market share on the more established mobile operating systems. A December 2009 survey[1] shows that 21% of respondents want their next smartphone purchase to run Android, a 350% increase from the same survey conducted three months prior. This is compared to the iPhone, which dropped 4% to 28% in the same time period. Gartner, a respected IT research firm, predicts that by 2012, Android will be the second most popular mobile operating system globally[2].

In addition to its rapidly increasing popularity, the Android platform offers several advantages over the iPhone platform. Most notably is the ability to run background processes (called SERVICES), which will allow a classifier application to run without interfering with the user's normal use of their mobile telephone. In addition, the Android OS provides access to the Bluetooth and GSM stacks, allowing for data from both to be used for activity detection.

The ability to run a background process will enable a proper API for sharing activity data with other applications, which will allow third-party developers to make their applications context-aware with relatively little work on their part. This is extremely desirable as it will allow rapid prototyping of applications, which will hopefully lead to innovative new uses of activity classification.

While it is purported[11] that there is research being done on bringing activity-awareness to Android platforms, there does not seem to be any work published on this matter or any applications available to support it. While there a small number of self-proclaimed "context-aware" applications for Android, this context is almost exclusively limited to geolocation. This project will therefore produce one of the first publicly available activity-aware applications for the Android platform.

8 Location analysis

Location-based services are currently undergoing an "explosion"[4], thanks to improvements in technology, and greater openness on the part of service providers and handset manufacturers. All modern smartphone platforms have a geolocation stack, usually backed by a GPS chipset and in most cases augmented with either a database of known cell tower locations, or a map of known WiFi network identifiers and locations, or both. The two databases allow for rough geolocation when GPS is not available, or for greatly decreased lookup time when a GPS lock is available.

However, while the geolocation stack is a rich source of data, it is a poor source of information. A latitude/longitude pair may describe the user's exact location, but a user would be hard-pressed to tell the difference between the latitude/longitude of their home, place of work, or of somewhere in between the two with no real significance. A great deal of research has therefore been devoted to detecting meaningful locations from GPS traces.

"Place recognition" has two phases: learning and recognising. An initial learning phase analyses a sensor log and segments the data into places where the device is stable (stationary), and designates this as a "waypoint". It then merges "waypoints" that appear to identify the same place being visited multiple times. The second phase uses these learned waypoints to recognise when the device is revisiting a place, and therefore also when the device is not visiting a previously known place (for example when it is moving between two)[14].

Unfortunately, quite a lot of research into location analysis uses GPS "blackspots" to identify useful places[22, 17]. With older GPS chipsets, the satellite signal would be lost when the user entered a building, and this allowed an inference that the current location was probably a place of interest. However, modern GPS chipsets receive a signal in most indoor locations. It is possible that a decrease in signal strength or number of locked satellites may still occur, or that GSM signal strength could be used instead, but these ideas have not been widely explored at present.

There is, however, plenty of research relating to the use of location data outdoors. One application[17] learns not only the user's frequently visited places, but the method of transport used between them and the typical routes taken. It can then offer instructions showing the user how to go from place to place, or issue alerts if the user appears to be going the wrong way (by getting on the wrong bus, for instance). The ability to correctly infer the user's destination would be extremely useful in a context-aware system: a user walking to do their grocery shopping is almost certainly going to want to interact with their phone differently than a user on a bus going to work.

9 Bluetooth

The user's context depends on not only what they are doing, where they are doing it, but also who they are with. Sitting and eating lunch with a manager is quite a different context to sitting and eating lunch with a spouse. It would therefore be desirable to be able to identify between different people when performing context analysis.

One of the few ways that a mobile telephone can identify other people is by searching for *their* mobile telephones. This can be done by scanning for Bluetooth devices, which involves broadcasting a "device inquiry" message; if a device chooses to answer the inquiry, it discloses its unique MAC address and device class[4]. Unfortunately, this requires the person to not only be carrying a mobile telephone, but a Bluetooth-enabled model, and for them to have configured their device to have Bluetooth enabled and to be "visible". A study in 2004[10] found that only 1 in 150 people had such a configured device on a university campus. This figure will undoubtedly be greater now, and may well be greater when in public, but it highlights that only a handful of people may be detectable via their Bluetooth devices.

A study in 2006[21] used a similar technique to monitor the social context of the user, introducing the idea of "familiar" people, "unfamiliar" people and "familiar strangers". These labels were applied based on the number of times their Bluetooth devices were detected [5]. While the definition of "familiar" and "unfamiliar" are quite obvious, "familiar strangers" is a new class of people used to describe those who the user encounters repeatedly, but doesn't interact with. This may include neighbours that are passed on the street, or fellow commuters on a journey into work. The number of people in each of those groups (and any changes in those numbers) can be used to infer how "comfortable" the user feels with their social context, and whether their current activity is part of a normal routine or is novel.

This research has, to date, not been readily combined with activity-aware applications, and this project will aim to integrate the results of Bluetooth scanning with "classical" activity classification techniques and to evaluate whether it provides any benefit.

10 Power management

One major consideration when deploying an application on a mobile device is the amount of power it will use. An application constantly polling any one sensor can reduce battery life significantly, and an application which kept all available sensors active (in addition to doing CPU-heavy analysis on them) would drain the battery in a typical smartphone in a matter of hours. A context-aware application is not very useful for a user if they can only use their telephone for an hour or two before it needs recharging!

One solution[33]is to only use one or two sensors to monitor the user's activity until it appears to be transitioning. For example, if the user is believed to be walking, the application only needs to periodically check either the accelerometer (to confirm the user is still making walking motions) or GPS (to confirm the distance traveled is still consistent with walking) to know that their activity has not changed. As soon as the user's behaviour becomes inconsistent with

[4] the device class tells us whether the device is a computer or a mobile telephone, for example

[5] and by extension the number of times the user had come into contact with them

walking, the application can bring other sensors online until it has successfully reclassified the activity, and then resume monitoring with minimal sensors.

Another option[33] (which can be used in conjunction) is to only enable sensors for a short amount of time, and then sleep for a period before reactivating them. The "duty cycle" suggested for accelerometers is 6 second of sensing followed by 10 seconds of sleeping. The six second window is enough time to allow for capturing a full range of motion (several complete strides) if the user is walking or running, and then the ten second sleep stops the accelerometer using battery power until the next cycle. This process obviously means that a sudden switch in activity will not be noticed immediately, but a delay of a few seconds is acceptable as most activities will last for minutes or longer.

The battery life on modern smartphones rarely exceeds 24 hours of typical use, so it is extremely important that any applications developed for this project does not significantly reduce this. A balance between prompt detection and notification of activity changes and battery use by sensors and processing algorithms will need to be found.

Part III. Activity classification

11 Sensor Logger application

The primary component of context that this project aims to expose is the user's activity. The main factor in determining a user's activity is the data retrieved from the device's accelerometers. This gave rise to the project's first published application, titled Sensor Logger. The first version of Sensor Logger consisted of a single activity containing a large amount of text describing the project, an editable text field where the user could name their activity, and a button which initiated logging.

Once the user tapped the 'Start' button, the application launched a service which registered with the device's SensorManager and requested fast updates from both the accelerometer and the magnetic field sensor. Every 50ms the last value received from each axis on each sensor was written along with a timestamp to a file on the device. After 1024 samples were collected (a total of around 51 seconds), the service launched an uploader service and terminated itself. The uploader service read the file from the device's flash memory, opened a HTTP connection to the project website, and submitted the data to a PHP script. The PHP script in turn stored the data in a MySQL database. Figure 1 on page 21 shows the relationship between the various components involved in the Sensor Logger application.

11.1 Market and user input

The Sensor Logger application was made available on the Android Market, under the name 'Sensor Logger Test'. The description briefly outlined the aim of the project and emphasised that the Sensor Logger was a data gathering tool and didn't really provide any utility to end users. Despite this, over 1,000 submissions were received from anonymous users. While some of these did not provide any use for the reasons discussed in Section 12, a number were manually classified and used to generate the model used in the final application.

Version 0.2.0 of the Sensor Logger application introduced functionality where it classified the activity on the device before asking the user to name it (see Section 14). This allowed the submitted data to be augmented with an extra field saying what the activity classification algorithm thought the activity was. If the user confirmed that the activity was correct, the manual activity annotation was set to 'UNCLASSIFIED/NOTCORRECTED'.

Figure 2 on page 22 shows a breakdown of all the results that were received from version 0.2.0 or later of the application. The raw data is included in Section H. Nearly half of all the submissions were classified correctly, and only 15% were classified incorrectly. A large proportion of the submissions either had no text at all or had an activity that didn't make sense, such as “sjxjxgzog” or “it is a cat!”. There were also 30 submissions where the user had annotated the data with an activity that makes sense, but which isn't supported by the Sensor Logger. The majority of these were 'Sleeping' - but it is unclear in these circumstances what

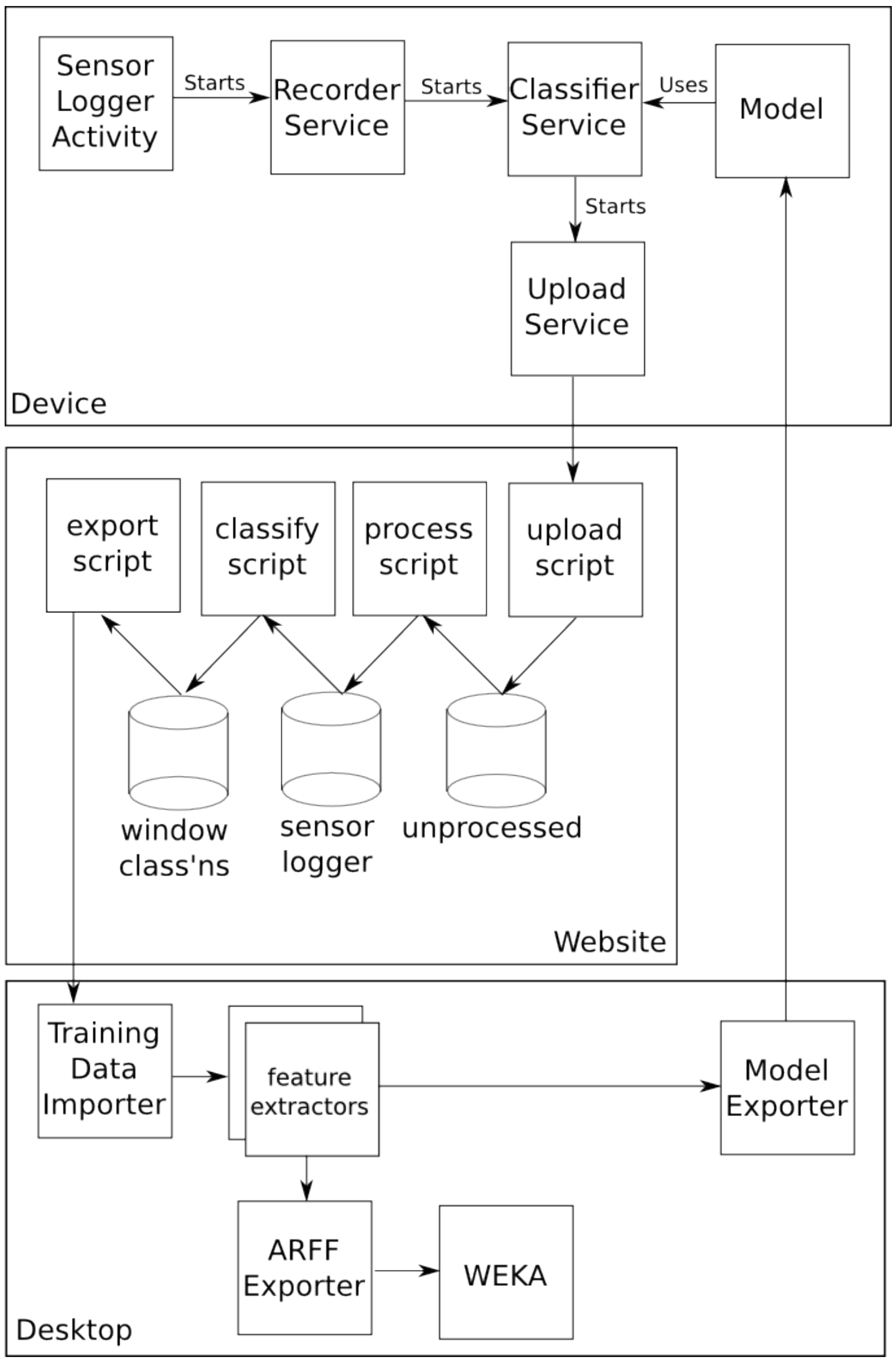


Fig. 1: Sensor Logger component diagram

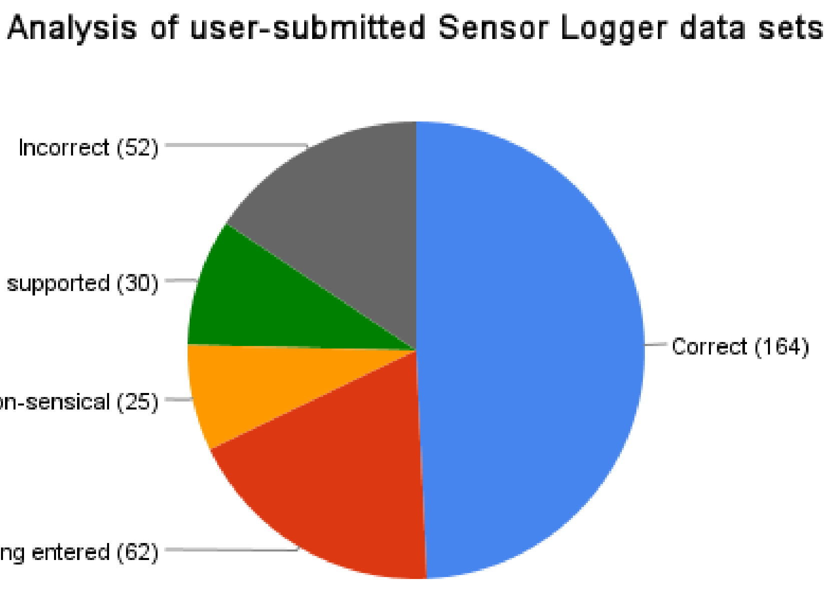


Fig. 2: Analysis of user-submitted Sensor Logger results

the user is actually doing with their device; if it is left on a bedside table, for example, then there is no way to distinguish the actual activity of the user.

Figure 3 on page 23 shows a breakdown of the results deemed incorrect. A large majority of these incorrect results occurred when the user was sitting down, and the Sensor Logger incorrectly classified them as being in a vehicle of some sort. Of the remaining incorrect entries, 15% correctly classified the correct top level in the hierarchy (such as 'VEHICLE' or 'WALKING') but then misclassified further levels; the remaining 25% incorrectly classified this top level as well.

11.2 Exception handling

Early user feedback on the Android Market indicated that the Sensor Logger application was "Force Closing". This is a reference to the dialog that appears when an application throws an unhandled exception and stops running. The Market provides no facilities to engage with users, and there were no force close issues present on either the Android emulator or several physical devices the application was tested on, so it was difficult to determine the cause.

In order to gain more data on this issue, an UncaughtExceptionHandler was written and appropriately registered with the application's thread. The exception handler is invoked by the Thread class any time a thread dies due to an unhandled exception. The implementation for the Sensor Logger application copied the details of any exception (including the reason and full stack trace)

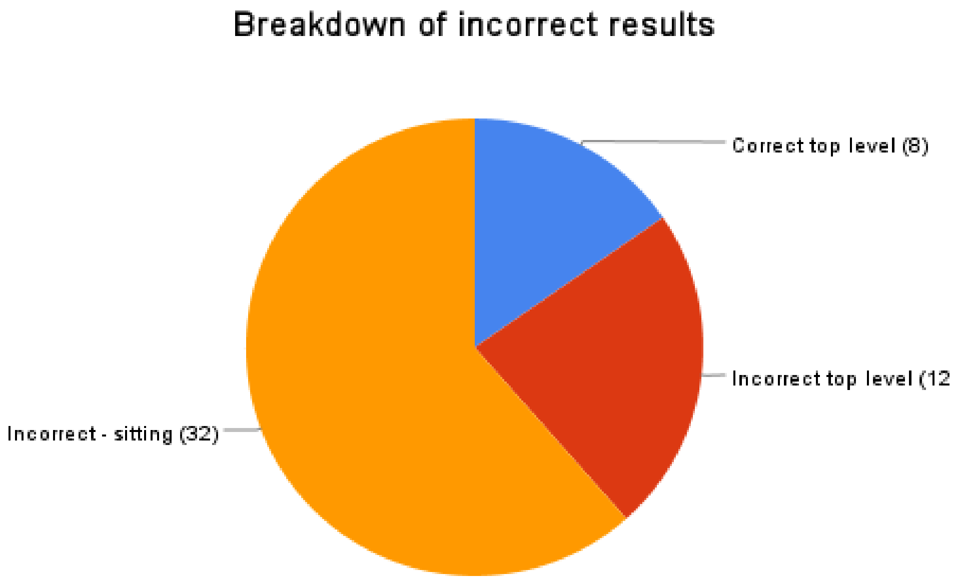


Fig. 3: Breakdown of incorrect results

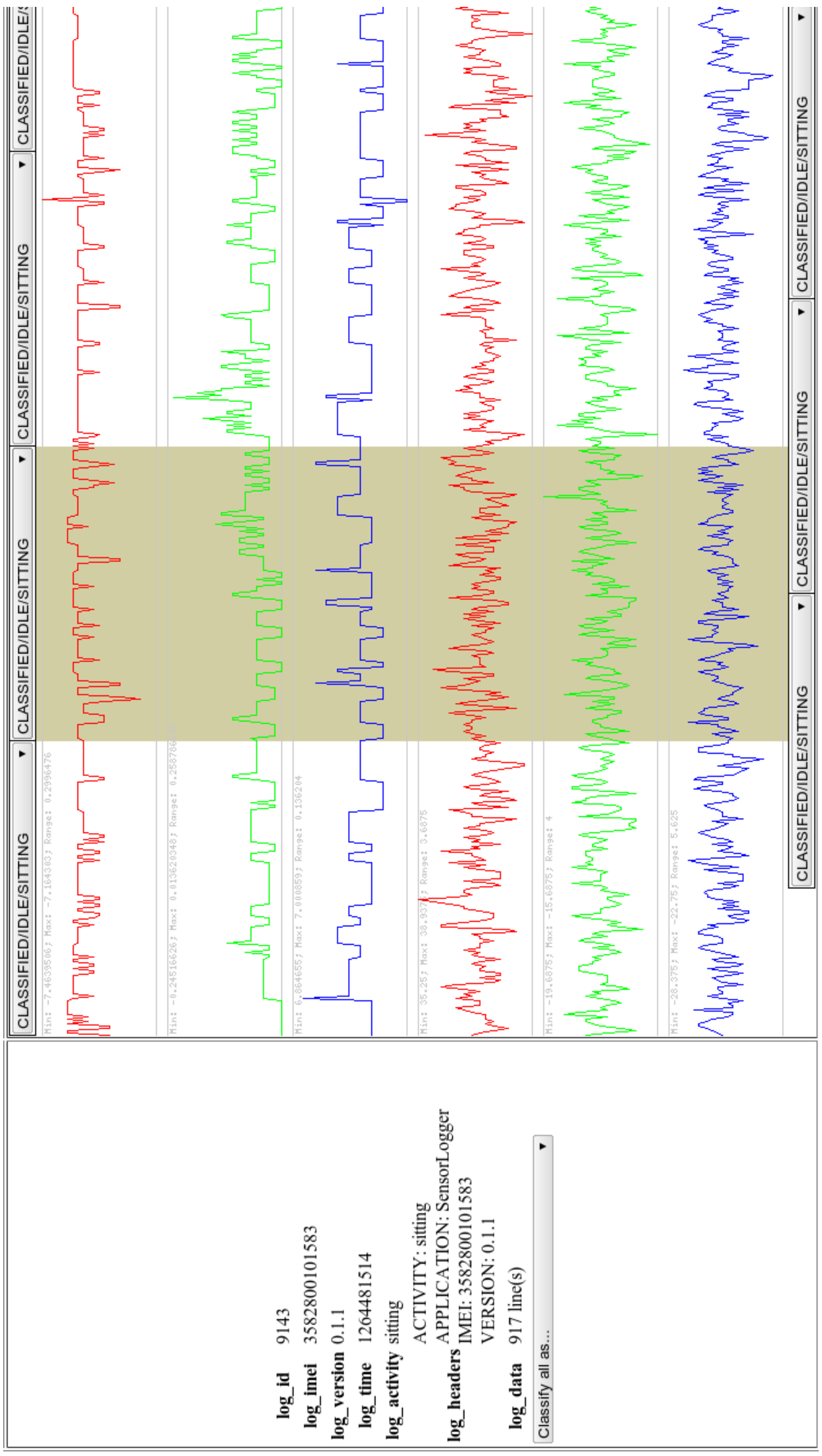
into a file, and then uploaded the file with some meta-data to the same website which was setup to handle uploading of accelerometer data.

Soon after an updated version of the Sensor Logger was published including the new exception handler a report was uploaded. The stack trace indicated that the problem was a NumberFormatException when trying to convert the device's IMEI number into a long. After some brief research it became apparent that CDMA devices do not use IMEI numbers, but instead MEIDs (Mobile Equipment Identifier). MEIDs are hexadecimal instead of numerical, so obviously cannot be converted directly to a numeric type. The code was adjusted to decode MEIDs properly and a new version of the application published, and the error reports ceased.

From a development perspective, the utility of being notified directly of exceptions is immense. It is very difficult for users to find the details of an exception following a force close, so extremely unlikely that anyone will report problems in sufficient detail for them to be fixed. The exception handling code was therefore abstracted into a common class, and included in every application released as part of this project.

12 Manual classification

Once the sensor data was logged in the database, a web interface provided a graphical representation of the acceleration and magnetic field readings. It also


CLASSIFIED/IDLE/SITTING
CLASSIFIED/IDLE/SITTING
CLASSIFIED/IDLE/SITTING
CLASSIFIED/IDLE/S
CLASSIFIED/IDLE/SITTING
CLASSIFIED/IDLE/SITTING
CLASSIFIED/IDLE/SITTING
CLASSIFIED/IDLE/SITTING
Min: -7.4639506; Max: -7.164303; Range: 0.2996476
Min: -0.24516626; Max: 0.013620348; Range: 0.258786
Min: 6.864655; Max: 7.000859; Range: 0.136204
Min: 35.25; Max: 38.9375; Range: 3.6875
Min: -19.6875; Max: -15.6875; Range: 4
Min: -28.375; Max: -22.75; Range: 5.625
Classify all as...

log_id	9143
log_imei	3582800101583
log_version	0.1.1
log_time	1264481514
log_activity	sitting
log_headers	ACTIVITY: sitting APPLICATION: SensorLogger IMEI: 3582800101583 VERSION: 0.1.1
log_data	917 line(s)

Fig. 4: Website for manually classifying windows

allowed a handful of authenticated users to manually classify overlapping "windows" of 128 data points. The classification portion of the website is shown in Figure 4 on page 24; it shows the first six overlapping windows for one submission, with the third window highlighted because the user's cursor is over the corresponding dropdown. Users of this system could define activities in a hierarchical fashion, starting with two root nodes - 'CLASSIFIED' and 'UNCLASSIFIED'. Early data submitted resulted in a classification hierarchy of:

- UNCLASSIFIED
  - PENDING (not yet manually assigned)
  - UNKNOWN (unable to determine actual activity from user label)
  - DNI (short for "Do Not Include", for windows which appear erroneous)
- CLASSIFIED
  - WALKING
    - STAIRS (not used in itself)
      - UP
      - DOWN
  - IDLE (not used in itself)
    - SITTING
    - STANDING
  - VEHICLE (not used in itself)
    - CAR
    - BUS
  - DANCING

It was observed that many samples had erroneous data either at the start or the end of the recording. This was a result of the user putting the device away in a pocket (as instructed) or picking it back up to check the results. A classification of 'DNI', short for 'Do Not Include' was therefore introduced which allowed the erroneous windows to be summarily excluded from later analysis. Later iterations of the Sensor Logger application included a 10 second delay at the start to give the user time to put the device away, and sounded an audible alert at the end, which reduced the number of records submitted with bad data.

Another problem with the user submitted data was that some activity descriptions didn't make sense. With the earliest versions of the Sensor Logger, a significant number of users entered their own name or a nonsensical string into the textbox labeled "Activity name". This suggested that users were either not reading or misunderstanding the instructions. A classification for 'UNKNOWN' was introduced to facilitate removal of the records where classifications couldn't be inferred from the activity name.

13 Feature extraction

Once a reasonable sample of data had been recorded, the PHP script was modified to allow exporting of all classified windows. It produced a plain text file containing the sensor readings and timestamp for each of the 128 points in each window, as well as the manually-applied classification. It did not include information as to which windows were from the same sample, or the original user-supplied activity name. A small sample of this data is included in Appendix Section F.

A Java program was written which imported the exported data. A series of 'feature extractors' were written. These calculated the:

- maximum
- minimum
- range
- median
- mean

Each of these extractors was run over the set of 128 data points from each axis on each sensor, giving a total of 30 features. It was planned to add further features including Fast Fourier Transforms and energy, as suggested by multiple papers consulted in background research.

The program was modified to export the features and classification of each window in ARFF (Attribute Relation File Format). This is a format used by Weka, a popular data mining suite developed by the University of Waikato. A small sample of the ARFF data is included in Appendix Section G. The ARFF file was imported into Weka and analysed.

Inspection of a graphical representation of the correlation between mean and activity showed that in some samples the mean was negated but of a similar magnitude to other samples. This can be explained by the device being orientated in a different manner when the samples were taken - a static device would record a downwards acceleration of 9.8m/s if it is upright, but -9.8m/s if it is upside down. An extra feature extractor was therefore added which calculated the *absolute mean.*

With these features, it was found that Weka could correctly classify activities with an accuracy of in excess of 95% (measured by holding back $\frac{1}{3}$ of the training data) using a K-Nearest Network algorithm with $K = 1$. It was therefore apparent that more complicated feature extraction techniques such as Fast Fourier Transforms or Energy would not be necessary to achieve a very high classification rate.

A lot of time was spent attempting to reduce the already small set of features further. Every feature that had to be calculated would result in more CPU usage when implemented on the device, which would in turn reduce battery life by a greater amount. This was done by a combination of experimentation based on educated guesses and assumption, and Weka's built in "select attributes" functionality.

In the end, a similar accuracy was achieved using only four features. These were the absolute mean and range of the X and Y axes of the accelerometer. No data at all was used from the magnetic field sensor or the Z axis of the accelerometer. This is an very useful result as the features are extremely cheap to calculate, and almost trivial to implement on the device. No libraries are needed to perform advanced mathematical functions, and individual samples do not need to be stored in memory until a complete window is obtained - the minimum, maximum and total of samples from the two axes simply have to be recorded. This is, perhaps, the first time activity inference has been performed successfully with such a small number of features.

The application was again modified to read the exported data, calculate the four interesting features, and create a mapping of data points to activity. This map was serialised into a file using an ObjectWriter, which allows it to be read on both desktop platforms running a standard JVM, and the Android platform which contains compatible serialisation technology.

14 On-device classification

The serialised model was bundled with a new version of the Sensor Logger application, along with a completely redesigned user interface. The application now consisted of a series of activities: on opening the application, the user is presented with an introduction activity which explains what the application does and the aims of the project; when they click the 'next' button a 10 second countdown is displayed with the instruction to put the phone away repeated. During the data collection period an animated pattern of dots is displayed so the user knows the application is still working if they look at it. When data collection is completed, the device adds a notification with sound and vibrate settings, and displays another progress screen while a background service analyses the data. The result of the classification is then displayed prominently and the user is presented with two buttons - one to confirm the classification and one to reject and correct it. The on-device classification (and any correction made to it) is submitted along with all the data previously submitted. Finally, the user is then presented with a 'Thank you' message, and a unique link to the project website where they can view graphs of their activities.

15 Activity Recorder application

The Activity Recorder application was the second application published on the market. In contrast to the sensor logger, it only records the two relevant axes of the accelerometer, and all data is kept in memory instead of being written to a file.

The activity recorder consists of a background service which records 128 samples of sensor data once every 30 seconds. At all other times, the accelerometer is not accessed to conserve battery power. The samples are then classified using the same model as the Sensor Logger application, and the resulting activity is appended to a list. Consecutive samples which are classified as the same activity are merged together.

The user interface displays a list of activities, along with their start time and duration in minutes or hours.

15.1 Acquiring data while screen is off

The activity recorder brought to light a problem that was discovered to be present in the sensor logger as we, but hadn't been noticed at the time. When the device's screen was turned off, both applications stopped receiving sensor events. Some research and experimentation revealed that in order to continue receiving sensor data when the device went to "sleep", the application had to acquire a partial wake lock from the system's power manager. A full wake lock keeps the device fully awake with the screen on, whereas a partial wake lock allows background processing to continue while the screen is deactivated.

Once the applications were modified to acquire a partial wake lock, they both received sensor data as expected with the screen off. However, testing on Android 2.0 ("Eclair") devices still showed the original problem. The cause of this was a change in the functionality of the sensor manager introduced between 1.6 and 2.0 which prevents sensor data being received when the device is sleeping, regardless of any wake locks held. An issue raised on their official issue tracker revealed that the Android developers considered this a bug, and that it would be rectified in a future version. When Android 2.2 ("Froyo") was released, testing revealed that the issue was indeed fixed, and all applications worked as expected. There therefore exists an incompatibility with handsets running Android 2.0 and 2.1.

15.2 Aggregating classifications to smooth results

Another problem raised by the Activity Recorder application is that of occasional incorrect results. Because each data was only sampled for five seconds once a minute, it is possible that during those five seconds the activity being performed does not lend itself to proper classification of the user's actual, medium-term activity. For example, if a user is walking for 20 minutes and happens to stop several times to cross roads, samples while the user is stopped will (correctly) indicate that they are standing still, but for all practical purposes they are still engaged in the act of walking. Similarly, when traveling in a car it is possible for occasional samples to be misclassified as traveling by bus, or vice-versa.

To combat this problem, an activity "aggregator" was introduced. This accepts classifications and uses them to adjust a set of internal probabilities for each possible (sub-)classification. The algorithm increases the likelihood score of each component in a classification by a constant amount, and reduces all other components by a fixed ratio. This smooths out results, and allows for classifications not directly supported by the model when the data is unclear and alternating.

This behaviour is most useful when traveling by vehicle and classifications alter between “car” and “bus” - both contributed positively to the likelihood of “CLASSIFIED/VEHICLE”, and this ends up being the result presented by the aggregator.

Part IV. Other sensors

16 Bluetooth

Being able to identify who the user is accompanied by would be extremely beneficial in a context-aware system. One of the few ways to do this at present is to monitor the presence of other user's mobile telephones by scanning for visible Bluetooth devices. As discussed in Section 9, if sufficient devices are visible, users can be classified according to whether they are familiar, strangers, or familiar strangers. This provides a great deal of context to the user's activity.

Before this Bluetooth algorithm can be implemented, it needs to be determined whether there are a sufficient number of discoverable devices to make it worthwhile. As with any radio transceiver, a lot of battery power is consumed when Bluetooth is enabled and the device is actively scanning for others. This means that if there are insufficient discoverable devices, the cost of enabling and scanning for Bluetooth devices would outweigh the benefits. As mentioned previously, there is no use in a very accurate context API if the user's battery only lasts for a few hours.

To determine the utility of scanning for Bluetooth devices, a group of three volunteers were asked to manually enable Bluetooth and scan for devices using the built in functionality exposed in Android's settings screens. It was planned to create an application which could scan and classify devices appropriately but the Bluetooth API is only exposed to user-space applications in version 5 of the Android SDK, which corresponds to the "Eclair" or 2.0 release; at the time of the experiment, only one of the three volunteers was using a device for which Eclair was available. This also means that adding Bluetooth support would have a further cost of having to either limit access to pre-2.0 users, maintain two separate versions of the application, or spend extra time developing a solution which would attempt to use the Bluetooth API if and only if it is available.

Environment	Devices	People	Discoverable	Familiar	Familiar Strangers	Strangers
Car	0	1	0%	1	0	0
Bus	1	20	5%	0	0	20
Street	2	15	13%	0	0	15
Bar	2	12	17%	1	1	10
Office	0	6	0%	5	1	0
Office	0	4	0%	4	0	0
Home	0	1	0%	1	0	0
Home	0	2	0%	2	0	0
Lecture	5	80	6%	10	70	0
Meeting	1	6	17%	6	0	0
Restaurant	2	22	9%	1	2	19
Supermarket	2	1	200%	0	0	1

Tab. 1: Bluetooth scanning results

For the experiment, the users were asked to record the number of visible Bluetooth devices, their environment and the estimated number of people around them. They were also asked to classify the people into rough groups of "familiar" (such as co-workers, family and friends), "familiar strangers" (people they see regularly but are not particularly familiar with), or "strangers". The results are itemised in Table 1 on page 30.

The results show that in general there is a very poor proportion of devices that are visible. The results also suggest that Bluetooth would be a poor method for finding "familiar" people, as the situations with very few strangers generally had almost no discoverable devices. There are several possible reasons for the low proportion of devices: modern phones tend to default to having Bluetooth switched off for reasons of battery life and user privacy, and even when enabled the devices tend to default to being non-discoverable. In fact, Android devices will only allow the user to make the device discoverable for 30 seconds at a time - the setting is automatically reverted after this period. Another consideration is that the three volunteers are all involved in extremely technology-focused settings: one was a student studying a computing course, one a software engineer, and one a systems administrator. Whether or not these technology-rich settings would artificially inflate results (more technology means more discoverable devices) or deflate them (more tech-savvy users mean fewer needlessly enabling Bluetooth discovery) is unclear.

17 Microphone

Reading raw data from the Microphone on Android is relatively straight forward thanks to the built-in AudioRecord class. This allows raw data to be retrieved directly into an array of bytes, where it can then be analysed. A class was written which constructs an AudioRecord instance, retrieves a sample of data, and calculates the sound pressure level (SPL). The SPL can be used to measure the ambient volume of the environment the device is in - for example, a quiet room would typically have an SPL of 20-30 dB, a television might raise that to the level of 60 dB, and a busy road could be up to 90 dB. SPL is calculated using the formula $L_p = 20log_{10}(\frac{p_{rms}}{p_{ref}})$. p_{ref} is the reference sound level and is usually taken to be 20 micropascals, which is the limit of human hearing; p_{rms} is the root mean square pressure being measured.

While the class was able to correctly calculate the SPL, it is not used in any of the published applications. The original intention was to use SPL to assist in activity inference, but as shown in 11.1 the accelerometer alone achieves a satisfactory classification rate on its own. The large majority of incorrect classifications revolved around the user sitting down and, as sitting is such a universal activity (i.e., it's performed everywhere), knowing the ambient SPL will not help in this classification.

While there are certainly use cases for knowing the SPL (the most obvious being increasing or decreasing the ringer volume in proportion to the SPL to ensure it can be heard), most of them do not involve any other form of context that this project is aiming to expose. It was therefore decided to not include the SPL data in the main context API.

18 Camera

The aim of incorporating data from the device's camera was to determine the ambient lighting conditions. It could possibly then be inferred whether the user was in natural or artificial light, or in darkness. However on consultation with a group of five prospective users it became clear that at the times at which the device would be classifying context, it would almost always be inside a pocket or a bag. The only times the device would be able to detect lighting conditions would be as the user was using it (which is too late as the device should have already adapted to the user's context) or when the device had been left on a desk or shelf. It was therefore decided to not include camera data at this stage of the project.

Part V. Places

19 Use of GPS/GSM blackspots

As discussed in Section 8 a lot of existing research into identifying interesting places relied on the fact that with hardware that is now 5-10 years old, you could not reliably get a GPS signal indoors. Thus whenever the user remained somewhere without a GPS signal it was likely that they were spending time indoors, and therefore their current location would be noteworthy.

Unfortunately, modern GPS hardware is much more sensitive and can quite easily get a signal indoors in most circumstances. One possible solution to this was to monitor the strength of either the GPS lock or the GSM signal, and attempt to determine whether or not a difference was noticeable. An experiment was therefore conducted to record the GPS and GSM status for multiple indoor and outdoor locations.

Location	GSM strength	GPS satellites	GPS time to fix
Indoors	-101 dBm	10	<1 sec
Outdoors	-101 dBm	8	<1 sec
Indoors	-71 dBm	6	2 sec
Outdoors	-67 dBm	7	2 sec
Indoors	-67 dBm	11	<1 sec
Outdoors	-67 dBm	10	3 secs
Indoors	-67 dBm	10	<1 sec
Outdoors	-43 dBm	10	<1 sec

Tab. 2: GPS and GSM strengths

At four separate indoor locations, the third-party GPS Status application was used to observe the number of satellites the device was receiving a signal from, and the amount of time it took to acquire a fix on all of those satellites. The GSM signal strength as reported by the device's built in debugging tools was also recorded. The device was then moved outdoors to the nearest appropriate open area (footpath, outdoor seating area, etc), and the process was repeated. The results are summarised in Table 2 on page 33.

While the indoor GSM strength is lower than the corresponding outdoor strength in two out of the four samples, it is still significantly stronger than the weakest recorded indoor or outdoor signal. In the other two cases the signal strength was the same for both indoors and outdoors. The GSM signal strength therefore seems to be a poor indicator of whether or not the user is indoors - it would have to be continually monitored to detect a drop in signal strength, and the (limited) experimental data shows that this would have a 50% false negative rate. The experiment does not allow us to reason about false positives, but it is easy to imagine that many circumstances would give rise to drops in signal strength - the most obvious being moving away from the cell tower.

The GPS results show even less correlation between indoor and outdoor readings. In two out of the four locations, the indoor test identified more satellites than the outdoor test - the opposite to what would be expected. This could be because the indoor locations were typically above ground level - the extra altitude may be more beneficial to obtaining a lock than the obstruction of the building itself. It seems more likely, however, that the numbers are not correlated in any way, and a larger experiment would reveal it to be random variance. Similarly, the time taken to lock the satellites is typically very small, but in a few situations there is a noticeable delay. The location which resulted in a three second lock time was repeated after a thirty second pause and obtained a sub-second lock time, but the original data was included in the results.

The experiment, although extremely limited in size, shows that it is unlikely that GSM strength or GPS metadata would be useful in determining interesting places. Alternative methods therefore had to be considered.

20 Detecting places by time spent

Without information about whether or not the user is indoors, the next best method to detect interesting places is by monitoring the amount of time the user spends there. It stands to reason that a location where the user is only present fleetingly as they pass through is less important than somewhere they spend half a day.

The Android operating system provides two methods of determining the user's location - "coarse", which uses a cell tower and WiFi access point database, and "fine" which uses the device's GPS chipset. While "coarse" is less accurate than using GPS, it also uses significantly less battery power. Brief testing revealed that while GPS regularly obtained accuracies of below five metres, "network" location (using cell tower IDs and a built-in database) obtained accuracies in the region of 500 metres. Network location augmented with the built-in database of WiFi access point locations obtained accuracies in the same region as GPS. However, WiFi augmentation only works when the user has elected to enable WiFi on their device, and this puts a large drain on the battery.

As the most important places a user typically visits will be much greater than 500 metres apart, "coarse" location will was used for all location-related tasks in the project. Because of this large inaccuracy, places could not be represented by a single point but must have a radius to compensate for the inaccuracy. Based on earlier experimentation, this radius was fixed at 500 metres.

A new "place" is identified when the user remains within 500 metres of a point for at least 3 minutes. This figure is, hopefully, long enough to eliminate places where the user is temporarily held up (such as traffic lights), but short enough to include places the user visits but doesn't remain for a long length of time (such as supermarkets). When implementing this algorithm, care was taken to ensure that the length of time was defined as a constant and so could easily be changed if experimental data showed that it was either too long or too short.

Part VI. The Context Analyser and applications

The primary output of this project is the Context Analyser application, and several further applications which use the services exposed by the Context Analyser.

21 The Context Analyser

The Context Analyser itself is an application with a single service, a single activity, and four content providers. Its purpose is to monitor the user's context, analyse it as necessary, and expose the data to other applications which will actually do something useful with it. It reuses components developed for the Activity Recorder application to retrieve and process accelerometer input, and a LocationMonitor class which wraps around Android's built in location services. Persistent data such as places is stored in a SQLite database, with logic in a custom helper class which handles creation and maintenance of tables and allows interaction with the database without exposing SQL to the rest of the application.

21.1 Accelerometer and location components

Figure 5 on page 36 shows the key components used in the activity inference portion of the Context Analyser. The accel package handles low-level collection of accelerometer readings. The RealAccelReader class registers itself to receive sensor events and then makes the received data available through a public method. It also handles acquiring of a wake lock as discussed in 15.1 to keep the device sufficiently 'awake' to collect sensor data. AccelReaders are constructed by a factory so that they can be swapped out for test implementations if required, or when using an emulator which lacks real acceleration sensors.

The Sampler class handles automatic, timed sampling of a reader. Once started, it starts its AccelReader and then records a sample every 50ms until it has obtained 128 samples. As it retrieves each sample, it computes the minimum, maximum and sum of both axes that are needed for classification. When 128 samples have been recorded, it executes the run method of a Runnable provided at construction, so the owner can retrieve the results and analyse them appropriately.

The Classifier is identical to that used in both the Sensor Logger and Activity Recorder applications, and applies the K-Nearest Network algorithm to its given input and model. The ModelReader is a utility class which reads and deserialises the model bundled with the application.

The Aggregator handles aggregation of a stream of classifications, smoothing out occasional spurious results as discussed in 15.2. The AutoAggregator is an extension of this which automatically uses a Sampler and a Classifier

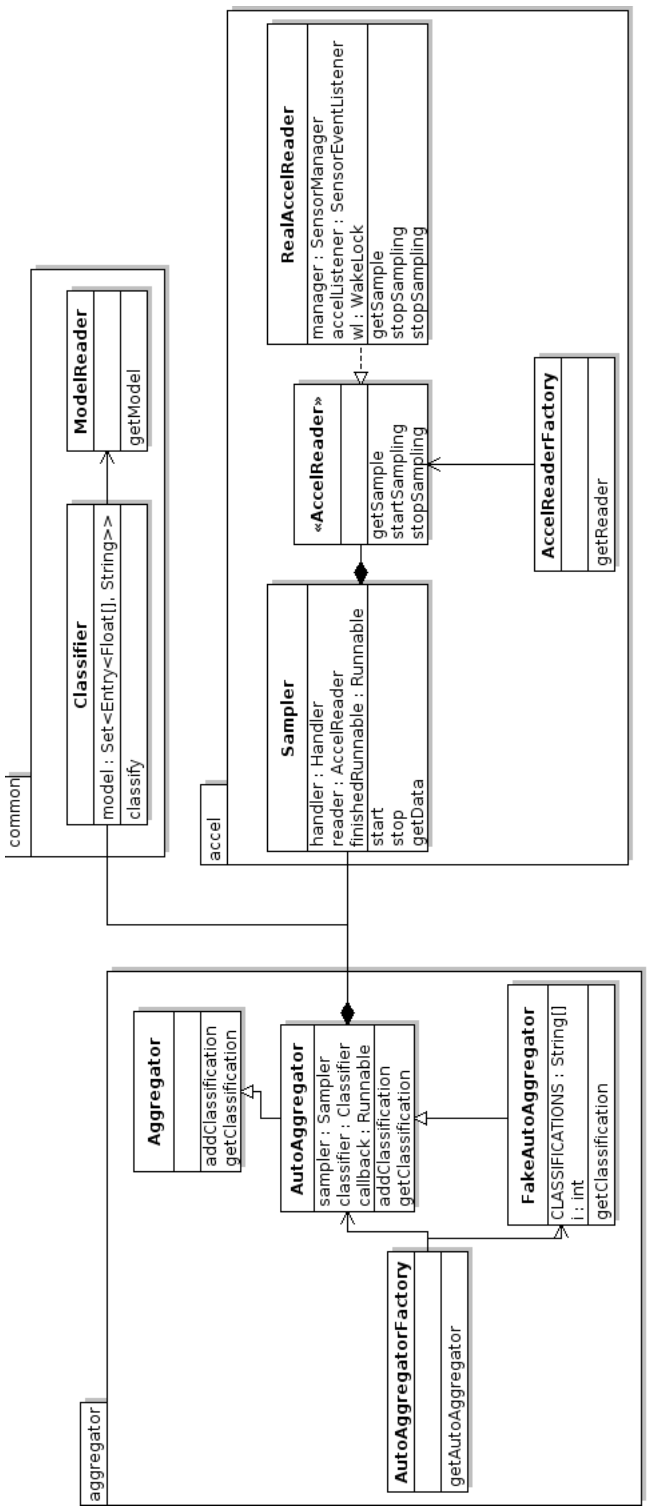


Fig. 5: Accelerometer component class diagram

to obtain the classifications itself. As with AccelReaders, it is constructed by a factory so it can be swapped out for a stubbed implementation with ease. The factory will provide a FakeAutoAggregator if it detects it is running on the Android Emulator; this implementation simply cycles through a list of pre-defined activities to allow mocking of the entire activity inference procedure.

Together, these components are capable of inferring and aggregating the user's current activity from accelerometer data. The factory classes facilitate injection of deterministic classes which can be used to test the system or to eliminate the system from any testing entirely.

The location handling package consists of one interface, a factory, and two concrete implementations. The LocationMonitor interface simply specifies methods to retrieve the current latitude, longitude and accuracy. A concrete RealLocationMonitor implements this by registering for location events with Android's built in LocationManager service, while the FakeLocationMonitor merely cycles through a pre-defined list of latitude and longitude pairs. A factory class determines which implementation to return based on whether or not the code is being executed on an emulator.

21.2 The ContextAnalyserService

The ContextAnalyserService is an Android Service that handles periodic querying of the accelerometer and location components described above, and other assorted tasks related to that data. When it starts, the service creates a new AutoAggregator, a new DataHelper, an instance of Android's built in Geocoder class, and a LocationMonitor. It also retrieves a list of place names which have not yet been geocoded, registers a listener for an application preferences setting titled "run", and schedules its polling method to be called after a sixty second delay.

Every sixty seconds, the service then:

- Retrieves the current latitude and longitude from its LocationMonitor
- Computes the distance between the last stored location and the new position
  - If the distance is less than 500m, there is no known Place associated with the current location, and this location has been observed at least three times:
    - Creates a new place with the latitude and longitude as a name
    - Tells the data helper to add the place to the database
    - Adds the place to the list of place names needing geocoding
  - If the distance is greater than 500m:
    - Update the stored latitude and longitude to the new values
    - If the previous location had an associated Place, tell the data helper to record the visit
  - In both cases:

    * Ask the data helper to find a Place corresponding to the new location, if a place is found:
    * Record the current time as the start time for this visit
    * If a place had been visited previously, tell the data helper to record a new journey with the activity log
    * If the place is different to the last know place, send a broadcast intent

- Attempts to geocode any places with latitude/longitude names
- Starts the auto aggregator

When the auto aggregator finishes, the service retrieves the new aggregated classification. If the user is not at a known place, but has been to one in the past, the activity is added to the activity log and the service checks for predictions. If the activity is different to the last known activity, a broadcast intent is sent.

Predictions work by checking for journeys which begin with similar activities to those recorded in the activity log. The data helper retrieves all known journeys originating at the user's last known location. Each journey is then analysed to determine if it is compatible with the current activity log. This involves comparing the activities in the journey to make sure that they're the same as those logged, and the amount of time spent doing that activity is reasonably close. The current implementation allows for a $\pm 50\%$ variance in the number of repetitions of an activity. The activity log is also allowed to be a subset of the journey being tested, as long as the subset includes the beginning of the journey (that is, the activity log is allowed to be incomplete). If the activity log is found to be compatible with the journey, the journey is added to a list.

Once a list of compatible journeys has been found, they are grouped by destination and the number of times each journey was made is summed into a "score" for each destination. The best destination is then included in a broadcast intent, and all of the predicted destinations are cached for use by the predictions content provider.

The ContextAnalyserService exposes a Bindable interface with methods for retrieving the current activity and any predictions. This is used for intra-process communication: content providers for the activity and predictions can *bind* to the service and request the data they need directly. Other content providers simply access the shared SQLite database themselves, so do not need to communicate directly with the service.

21.3 The public API

The primary purpose of the ContextAnalyser application is to expose the contextual information to third-party developers in an easy-to-user manner. To cater to different types of applications, two distinct methods are provided that expose data: a set of *broadcast intents* which notify interested parties as and when a change occurs, and a collection of *content providers* that allow contextual data to be queried as it is needed.

Broadcast Intents

As mentioned above, the ContextAnalyserService fires broadcast intents on certain occasions, namely:

- whenever the (aggregated) activity changes
- whenever the place associated with the user's location changes
- whenever predictions are calculated

Third-party applications create BroadcastReceivers which can receive these intents. BroadcastReceivers can be registered in code or in the application's manifest file; in the latter case, their creation and maintenance is handled by the OS itself, so there is very little overhead in terms of memory or CPU usage for the application. Each of the broadcast intents include relevant data such as the new activity, new place ID, or most likely predicted destination. This allows applications to implement a lightweight receiver which is able to check conditions and launch a service or activity in response to the user's context.

Content providers

For applications which need more detailed information, or wish to poll the current state instead of receiving it asynchronously, the ContextAnalyser defines a set of Android content providers. An activity or service can query a content provider by simply providing its URI, and optionally a projection, selection, and custom ordering. In return, it receives a Cursor object which can be used to iterate over and retrieve all the results of the query. Content providers are used extensively by the Android OS, for example for accessing contacts, e-mail and SMS messages.

Two content providers - journeys and places - simply pass queries directly on to the backend database, which retrieves data from (or modifies data in) the journey, journeysteps and locations tables. This is extremely simple to implement as the Android SQLiteDatabase class has a query method with the same signature as that implemented by content providers.

A further two content providers - activities and predictions - are backed by data retrieved from the ContextAnalyserService via its bindable interface, as discussed previously. The retrieved data is added to a MatrixCursor and returned to the caller. These two providers, at present, do not allow the caller to specify a projection, selection arguments, or an ordering. These parameters are all biased towards a SQL-like model, and there is no simple way to apply them to a MatrixCursor. Some built-in Android content providers, such as the gmail-ls provider which facilitates access to the user's gmail account, also do not implement projection, selection or ordering; this suggests that in situations where the provider is not backed by a SQLite database, it is acceptable to just ignore these parameters. Of course, they would be more useful if the parameters were respected, so the implementation may be reconsidered in the future.

API class

In order to use receive intents or use content providers, developers need to know the URIs and key names data structures involved. These are published in the developer documentation (Section E), and an optional lightweight "API" is also provided to save developers having to hard code common String and URI values. This API, as discussed in the developer guide, consists of a set of nested public classes with public String and URI constants. Developers simply drop a single JAR archive or a single JAVA source file into their projects, and they can then access constants for the content provider URIs, column names for each exposed content provider, and intent names and metadata for broadcast intents.

Permissions

In order for the Context Analyser function, it has to ask the user for the relevant *permissions* to use the accelerometers, access their location, and keep the device awake. All of these permissions are classed as "dangerous" and are flagged to the user when they install the application. The Android OS enforces access control based on permissions, and will cause runtime exceptions to be thrown if applications attempt to access a resource they do not have access to.

In order to protect users' privacy, the Context Analyser defines its own set of permissions which third-party applications must request in order to retrieve context information. These permissions are all designated "dangerous" so that users will be aware of the information that applications are requesting. Without these permissions, the Context Analyser would open up a loophole whereby applications could retrieve location information from the Context API that they didn't have permission to access directly through the OS. The Android OS makes it extremely simple to enforce these permissions - in most cases a single tag is added in the application's manifest file, or a single method is invoked in code, and the OS handles everything else.

One extra permission was also created which is requested by the Context Analyser itself. This permission is called "BROADCAST" and is used to ensure data integrity for third party applications. Applications are encouraged to check for the "BROADCAST" permission when receiving broadcast intents (and again, the Android OS makes this as simple as adding a tag to an element in the manifest file) in order to ensure that the broadcasts came from a source that the user trusts to make them. This prevents malicious third-party applications broadcasting erroneous data, or broadcasting high volumes of data, to negatively impact context-aware applications.

22 Places application

The "places" application consists of a single activity which shows an Android MapView, with two custom overlay layers. One layer renders a star at the latitude/longitude of each place known to the Context Analyser, while the second layer renders red lines between places that are connected by journeys.

The thickness of the line is proportionate to the number of times the journey has been undertaken.

The application retrieves its data from two of the Context Analyser's content providers. For each location returned by the places content provider, the application creates a new OverlayItem containing the latitude/longitude, name, and a summary of the statistics associated with the place. The corresponding overlay shows a new *toast* containing the name and statistics when the user clicks on or near the overlay item (see Figure 9 on page 56).

The journeys overlay constructs a 2-dimensional map of OverlayItems to OverlayItems to the number of times that journey has been performed (i.e., Map<OverlayItem, Map<OverlayItem, Integer>>). It is passed a list of items created based on the places content provider, and then retrieves all known journeys from the journeys content provider and uses them to build the map. Journeys are normalised so that the ID of the start point is always less than the ID of the end point as the overlay is not concerned with direction.

As each OverlayItem holds the latitude and longitude of the point, it is therefore trivial to iterate over each pair of items, request the MapView translate the latitude/longitude into screen co-ordinates, and render a line between the two points. The line width is set such that the most traveled journey is 10 pixels wide, and all other journeys are scaled in proportion.

23 Locale plugin

The locale plugin is a lightweight application which connects the Context Analyser to the third-party Locale software, which is designed to allow users to change settings of their device automatically when certain "situations" are encountered. By default, situations can include location, time, battery power, orientation and contact information. Many additional plugins are available on the Android Market to augment these conditions. The plugin for the Context Analyser allows users to add conditions for their current activity and (most likely) predicted destination.

In order to show new conditions in Locale, the plugin simply has to define an activity which can handle a certain predefined intent. One such activity is created for each condition, and they simply provide a lightweight user interface to allow the user to select the options for the condition. The activity for selecting a destination, for example, queries the places content provider and displays the result in a combobox. The user then selects the desired place, and either presses their back button, or presses menu and taps "Save" (this is a slightly odd user interface, but it is a standard amongst Locale plugins and highly recommended by the Locale developers).

A broadcast receiver is then created which responds to queries from Locale. It does this by querying the prediction and activity content providers, and comparing the values they return with the values previously supplied for the condition by the user (these are stored in an Intent's *extra* fields, which are provided to the receiver by Locale). It then sets a result code to indicate whether or not the situation is satisfied at present

The broadcast receiver also listens for intents from the content analyser, and sends a special intent to LOCALE to suggest that it should requery the plugin. This allows LOCALE to be aware of when conditions change, and adjust its schedule appropriately. This workflow also allows LOCALE to govern how often the conditions are queried, which allows it to reduce the impact on battery life.

24 Context-aware Home Screen

The context-aware home screen took a substantial amount of development time. It is a replacement for the stock Android 'home' screen which not only gives the user access to their applications and contacts, but displays recent e-mails, text messages, upcoming appointments and missed calls. In addition to this, it learns and adapts to the user's behaviour in a context-sensitive fashion.

The context-aware home screen consists of a set of "modules" which each take up a small portion of the screen real estate. Each module displays one or more actionable items, such as a set of application shortcuts, an e-mail message, or a missed call. The home screen supports a number of "fixed" modules which are displayed at the top of the screen, and a set of other modules which are displayed below and may be scrolled vertically. This allows important or frequently used modules such as application shortcuts to be visible regardless of the other information displayed.

Each module is expected to tailor its behaviour to the user's current context, either by ordering its contents appropriately, or selectively displaying the most relevant content. For example, the application shortcut module will order the shortcuts by those it anticipates will be most likely to be used in the present context, whereas an e-mail module will chose to display the e-mail messages most likely to be read. In addition, modules which aren't fixed are ordered by the home screen according to the context and previous history.

The current context is represented as a set of tuples of context type and value. For example, a complete description of a context may be:

```
(location,1),
(destination,2),
(activity,CLASSIFIED/WALKING),
(day,Monday),
(hour,14),
(period,Afternoon)
```

When the user performs an action, a tuple consisting of the module name and a type and identifier for the action is created. It is then recorded against each tuple in the current context; so if the user tapped an e-mail with the label "work" in the above context, some of the generated tuples would be:

```
(email,label,work,location,1),
(email,label,work,destination,2),
(email,label,work,activity,CLASSIFIED/WALKING),
...
(email,label,work,period,Afternoon)
```

Such an event would also cause similar data to be logged for the sender of the e-mail and the read state. A helper class is provided by the home screen activity to each module to facilitate storing and querying of such tuples. On the database level, each tuple is associated with a *count* - the number of times it has been generated by an event.

When a module is deciding how to arrange or select its contents, it queries for all tuples which match the current context. It can then sum the number of times each action occurred in a related context, and use this score in its selection or ordering algorithm. The same algorithm is applied to the ordering of modules on the home screen itself - each interaction is recorded as a tuple of (HOMESCREEN, ACTIVITY, <MODULE>).

Some actions the user performs may be universal, rather than tied to a specific context. For example, a user is more likely to tap on an *unread* e-mail to read it than to tap an e-mail they've already seen. To compensate for this fact, a special context provider was introduced which constantly provides the tuple (GLOBAL, TRUE). This allows modules to seamlessly incorporate the total number of times an action was performed into their algorithms.

Part VII. Evaluation

This section contains exerts from the project specification and an evaluation of how well the goals were met.

25 Activity classification

> The classifier should be able to classify the following activities: walking, running, standing, sitting, traveling in a vehicle. ... It is expected that the classifier should correctly classify all activities with an accuracy of at least 70%, within 30 seconds of the activity being started.

The activity inference component of the context analyser is able to uniquely identify all of the activities stated in the specification, with the exception of running. In addition, it can distinguish between traveling by car and bus, and walking up or down stairs. Of over 1,000 data samples submitted by users, not a single one was described as "running", which suggests that the omission has not negatively impacted the project in any way.

The accuracy of the classifier when presented with supported activities is over 75% (see 11.1). This was all accomplished by using just two features extracted from two axes of the accelerometer; typical research applications that achieve similar accuracies often use hundreds of features and combine many different sensors.

The timescale for activity detection was consciously expanded during this project. To conserve battery life, polling was reduced to once a minute, and the aggregation of results adds a further delay to activities being classified. Brief tests with the context analyser suggest that activities take 2-3 minutes to be correctly identified. The change in timescale reflects a shift in aims from enabling applications to respond instantaneously to user activity (of which there are few genuine use cases) to facilitating a less obvious background enhancement of applications, as seen in the context home screen application.

26 Experimentation

> The results of the experimentation described should be written up as a report. The reports must include the data collected in each of the experiments, the conclusions drawn from those, and the impact of the results of the experiment on the project deliverables.

Several experiments were performed as part of the project. These include investigating Bluetooth ubiquity (Section 16) and methods of determining whether or not a device is indoors (Section 19). Consideration was also given to the use of data from a camera input (Section 18), and functioning code was developed to use a Microphone input (Section 17).

Ultimately, none of these experiments contributed positively to the end product. The experimentation did, however, save development time which would have been spent on features that would probably have not helped the project. The lack of positive results, while acceptable according to the specification, suggests that other areas should have been considered or alternative approaches attempted. Some possibilities for these are discussed in Section 31.

27 Deliverables

> The following items should be delivered: ... Context-aware Framework, ... Activity condition for Locale, ... Context-aware home screen.

A total of four (final) applications were delivered, along with two applications used during the development process. All three applications required by the specification were successfully implemented, along with the "Places" application which exposes some functionality of the context API which would otherwise be hidden from the user.

> The application should provide two different interfaces to retrieve the user's context. The first should be an implementation of the Android ContentProvider interface. The second interface should use Android broadcast Intents to notify any interested application whenever the user's activity changes.

The API provided by the Context Analyser exposes a total of four content providers, and makes use of three different broadcast intents. These are employed by the Places application, Locale plugin, and context home screen. The two methods compliment each other and facilitate two different methods of interfacing very well.

28 Testing

> Throughout the development of the project, unit tests should be created to test key functionality of all applications. It is expected that at the completion of the project, all unit tests should pass successfully, and they will have a code coverage of 80% or above.

While unit tests were employed throughout the project, the emphasis was on testing small parts of complex or potentially unstable code, and code which was not easily covered by systems tests. Tests were split between those ran on a standard JVM from the development environment, and those that needed to be ran on the device itself with an Android-specific test harness. While no figure for code coverage is available due to this split, and due to lack of coverage tools for the latter type of tests, it is expected that the figure would be well below 80% line coverage.

This represents a failure to adhere to Test-Driven Development (TDD) as originally intended (but not documented). This involves writing unit tests as a form of "specification" before implementing the unit itself. With this project, however, a lot of code started out as small "experiments" to determine whether or not something would work, or the exact output of some built-in functionality; it then evolved into a full implementation, and adding unit tests after the fact was not considered a high priority for most code.

> The classifier application should also have a suite of system tests. These should consist of a set of fake or pre-recorded inputs which are fed into the application in place of raw sensor data. The output of the classifier can then be compared to expected output for the data.

The input sources for the framework are all located behind factory classes. These classes are able to switch out the real information for a stubbed version which simulates the data being received; this is automatically done if the application is ran on the Android Emulator, which does not emulate sensor inputs of any kind. The stubbed classes allow for easy testing of the context framework and the applications that use it, which makes for verification of the whole system extremely easy.

29 User testing and feedback

> The Locale addon and context-aware home screen should be subject to user acceptance testing for evaluation. This should take the form of providing the applications to multiple end users, allowing them to use them for a period of time (providing instructions for certain tasks to complete). The users should then be presented with a questionnaire which they can use to evaluate the functionality, utility and design of the applications. ... In addition to providing the applications to a closed set of users, the applications should be published to the Android market.

All four Context Analyser applications were given to a small group of test users, along with copies of the user manuals and developer's guide. No questionnaires were used, but the users were asked to give feedback and evaluate. Both of the preliminary data gathering applications were published to the Android Market where they received comments and ratings by users. At the time of writing, the Context Analyser suite had not yet been published to the market.

The two test applications published to the market, which were never intended to be particularly valuable to end users, had between 100 and 500 downloads, and both received an average rating between 3 and 4 stars (annotated as "Average" and "Above Average" within the Market UI, respectively). Some positive comments were left, such as the following comment regarding the Sensor Logger application:

> It works on myTouch3G but it would be better if it ran in background and actually kept a log of what you do throughout the day. At least it was right!

Negative feedback mostly revolved around activities which the applications were not trained for being incorrectly classified. As evidenced with the annotated data which was submitted, a lot of users attempted to get the application to classify them sleeping, and were presented with seemingly bizarre results such as “Traveling by bus”. This is because one sample for traveling by bus was close to a “stationary” reading - a vertical acceleration of 9.8m/s, no horizontal acceleration, and little variance.

Methods to correct or detect this type of erroneous result would have given a much better user experience, and reduced the small amount of negative feedback received, but no such methods were attempted during this project. This was because early testing ignored activities which the classifier did not know of, and realisation of the scope of the problem came too late into the project to make the changes necessary. Some ideas for combating this problem are presented in Section 31.

Part VIII. Conclusion

30 Project outcomes

A context-aware API has been developed for the Android platform. This was the primary purpose and output of this project, and it has been evaluated to perform to an acceptable accuracy. While not all of the evaluation targets were met, a number of these were conscious decisions to deviate from the previous specification in response to a change in development method or change in target audience.

In developing the API, an algorithm to classify accelerometer data using an extremely small set of extracted features was devised. This is a significant deviation from established works in the field, which mostly attempt to use as many features as possible in an attempt to increase accuracy.

In addition to the main API, the project has also delivered a set of applications built on top of it, most notably the context aware home screen. This clearly demonstrates how context-aware systems can enhance a user's experience by tailoring content and behaviour to the current context.

31 Future work

There are many areas in this project which could be expanded on or enhanced in some way. Some of these are discussed below.

31.1 Using GSM metadata to enhance activity classification

While experiments with using GSM and GPS signal strength to determine interesting locations were unsuccessful, there is a possibility that some GSM data can be used in order to augment the existing activity classification system. In order to promptly hand off communications to alternative cell sites when the user moves around, all mobile telephones track the cell IDs and signal strengths of their *neighbouring* cell towers as well as the current one.

It is possible that measuring the rate of change in either the strength of neighbouring cell towers, or indeed the change in cell towers themselves, could be used to infer the approximate speed the user is traveling at. While this could be a piece of standalone contextual information, it would be possible to integrate this with the activity inference system to help reduce incorrect results.

As discussed earlier, a large percentage of incorrectly classified samples involved the user sitting down and being misclassified as being in a vehicle. A simple decision tree could be implemented that biased the activity inference algorithm to favour the "sitting" classification over vehicle-based ones if the user was not moving at speed. This has the potential to increase accuracy to the region of 90%.

31.2 Automatic classification of places

While the context analyser correctly identifies interesting places, it would be useful to know what type of place they were. “Home” and “Work” could potentially be inferred by the times of day spent at each location; interchanges could be identified by the duration of visits and the activity involved in getting to and from the location, etc.

This would enhance the behaviour of contextual location-aware services, which currently have to rely on analysing the user’s activity (either their real life activity or their interactions with the phone) in order to correctly adapt their behaviour to places. Instead of observing that a user frequently reads e-mails at place #17, an application could infer that as place #17 is a work place, the user is more likely to want to read e-mail than play games.

31.3 On-device training for new activities

Because it uses an instance-based learning algorithm, it would be very easy to add new samples to the activity classification model on the device itself. However, there are various challenges which would make this more difficult. At present the model data is stored in a binary file bundled with the Context Analyser; this means that it is not modifiable at runtime. In order to integrate new data, it would either have to be extracted to storage on the device and then modified, or imported into a database.

In addition, it would be hard to ensure that data entered on the device was of a good quality. Without a way to explore the model itself, the user has no way of finding and correcting or deleting poor quality training samples which lead to false classifications. Finally, removing the developer from the update process means that new activities have to be added by each and every user, instead of added once by the developer and shared to all users. Further work on this topic would have to explore ways to allow users to add high quality data, and then ensure that data is shared appropriately.

31.4 “Omniscient” context home screen

The present context home screen only records what happens when the user directly interacts with it. Further work could expand this to make the home screen aware of all actions that were performed on the device, and to incorporate these into the dashboard to further improve the user experience.

An obvious example of this is monitoring outgoing phone calls, which can be invoked from a multitude of applications without directly tapping the contact on the home screen. At present, the home screen sees a tap on the “Dialer” application (for example) and nothing else; if it monitored phone calls and other activities it could also make the contact involved more prominent on the home screen.

Another use would be monitoring the time spent in applications (rather than just the number of launches), and which activities within those are accessed. If a user repeatedly opens the market application to browse to the most recently added games, the context home screen may be able to detect that behaviour and automatically add a *new* shortcut which performs this action automatically for them. Work would need to be done in order to determine whether or not this was possible in a general case for any application.

Part IX. Appendices

A User Guide - Context Analyser

Welcome to the CONTEXT ANALYSER. This application analyses your *context* as you carry your phone around with you performing day-to-day activities. Other applications can query your context information and enhance their behaviour so that they're more relevant to what you're doing.

The context analyser currently provides the following context information:

- Your activity
- Your location
- Your predicted destination, if you're not at a known location

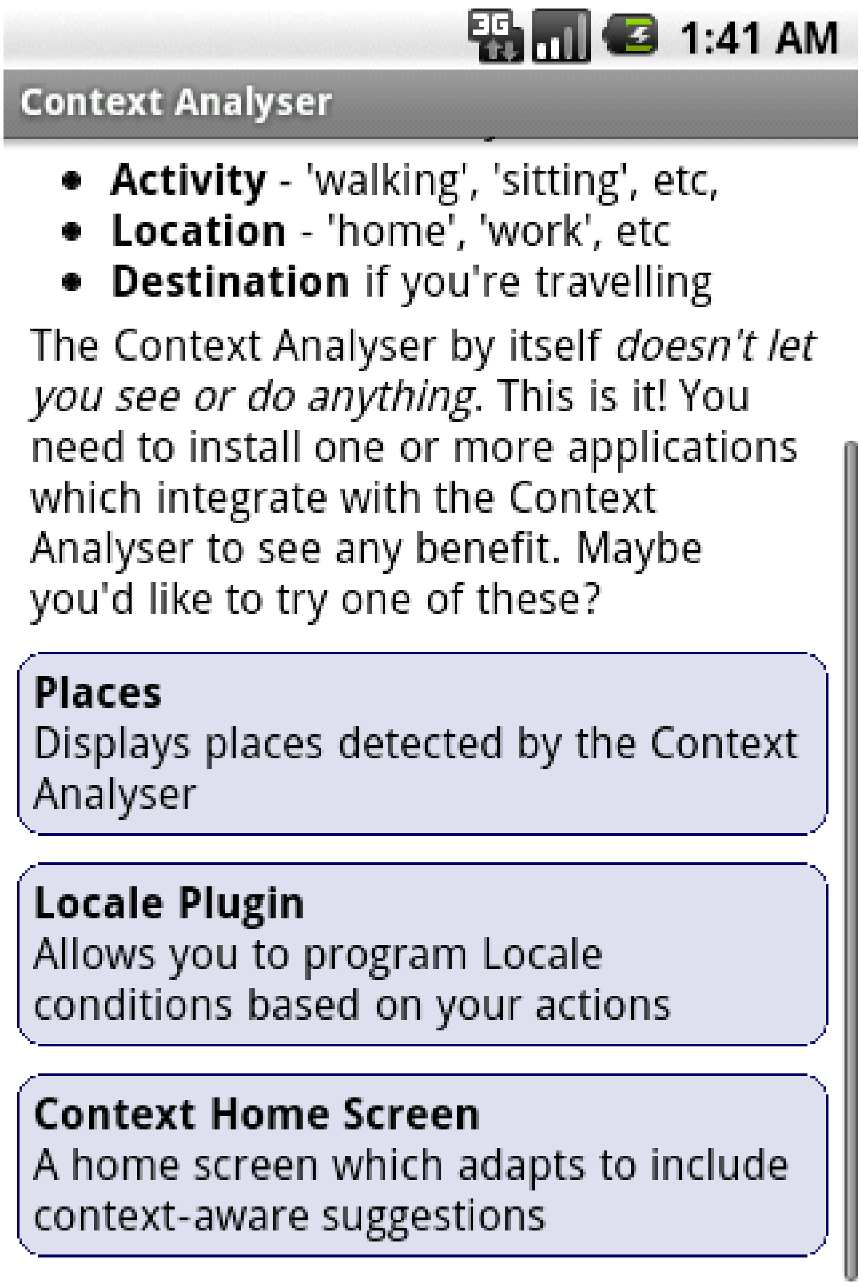
It does this by gathering data from your phone's accelerometers and location provider. Data is recorded for a few seconds once a minute, to preserve your battery life.

For your 'activity', the context analyser tries to detect whether you are walking, sitting, standing, or in a vehicle. It can also expand on these to detect the difference between a car and a bus, and walking normally and walking up or down stairs. Future updates may add more activities.

For your location and destination, the context analyser deals in 'places'. These are locations up to 500m wide where you have spent several minutes not moving. A place could correspond to your home, office, favourite coffee shop or a bus stop you frequently wait at. When you move between places, the context analyser notes which activities it thought you were performing, and stores these as a 'journey' which links the two places. When you next set off from one of these, it can compare your activities and see if it can guess where you're heading.

There is very little you can do with the context analyser on its own. It's built as a tool for other applications to hook into. You can see some of these featured applications listed when you open the context analyser (see Figure 6 on page 52) . Simply click on one of them and the Android market will open for you to view or install your chosen app.

The context analyser will run a background service as soon as you open the application, or whenever another application requests data from it. If you wish to disable the background service (which will stop the context analyser from detecting places or determining your activity), press the MENU button and select the DISABLE SERVICE option (see Figure 7 on page 53). The background service will not run until you repeat the procedure and select ENABLE SERVICE.


1:41 AM

Context Analyser

- **Activity** - 'walking', 'sitting', etc,
- **Location** - 'home', 'work', etc
- **Destination** if you're travelling

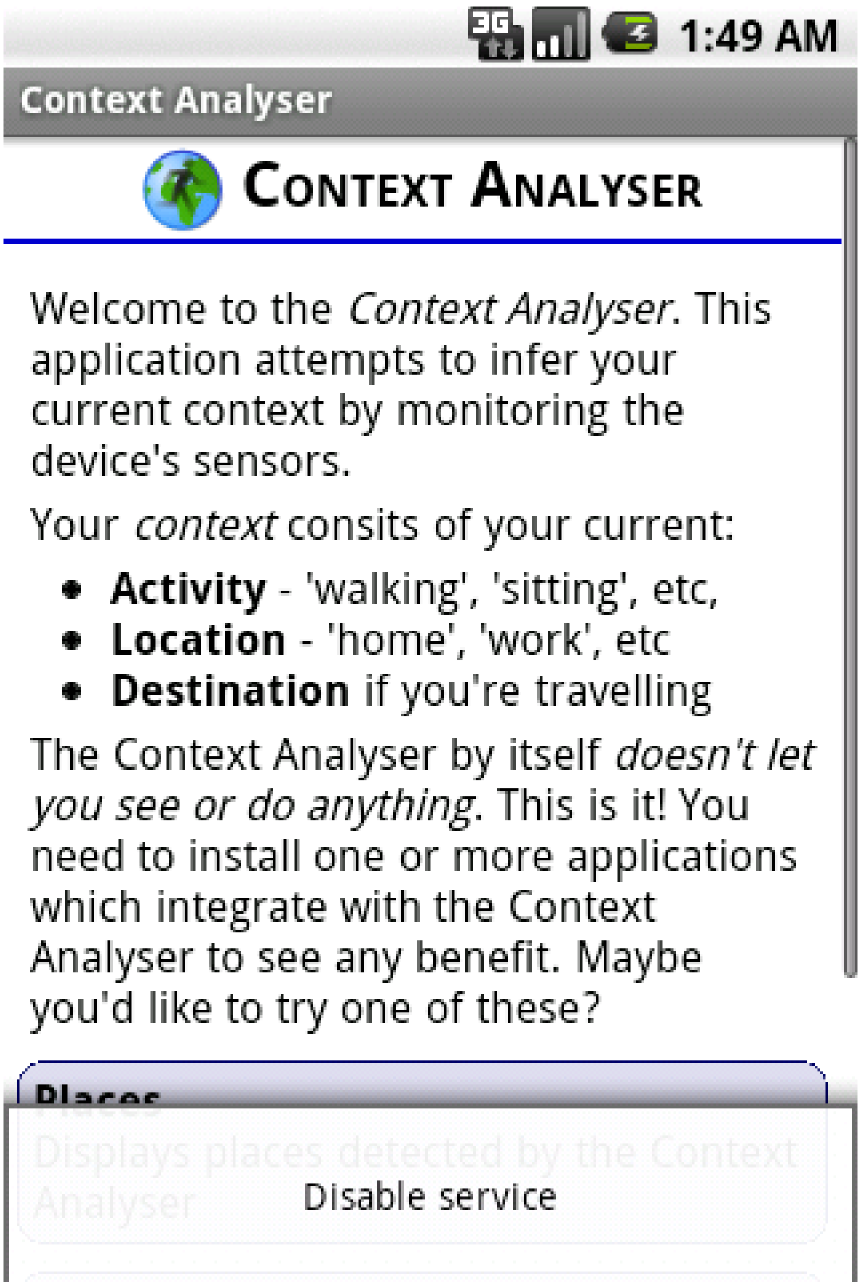
The Context Analyser by itself *doesn't let you see or do anything*. This is it! You need to install one or more applications which integrate with the Context Analyser to see any benefit. Maybe you'd like to try one of these?

Places
Displays places detected by the Context Analyser

Locale Plugin
Allows you to program Locale conditions based on your actions

Context Home Screen
A home screen which adapts to include context-aware suggestions

Fig. 6: Featured applications


1:49 AM

Context Analyser

CONTEXT ANALYSER

Welcome to the *Context Analyser*. This application attempts to infer your current context by monitoring the device's sensors.

Your *context* consits of your current:

- **Activity** - 'walking', 'sitting', etc,
- **Location** - 'home', 'work', etc
- **Destination** if you're travelling

The Context Analyser by itself *doesn't let you see or do anything*. This is it! You need to install one or more applications which integrate with the Context Analyser to see any benefit. Maybe you'd like to try one of these?

Places

Displays places detected by the Context Analyser

Disable service

Fig. 7: Disable service button

B User Guide - Places

The PLACES application shows you all of the places detected by the CONTEXT ANALYSER, and the journeys you make in between them. The CONTEXT ANALYSER is a tool which tries to determine your current activity and location, and your destination if you're on the move. If you don't have the CONTEXT ANALYSER installed, you'll be prompted to install it from the market.

The places application displays a map and marks each detected place with an orange star. Places you've traveled between are connected with a red line - the thicker the line, the more frequently you make that journey. In Figure 8 on page 55, you can see two places with a very frequently made journey between them.

You can pan the map by dragging it around with your finger, or by moving your phone's trackball (if it has one). To zoom in and out, tap the screen and wait two seconds and the zoom controls will appear at the bottom of the map. Simply tap either button to zoom in or out.

To see more information about a place, simply tap on its star. A small *toast* will appear giving you the name of the place, the number of visits, and the date and time of your last visit. Names are determined automatically by geocoding the latitude and longitude to a nearby place name - this could be the name of a street, a nearby landmark or a side alley. Figure 9 on page 56 shows the result of tapping one of the stars - the name of the place is "Rose Alley", it has been visited 37 times and the last time this happened was in the early hours of July the 10th.

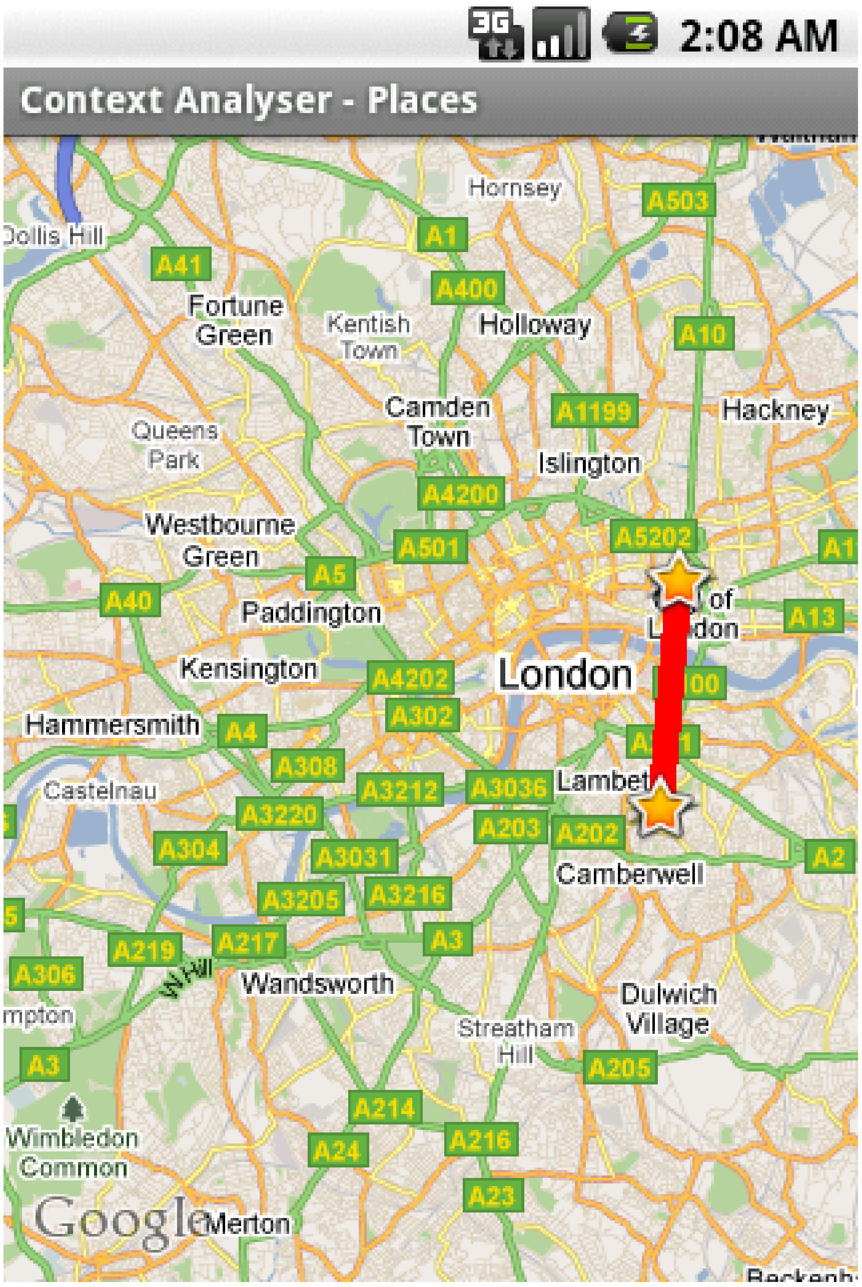
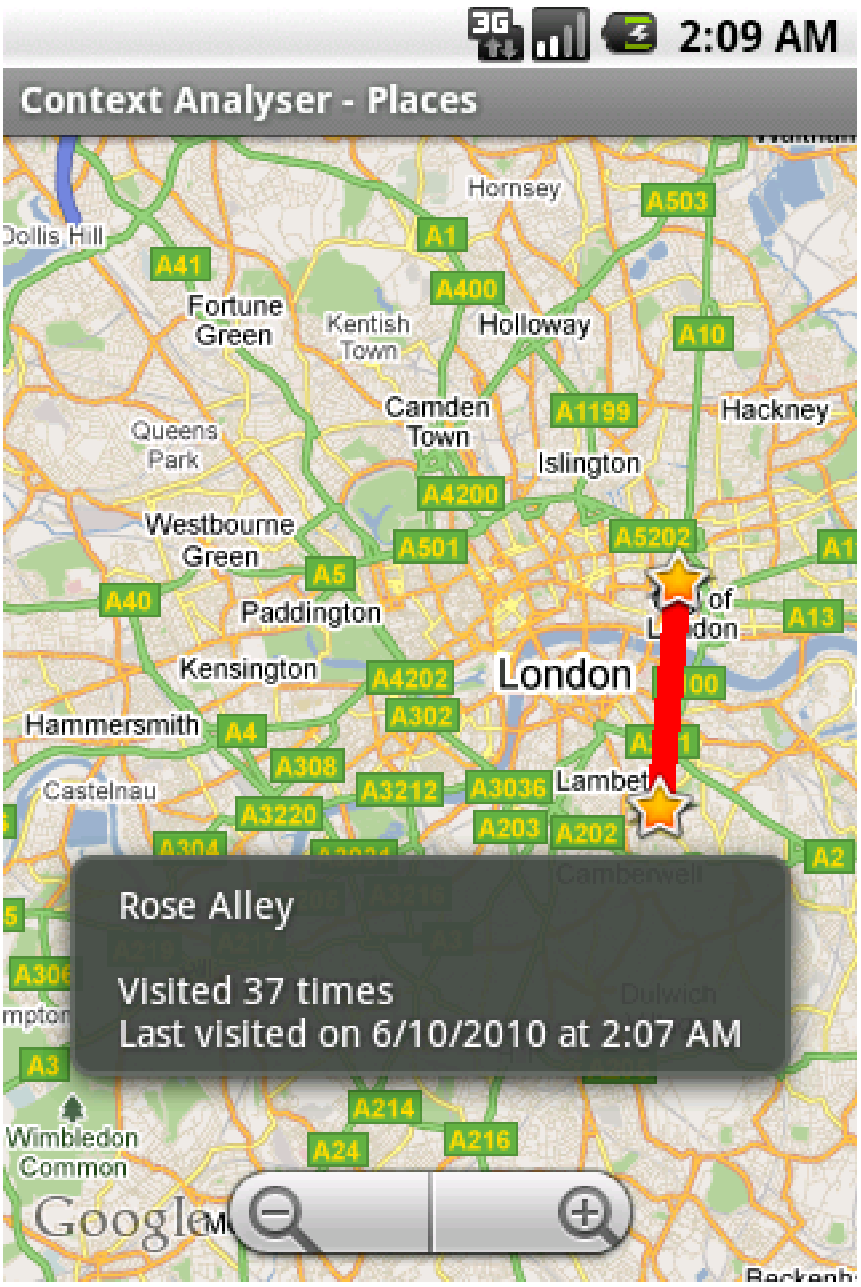


Fig. 8: Places application


2:09 AM

Context Analyser - Places

Rose Alley

Visited 37 times
Last visited on 6/10/2010 at 2:07 AM

Fig. 9: Place details

C User Guide - Locale plugin

The Context Analyser plugin for Locale allows you to create Locale situations based on information provided by the Context Analyser. Locale is an application which allows you to make your phone change settings, send tweets, or perform a variety of other actions, whenever a certain 'situation' occurs. A situation is just a set of conditions - such as your location, your phone's battery life, or the current time. The Context Analyser application automatically determines your current activity (such as 'walking' or 'sitting down'), and your predicted destination if you're traveling. This plugin requires that you have both Context Analyser and Locale installed. Both are available from the Android market.

The plugin adds two new conditions which you can use in Locale situations:

- Activity - the activity you are *most likely* performing
- Destination - your *most likely* destination, if you're not at a known location

A basic overview of adding these new conditions follows. For full documentation on how to use Locale, consult the Locale user manual.

When you open Locale, you will see a list of currently defined situations (Figure 10 on page 58). You can edit existing situations by tapping on them. Existing situations may be deleted by clicking on the red minus button to the right of the situation name. Finally, new situations can be created by tapping the 'Add Situation' button at the bottom of the screen. Locale shows any situations that are *active* (all the conditions currently hold true) in bold.

Once you've selected a situation to edit, or created a brand new situation, you will see Locale's 'edit situation' screen (Figure 11 on page 59). This allows you to set the name of the situation, add or edit conditions, and add or edit settings. The Context Analyser plugin adds two new conditions, which are shown in the list when you click 'Add condition' (Figure 12 on page 60). Select either 'Activity' or 'Destination' to add a new condition based on your current context.

When you select one of the plugin's conditions, you will be presented with a screen which allows you to specify which activity (see Figure 14 on page 62) or destination (see Figure 13 on page 61) you wish to match. For activities, you can select one of sitting, standing, walking, walking up stairs, walking down stairs, dancing, traveling bar car, or traveling by bus. For destination, you may select any place which the Context Analyser has previously detected.

Once you've selected your chosen activity or destination, press the menu button and select the 'Save Changes' option (see Figure 15 on page 63). You will then return to the situation editor where the new condition will be displayed. You may then add settings as with any other Locale situation, and Locale will automatically apply these when all of the conditions match.

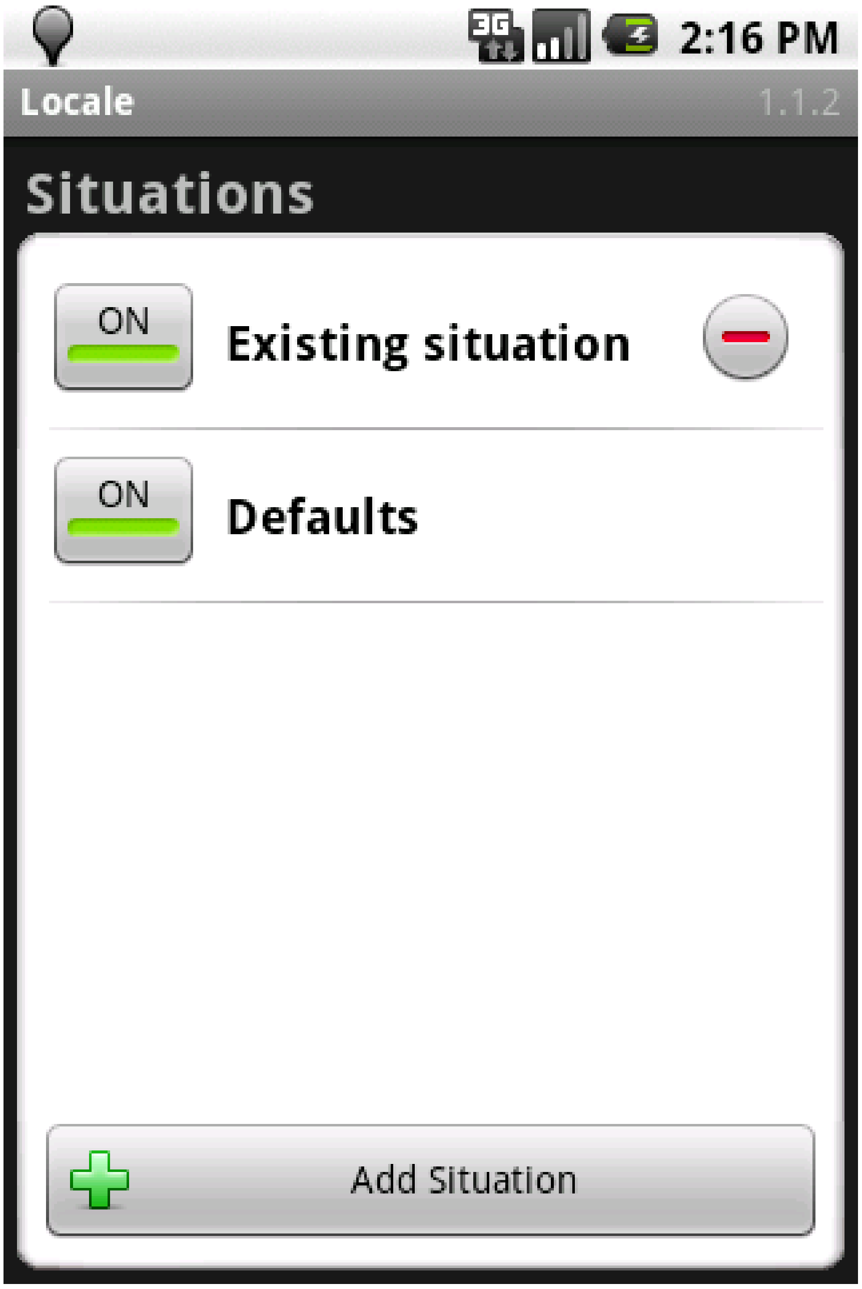


Fig. 10: Locale main screen

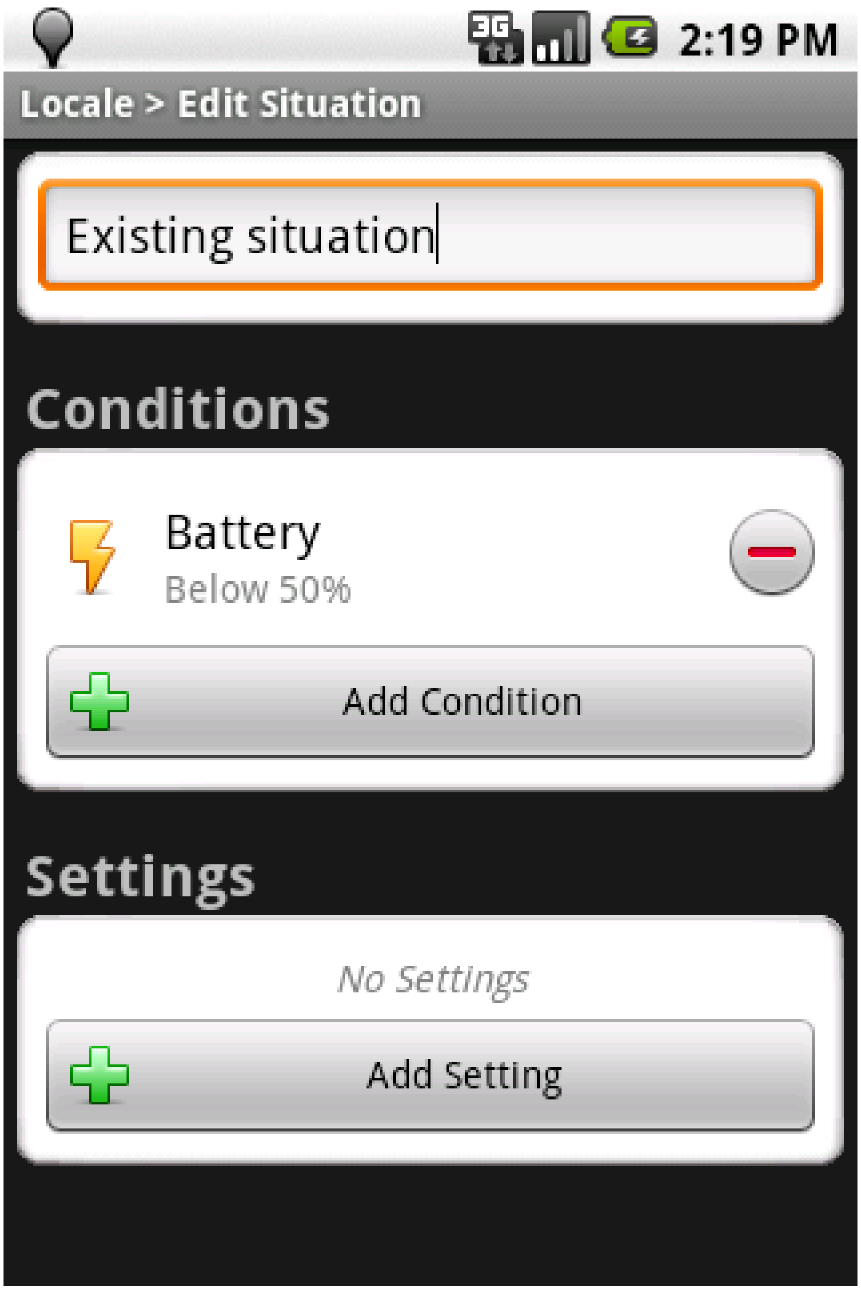


Fig. 11: Locale editing situation

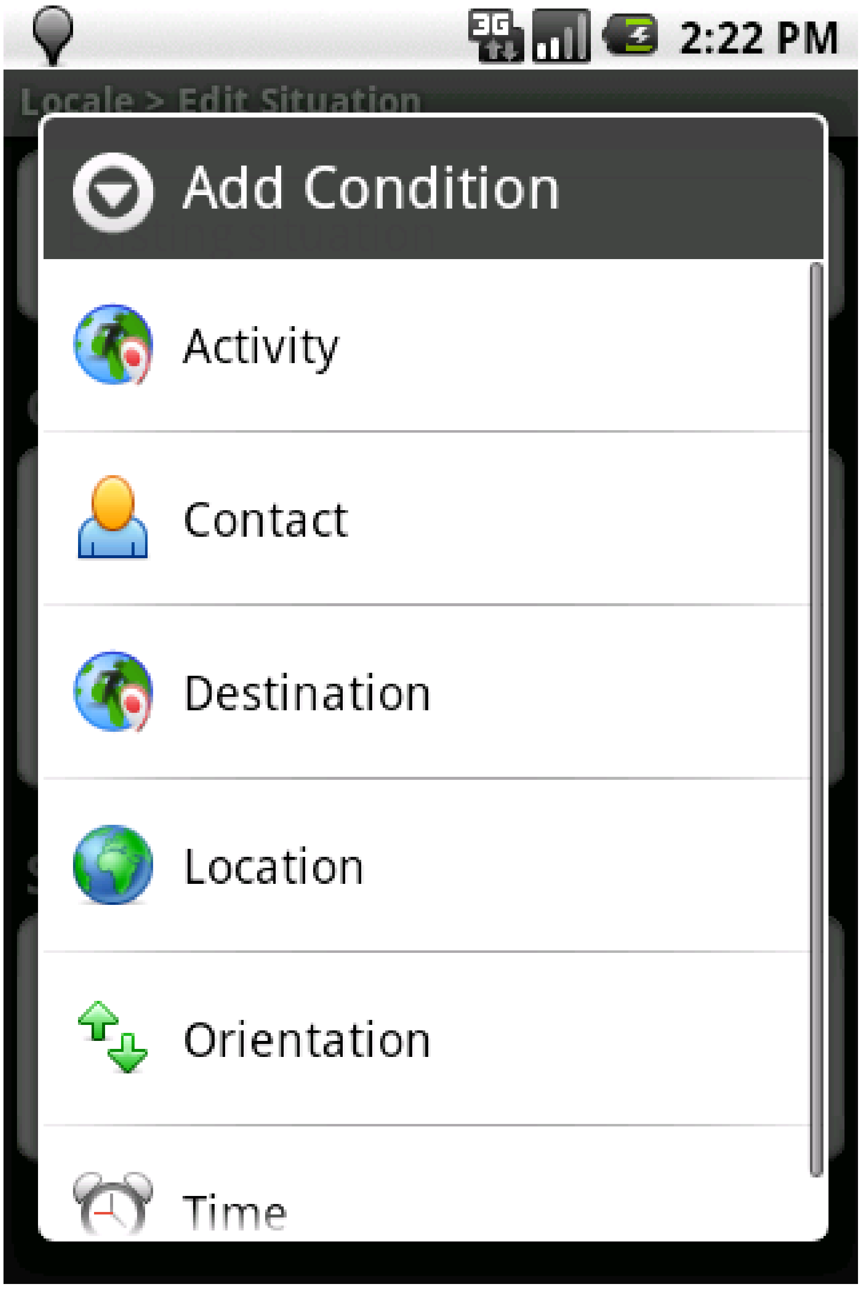


Fig. 12: Locale add condition popup

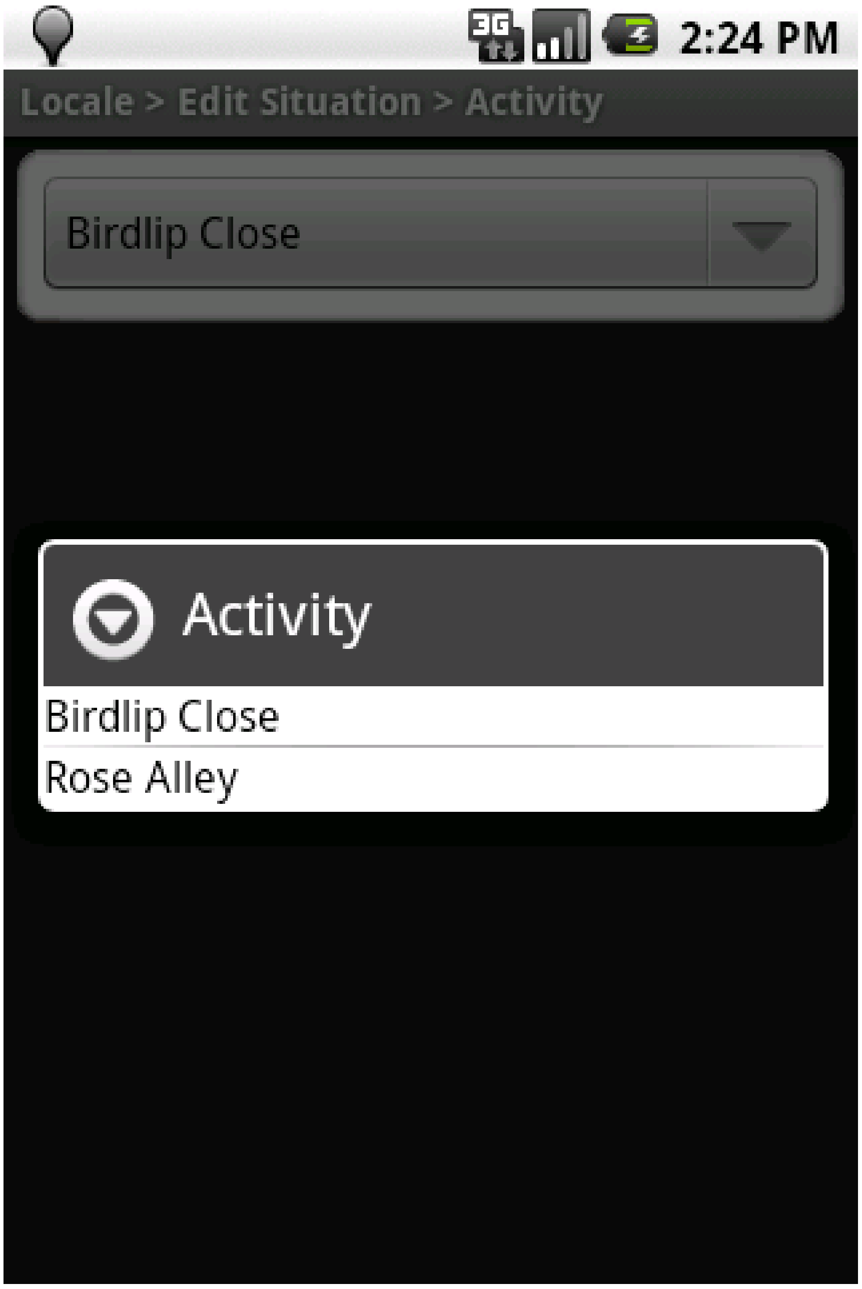


Fig. 13: Locale plugin selecting destination

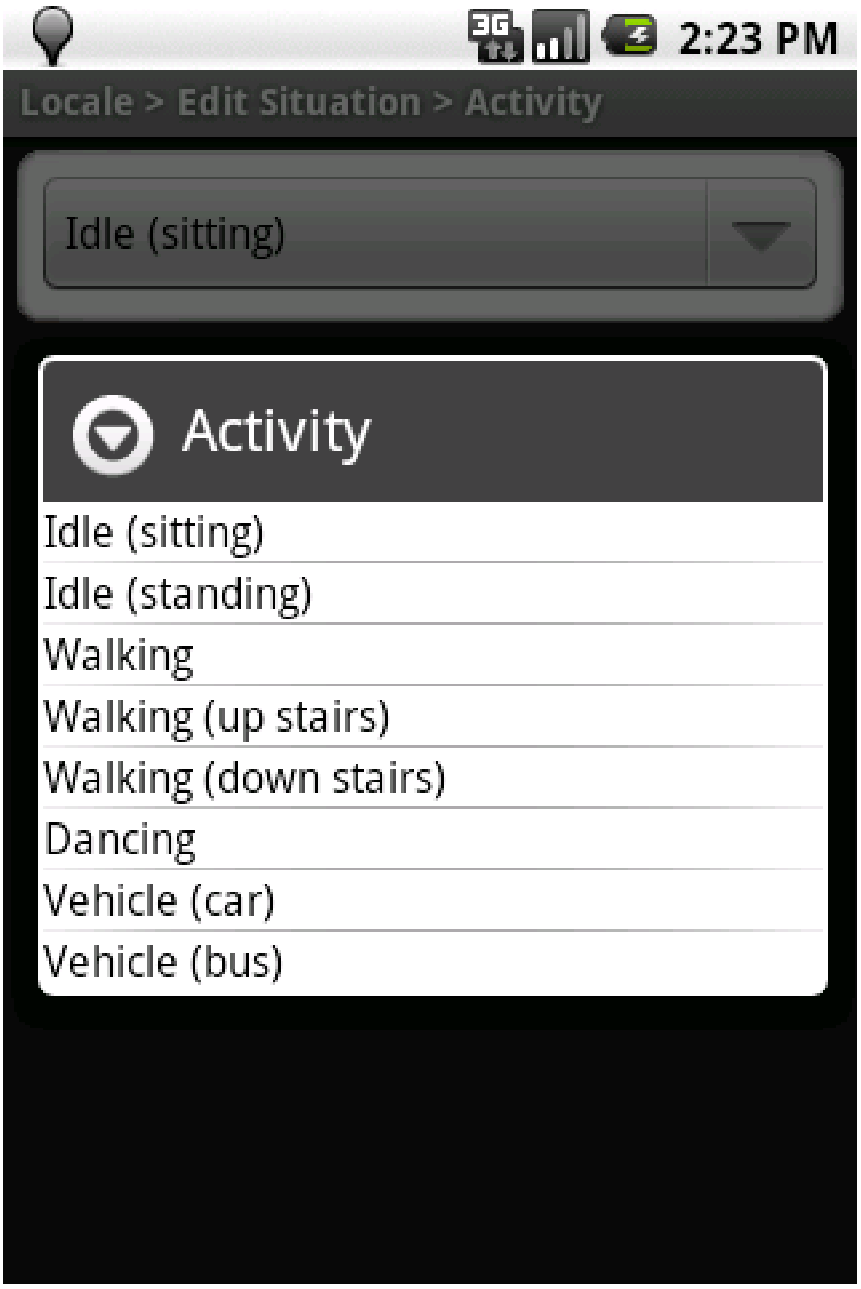


Fig. 14: Locale plugin selecting activity

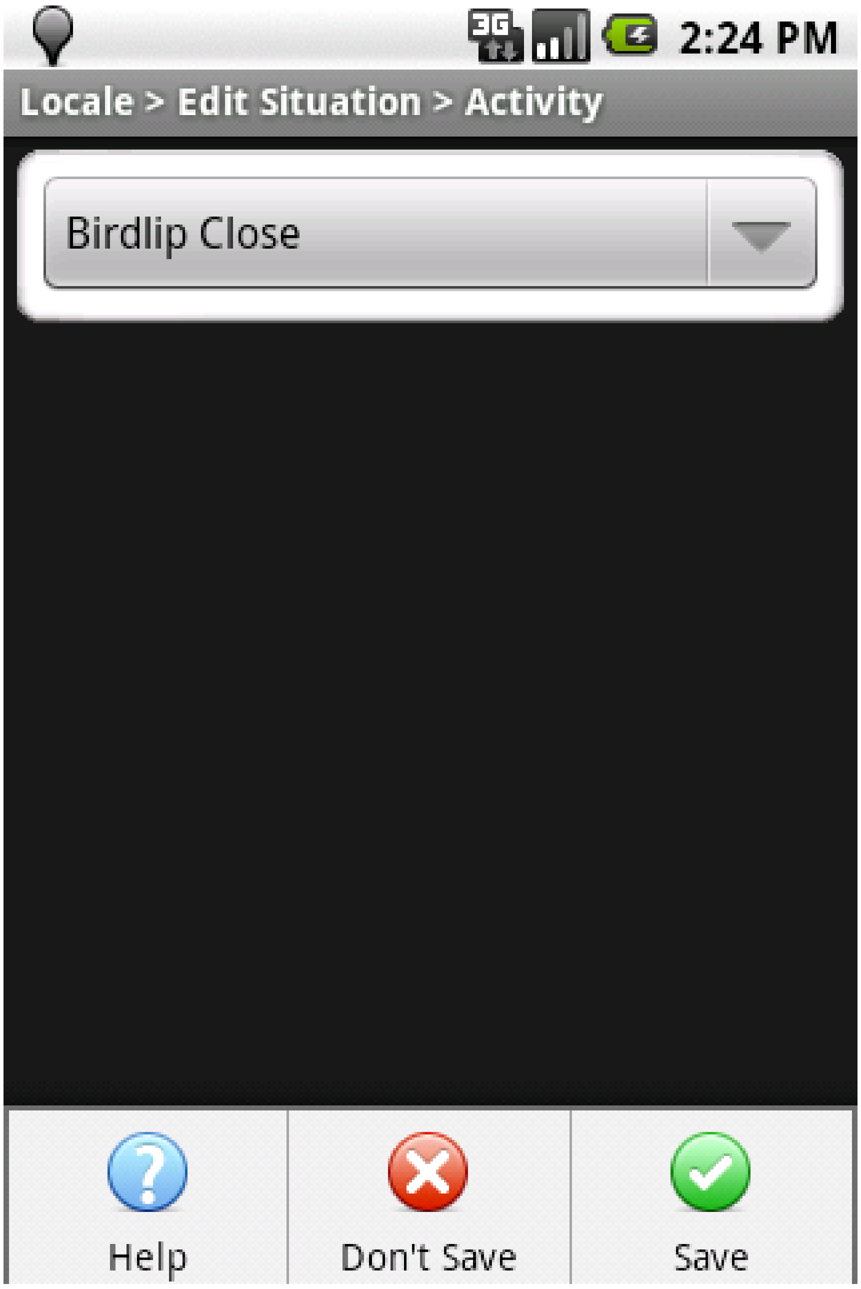


Fig. 15: Locale condition editor menu

D User Guide - Context Home Screen

The "context-aware home screen" replaces your phone's normal home screen with an information rich dashboard which can adapt to offer different information based on your current context. As you use the home screen (by, for example, launching applications, opening e-mails, etc), the home screen remembers what you were doing, when you did it, and where you were at the time. It then applies this information to automatically make the shortcuts or messages it thinks you will be most interested in more prominent.

Once the context-aware home screen is installed, pressing the "Home" key or turning on your device will prompt you as to which application you wish to provide the "Home" functionality, as seen in Figure 16 on page 65. Select "Context Home" to open the context home screen. If you want this to *always* open when you press home, tick the "Use by default for this action" checkbox at the bottom of the prompt before making your choice.

The context home screen is shown in Figure 17 on page 66. The top two lines of the home screen show shortcuts for your installed applications and contacts who have pictures associated with them. You can open the application or contact information screen by tapping on the appropriate image. To scroll left or right, simply drag your finger across the shortcuts. You may also use your device's trackball to navigate around the screen, if it has one.

The rest of the home screen shows you your recent e-mails, text messages, appointments, missed calls, and other information. You can scroll up and down in this area by dragging your finger up or down. Tapping on an item will open it in the relevant application. The context home screen will attempt to show contact pictures for each event where they are available; if the sender is not associated with a contact, or the contact has no photograph, then a white "running man" will be displayed.

As you use the context home screen, you should notice that the applications you use most frequently start appearing nearer to the start of the list and the types of event that you click on trickle up towards the top of the screen. If you regularly call someone every Saturday evening, you should find that person at the top of your contacts list when you look on Saturday evening. The more you use the context home screen, the better it adapts to your needs!

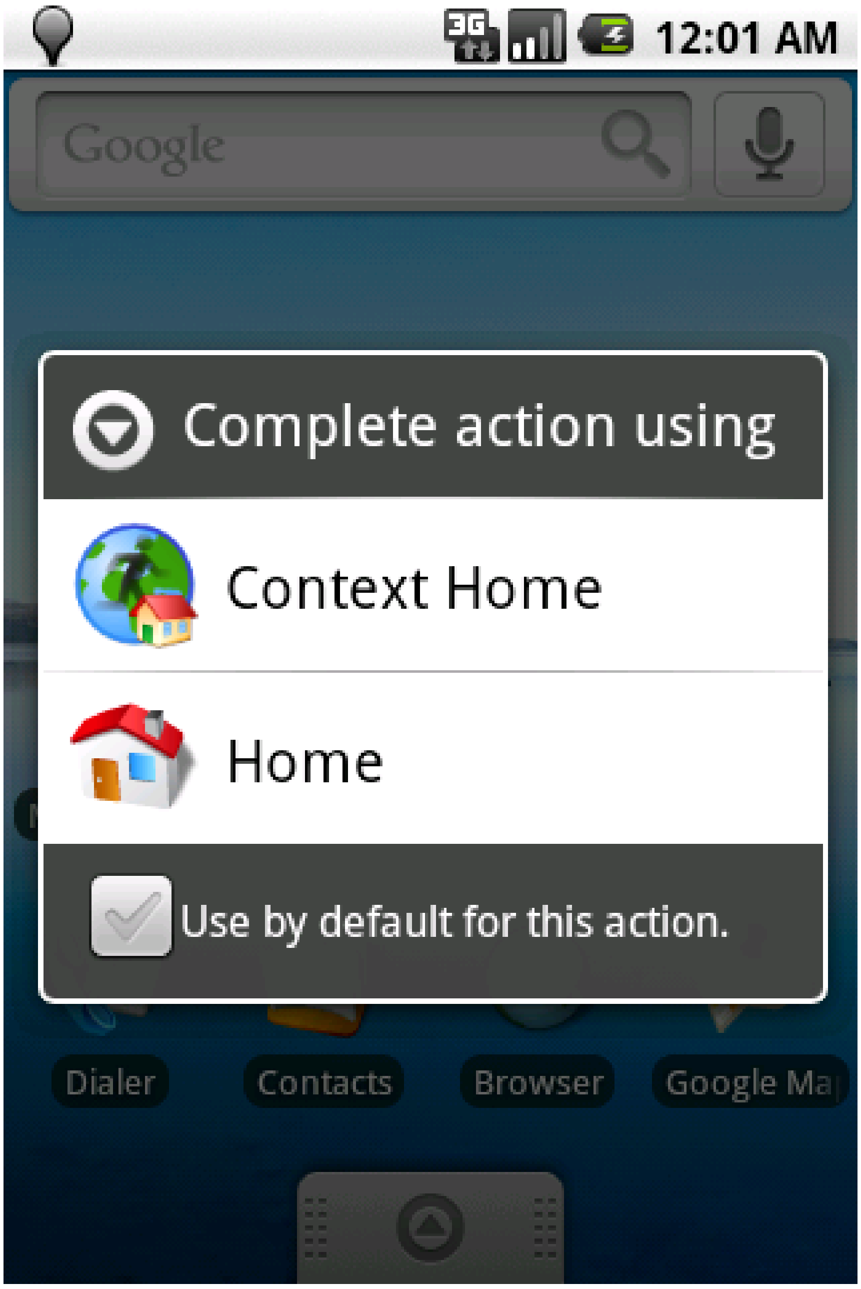
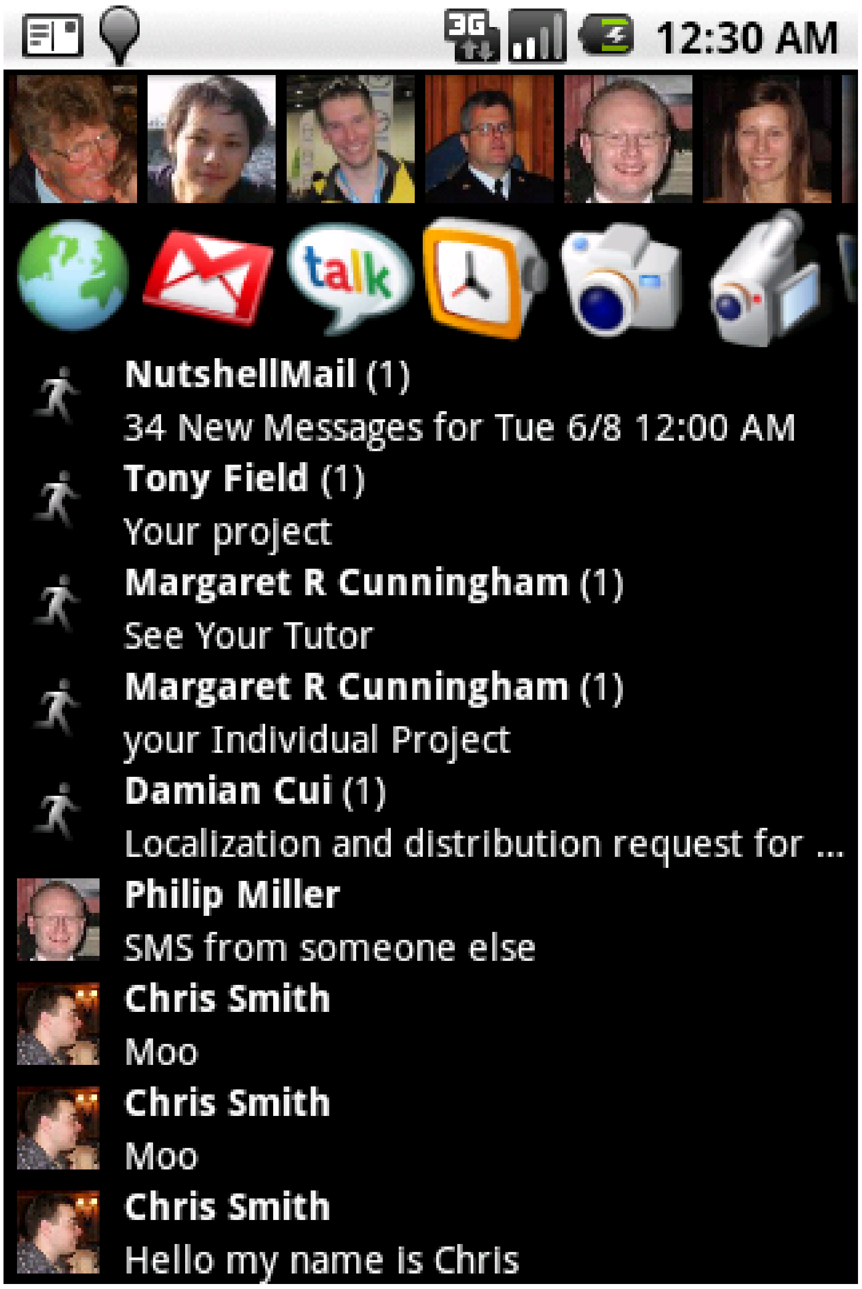


Fig. 16: Home screen selection



NutshellMail (1)
34 New Messages for Tue 6/8 12:00 AM
Tony Field (1)
Your project
Margaret R Cunningham (1)
See Your Tutor
Margaret R Cunningham (1)
your Individual Project
Damian Cui (1)
Localization and distribution request for ...
Philip Miller
SMS from someone else
Chris Smith
Moo
Chris Smith
Moo
Chris Smith
Hello my name is Chris

Fig. 17: Context home screen

E Developer Guide

This guide details how you can integrate the data made available by the Context Analyser into your own applications. It is expected that you are familiar with Android application development and general concepts. For an overview of how the Context Analyser works, please consult the user guide.

The Context Analyser exposes its data via a set of Content Providers. These are augmented by several broadcast intents which can be used by third party applications to receive immediate notification when the user's context changes.

E.1 Permissions

In order to protect the user's privacy, a series of permissions are defined by the Context Analyser, and access to data and broadcast intents is limited to applications which hold the appropriate permissions. The Context Analyser defines the following permissions:

- uk.co.md87.android.contextanalyser.RECEIVE_UPDATES - allows the application to receive real-time updates about context or place information through broadcast intents
- uk.co.md87.android.contextanalyser.READ_PLACES - allows the application to read places from the relevant content provider
- uk.co.md87.android.contextanalyser.WRITE_PLACES - allows the application to modify places via the relevant content provider
- uk.co.md87.android.contextanalyser.READ_JOURNEYS - allows the application to read journey information from the relevant content providers
- uk.co.md87.android.contextanalyser.WRITE_JOURNEYS - allows the application to modify journey information via the relevant content providers
- uk.co.md87.android.contextanalyser.BROADCAST - permission used when broadcasting context-related intents. Applications may wish to check that broadcasters hold this permission to prevent third parties broadcasting erroneous data.

If you do not declare the required permissions, a run time error will occur when your application attempts to access protected data. All permissions listed here are defined as dangerous, which means that end users will be prompted about them when installing any application using them; this is in line with the underlying accelerometer and location permissions.

E.2 Model

The Context Analyser is backed by a database containing tables for *places*, *journeys* and *journey steps*. Activities and predictions are stored in memory.

A **place** is a location represented by a latitude/longitude tuple. Places have a radius of 500 metres; whenever a user moves to within 500 metres of a known place, they are assumed to be located at that place. New places are identified when the user is observed to be in the same location (within 500 metres) for three consecutive readings, which occur at 1 minute intervals. Places also have an associated name, which is initially set to a string representation of the latitude and longitude; the Context Analyser will attempt to rename any place with such a name by geocoding the latitude/longitude into a nearby street or landmark name.

An **activity** is represented by a string delimited by forward slashes (/). This forms a hierarchy of classifications, with the following possible values:

- CLASSIFIED/DANCING
- CLASSIFIED/IDLE/SITTING
- CLASSIFIED/IDLE/STANDING
- CLASSIFIED/VEHICLE/BUS
- CLASSIFIED/VEHICLE/CAR
- CLASSIFIED/WALKING
- CLASSIFIED/WALKING/STAIRS/DOWN
- CLASSIFIED/WALKING/STAIRS/UP

It is possible for activities to contain only a part of the hierarchy - for example if the readings alternate between classifications for traveling by bus and by car, they will be aggregated and the resulting activity will be simply CLASSIFIED/VEHICLE. Similarly, if the user's activity is in constant flux, the resulting activity will be aggregated to simply 'CLASSIFIED'.

A **journey** is a recorded sequence of activities which occurred when the user moved from one place to another. Whenever the user leaves a known place, the Context Analyser begins a log of activities; when they then reach another known place, the log is converted into a journey. Each journey consists of a sequence of **journey steps**, which describe a single activity and a number of repetitions.

For example, if the user leaves place A, spends five minutes walking, then ten minutes on a bus, and arrives at place B:

- A journey will be created with a start point of A and an end point of B
- The journey will consist of two journey steps:
  - Step one will have an activity of CLASSIFIED/WALKING with five repetitions
  - Step two will have an activity of CLASSIFIED/VEHICLE/BUS with ten repetitions

When the user is on a journey, the Context Analyser compares their current history to that of previously recorded journeys leaving the same place. If any of the journeys match, their destinations are considered as **predictions**. The number of times each matching journey occurred is summed and the result is used as a 'score' for the destination involved. The destination with the highest score is considered the most likely destination for the user.

For the purposes of prediction, a partial journey is considered to match a historical journey if:

- it contains the same number of, or fewer, steps
- each step has the same activity
- the number of repetitions in each completed step (not the last step) is:
  - no less than 50% of the historical value
  - no greater than 150% of the historical value
- the number of repetitions of the last step is:
  - no greater than 150% of the historical value

E.3 Broadcast intents

The Context Analyser broadcasts three different intents to make other applications aware of certain events. Applications require the RECEIVE_UPDATES permission to receive these broadcasts, as discussed previously. These intents are:

- uk.co.md87.android.contextanalyser.ACTIVITY_CHANGED
- uk.co.md87.android.contextanalyser.CONTEXT_CHANGED
- uk.co.md87.android.contextanalyser.PREDICTION_AVAILABLE

The **ACTIVITY_CHANGED** intent is broadcast whenever the user's activity is discovered to have changed. It contains two string extras - the old activity and the new activity - under the keys "old" and "new" respectively.

The **CONTEXT_CHANGED** intent occurs whenever some other aspect of the user's context changes. Presently this only includes the user's current place. It contains an integer extra under the key "type" which describes which type of context has changed; this will have the value 1 for place updates. Place updates will also have two further integer extras - the place the user was previously in (or -1 if they were not in a known place) and the place the user is now in - under the keys "old" and "new", respectively. These place IDs can be resolved to names and latitude/longitude by querying the relevant content provider, described below.

The **PREDICTION_AVAILABLE** intent is broadcast whenever a prediction has been made by the Context Analyser. The best available prediction is included as a place ID in the "best_target" key, the total score for the prediction to that place is available as an integer in the “count” key, and the ’probability’ (the score for that place divided by the total scores for all predicted places) is a float in the “best_probability” key. A full set of predictions can be retrieved from the relevant content provider, described below.

As discussed in the permissions section (E.1), you may wish to make sure that all broadcasts you receive are sent by an application which holds the BROADCAST permission. This ensures that the user trusts the application to send these broadcasts, and reduces the chance that they are from a malicious application attempting to introduce erroneous data or otherwise compromise your receiver.

E.4 Content providers

The Context Analyser provides four content providers from which you can receive data:

E.4.1 Activities

Allows querying of the user’s current activity. This may be expanded in future versions to allow querying of all known activities. Read only, does not accept selection, projection or order parameters.

URI: content://uk.co.md87.android.contextanalyser.activitiescontentprovider/current

Content type: vnd.contextanalyser.activity

Columns:

Name	Type	Remarks
activity	string	The user’s current activity

E.4.2 Journeys

Allows querying of the user’s historical journeys and steps within them. Read/write access, supports selection and ordering.

Journeys URI: content://uk.co.md87.android.contextanalyser.journeyscontentprovider/jou

Content type: vnd.contextanalyser.journey

Columns:

Name	Type	Remarks
_id	long	A unique, persistent ID for the journey
start	long	The ID of the place at which this journey starts
end	long	The ID of the place at which this journey ends
steps	int	The number of steps in this journey
number	int	The number of times this journey has been made

Journey steps URI: CONTENT://UK.CO.MD87.ANDROID.CONTEXTANALYSER.JOURNEYSCONTENTPROVIDER

Content type: VND.CONTEXTANALYSER.JOURNEYSTEP

Columns:

Name	Type	Remarks
_id	long	A unique, persistent ID for the step
activity	string	The activity that was observed
repetitions	int	The number of times the activity occurred
journey	long	The ID of the journey that this step belongs to
next	long	The ID of the next step in the journey, or 0 if the last step in sequence

E.4.3 Places

Allows querying of the user's know places. Read/write access, supports selection and ordering.

URI: CONTENT://UK.CO.MD87.ANDROID.CONTEXTANALYSER.PLACESCONTENTPROVIDER/PLACES

Content type: VND.CONTEXTANALYSER.LOCATION

Columns:

Name	Type	Remarks
_id	long	A unique, persistent ID for the place
name	string	The name of the place
latitude	double	The latitude of the centre of the place
longitude	double	The longitude at the centre of the place
duration	long	Amount of time in seconds spent at the place
times	long	The number of times the place has been visited
lastvisit	long	The unix timestamp of the last visit (seconds)

E.4.4 Predictions

Allows querying of the current predictions, if any. Read only, does not accept selection, projection or order parameters.

URI: CONTENT://UK.CO.MD87.ANDROID.CONTEXTANALYSER.PREDICTIONSCONTENTPROVIDER/PREDICTION

Content type: VND.CONTEXTANALYSER.PREDICTION

Columns:

Name	Type	Remarks
_ID	long	A unique ID for the prediction
place	long	The ID of the predicted destination
count	int	The 'score' of the prediction

E.5 Context API

A small "API" is available to facilitate easier access to the services exposed by the Context Analyser. This consists of the ContextApi class and its assorted subclasses. These subclasses contain constant values for all column names, URIs and content types, as well as intent names and enumeration values.

The ContextApi.Intents class defines string constants for the three broadcast intents which are used by the Context Analyser. The ContextTypes inner class contains integer constants for the possible 'type' values for the CONTEXT_CHANGED intent.

The Places, Journeys, JourneySteps, Predictions and Activities classes all contain a CONTENT_URI field containing the Uri of the the content provider, a CONTENT_TYPE field containing the string mime type for that provider, and a static ColumnNames class which contains string constants for each column returned by the content provider.

The following code from the Places application shows the use of these constant values:

```
final Cursor cursor = managedQuery(ContextApi.Places.CONTENT_URI,
         new String[] { ColumnNames.LATITUDE, ColumnNames.LONGITUDE,
         ColumnNames._ID, ColumnNames.NAME, ColumnNames.LAST_VISIT,
         ColumnNames.VISIT_COUNT }, null, null, null);
if (cursor.moveToFirst()) {
    final int nameColumn = cursor.getColumnIndex(ColumnNames.NAME);
    final int idColumn = cursor.getColumnIndex(ColumnNames._ID);
    do {
        final String name = cursor.getString(nameColumn);
        final int id = cursor.getInt(idColumn);
        // etc
    } while (cursor.moveToNext());
}
```

Note that the code imports both the ContextApi class, and the ContextApi.Places.ColumnNames class.

F Extract from export of window data

Activity: CLASSIFIED/WALKING

```
1264518159452:-7.804459,-1.3620348,-0.55843425,99.4375,73.3125,-32.6875,
1264518159502:-8.853226,-0.53119355,-0.0,100.4375,72.0625,-33.6875,
1264518159557:-11.699879,0.8308412,-1.2803127,101.1875,70.8125,-32.9375,
1264518159601:-13.797412,-0.24516626,-1.1168685,101.4375,70.5625,-31.9375,
1264518159652:-10.59663,-4.8760843,1.96133,102.375,71.5625,-31.1875,
1264518159702:-10.528529,-5.3119354,0.6946377,102.125,71.5625,-29.6875,
1264518159751:-14.219643,-2.506144,2.4108016,102.375,69.5625,-28.0,
1264518159801:-9.275456,-1.2666923,1.253072,102.875,69.8125,-29.0,
1264518159852:-5.1621118,-0.7491191,0.27240697,103.125,69.8125,-30.6875,
1264518159901:-3.568531,-0.8036005,0.88532263,103.625,68.5625,-31.9375,
1264518159951:-4.4810944,-0.51757324,1.253072,103.625,69.0625,-31.6875,
1264518160001:-6.9191365,1.4028958,1.0487667,103.875,70.5625,-30.4375,
1264518160061:-13.007432,1.3484144,0.32688835,105.0625,70.5625,-29.0,
1264518160102:-16.42614,1.2394516,-0.10896278,104.375,71.3125,-27.25,
1264518160152:-15.908566,-2.3018389,-0.06810174,104.625,71.0625,-27.25,
1264518160202:-9.915613,-4.780742,1.6344417,105.3125,71.0625,-29.0,
1264518160251:-10.242501,-2.465283,2.73769,105.0625,70.3125,-30.4375,
1264518160301:-10.351464,-4.8897047,0.58567494,104.375,71.3125,-32.6875,
1264518160351:-7.600154,-2.0294318,0.81722087,102.375,73.0625,-33.6875,
1264518160401:-6.1291566,-1.0760075,0.7218784,101.1875,72.3125,-34.375,
1264518160451:-5.3936577,-1.5527196,0.3677494,101.1875,71.8125,-35.625,
1264518160502:-6.101916,-1.3892754,-0.23154591,100.6875,72.0625,-35.625,
1264518160551:-7.2187843,-1.525479,-0.8308412,100.9375,68.0625,-36.625,
1264518160694:-15.227549,-2.2609777,0.6401563,101.4375,68.0625,-36.625,
1264518160697:-15.227549,-2.2609777,0.6401563,101.4375,68.0625,-36.625,
1264518160701:-15.227549,-2.2609777,0.6401563,101.4375,68.0625,-36.625,
1264518160752:-9.956474,-3.840938,3.405087,103.125,68.8125,-36.375,
1264518160802:-12.598822,-5.298315,0.32688835,103.875,69.3125,-34.875,
1264518160858:-13.756551,-2.9283748,2.5333846,105.0625,69.3125,-32.6875,
1264518160901:-8.962189,-1.6344417,0.9942854,105.5625,68.0625,-34.125,
1264518160951:-4.399372,-1.4709976,0.54481393,104.625,68.8125,-35.875,
1264518161001:-3.050958,-1.4573772,0.58567494,105.0625,68.5625,-35.875,
1264518161051:-4.5491962,0.19068487,0.6537767,105.0625,67.8125,-35.375,
1264518161101:-6.7965536,1.3484144,0.87170225,105.5625,69.3125,-33.4375,
1264518161151:-12.993812,1.920469,1.253072,107.3125,70.3125,-30.4375,
1264518161201:-16.303556,0.53119355,0.14982383,106.8125,71.0625,-29.5,
1264518161251:-14.09706,-4.7535014,0.20430522,105.5625,71.0625,-29.0,
1264518161301:-10.746454,-4.971427,1.4709976,105.5625,70.3125,-30.1875,
1264518161351:-11.34575,-4.0588636,1.525479,104.125,70.5625,-32.9375,
1264518161401:-8.989429,-4.930566,0.88532263,103.375,70.8125,-35.125,
1264518161451:-6.4560447,-2.1111538,0.47671217,101.625,71.3125,-35.125,
```

G Extract from ARFF representation

```
    @RELATION activity
    @ATTRIBUTE "Absolute Mean (series 0)" numeric
    @ATTRIBUTE "Absolute Mean (series 1)" numeric
    @ATTRIBUTE "Absolute Mean (series 2)" numeric
    @ATTRIBUTE "Absolute Mean (series 3)" numeric
    @ATTRIBUTE "Absolute Mean (series 4)" numeric
    @ATTRIBUTE "Absolute Mean (series 5)" numeric
    @ATTRIBUTE "Maximum (series 0)" numeric
    @ATTRIBUTE "Maximum (series 1)" numeric
    @ATTRIBUTE "Maximum (series 2)" numeric
    @ATTRIBUTE "Maximum (series 3)" numeric
    @ATTRIBUTE "Maximum (series 4)" numeric
    @ATTRIBUTE "Maximum (series 5)" numeric
    @ATTRIBUTE "Mean (series 0)" numeric
    @ATTRIBUTE "Mean (series 1)" numeric
    @ATTRIBUTE "Mean (series 2)" numeric
    @ATTRIBUTE "Mean (series 3)" numeric
    @ATTRIBUTE "Mean (series 4)" numeric
    @ATTRIBUTE "Mean (series 5)" numeric
    @ATTRIBUTE "Median (series 0)" numeric
    @ATTRIBUTE "Median (series 1)" numeric
    @ATTRIBUTE "Median (series 2)" numeric
    @ATTRIBUTE "Median (series 3)" numeric
    @ATTRIBUTE "Median (series 4)" numeric
    @ATTRIBUTE "Median (series 5)" numeric
    @ATTRIBUTE "Minimum (series 0)" numeric
    @ATTRIBUTE "Minimum (series 1)" numeric
    @ATTRIBUTE "Minimum (series 2)" numeric
    @ATTRIBUTE "Minimum (series 3)" numeric
    @ATTRIBUTE "Minimum (series 4)" numeric
    @ATTRIBUTE "Minimum (series 5)" numeric
    @ATTRIBUTE "Range (series 0)" numeric
    @ATTRIBUTE "Range (series 1)" numeric
    @ATTRIBUTE "Range (series 2)" numeric
    @ATTRIBUTE "Range (series 3)" numeric
    @ATTRIBUTE "Range (series 4)" numeric
    @ATTRIBUTE "Range (series 5)" numeric
    @ATTRIBUTE classification {CLASSIFIED/VEHICLE/CAR, CLASSIFIED/IDLE/STANDING,
CLASSIFIED/VEHICLE/BUS, CLASSIFIED/WALKING, CLASSIFIED/IDLE/SITTING,
CLASSIFIED/WALKING/STAIRS/DOWN, CLASSIFIED/DANCING, CLAS-
SIFIED/WALKING/STAIRS/UP}
    @DATA
    0.44361898, 4.7165775, 8.409821, 10.608887, 10.081055, 16.005371, 5.5162406,
9.479762, 11.35937, 15.125, 1.4E-45, 1.4E-45, -0.44361898, 4.7165775, 8.409821,
10.608887, -10.081055, -16.005371, -0.19068487, 4.9169455, 8.7034025, 10.9375,
-9.875, -15.6875, -3.8273177, 1.4437568, 3.4187074, -0.0625, -18.6875, -23.0625,
9.343558, 8.036005, 7.940663, 15.1875, 18.6875, 23.0625, CLASSIFIED/VEHICLE/CAR
    3.9024422, 2.722155, 8.665946, 12.640137, 3.3691406, 15.688477, 1.4E-45,
4.6036777, 15.649779, 15.125, 1.4E-45, 1.4E-45, -3.9024422, 2.722155, 8.665946,
12.640137, -3.3691406, -15.688477, -3.840938, 2.7240696, 8.499097, 12.6875, -
3.375, -15.6875, -7.518432, 1.0351465, 4.69902, 8.75,-5.875, -17.625, 7.518432,
3.5685313, 10.95076, 6.375, 5.875, 17.625, CLASSIFIED/VEHICLE/CAR
    2.9535933, 1.9287692, 9.193417, 8.516602, 3.9960938, 16.75, 1.4E-45, 3.173541,
14.669114, 11.1875, 1.4E-45, 1.4E-45, -2.9535933, 1.9287692, 9.193417, 8.516602,
-3.9960938, -16.75, -2.901134, 2.083913, 9.275456, 8.5, -3.875, -16.875, -5.0803895,
0.50395286, 4.6445384, 5.5625, -6.625, -18.875, 5.0803895, 2.669588, 10.024576,
5.625, 6.625, 18.875, CLASSIFIED/VEHICLE/CAR
    3.7644293, 0.39350045, 9.493064, 4.345215, 4.4628906, 17.933105, 1.4E-45,
2.002191, 15.704261, 10.75, 1.4E-45, 1.4E-45, -3.7644293, 0.39350045, 9.493064,
```

4.345215, -4.4628906, -17.933105, -3.5957718, 0.50395286, 9.493382, 4.8125, -4.375, -17.875, -8.567199, -1.3756552, 6.006573, -2.0, -7.125, -20.5625, 8.567199, 3.3778462, 9.697687, 12.75, 7.125, 20.5625, CLASSIFIED/VEHICLE/CAR

1.8751402, 2.4263377, 9.45008, 0.78222656, 5.080078, 16.560059, 1.4E-45, 3.8954194, 9.874752, 2.625, 1.4E-45, 1.4E-45, -1.8751402, 2.4263377, 9.45008, 0.78222656, -5.080078, -16.560059, -1.8387469, 2.587866, 9.425281, 0.9375, -5.125, -16.625, -2.4789033, 1.334794, 9.220976, -1.3125, -6.625, -17.875, 2.4789033, 2.5606253, 0.65377617, 3.9375, 6.625, 17.875, CLASSIFIED/VEHICLE/CAR

1.7938437, 2.6545851, 9.423793, 0.110839844, 4.611328, 16.045898, 1.4E-45, 4.099725, 9.697687, 1.625, 1.4E-45, 1.4E-45, -1.7938437, 2.6545851, 9.423793, 0.110839844, -4.611328, -16.045898, -1.8115063, 2.7240696, 9.41166, 0.1875, -4.625, -16.125, -2.5333846, 1.4437568, 9.248216, -1.0625, -5.625, -17.375, 2.5333846, 2.655968, 0.44947147, 2.6875, 5.625, 17.375, CLASSIFIED/VEHICLE/CAR

H User-annotated Sensor Logger Results

Count	User annotation	On-device classification
3		CLASSIFIED/DANCING
6		CLASSIFIED/IDLE/SITTING
2		CLASSIFIED/IDLE/STANDING
9		CLASSIFIED/UNKNOWN
31		CLASSIFIED/VEHICLE/BUS
11		CLASSIFIED/VEHICLE/CAR
1	10	CLASSIFIED/IDLE/SITTING
1	beed	CLASSIFIED/VEHICLE/BUS
1	breathing	CLASSIFIED/VEHICLE/BUS
1	cycle	CLASSIFIED/VEHICLE/BUS
1	Dancing	CLASSIFIED/WALKING
1	driving	CLASSIFIED/WALKING/STAIRS/DOWN
1	drunkenly going to the bathroom to pee.	CLASSIFIED/VEHICLE/CAR
1	fixing my clothes	CLASSIFIED/VEHICLE/BUS
1	hhhg	CLASSIFIED/VEHICLE/BUS
2	In a car	CLASSIFIED/VEHICLE/BUS
1	in a house	CLASSIFIED/VEHICLE/CAR
4	in bed	CLASSIFIED/VEHICLE/BUS
2	it is a cat	CLASSIFIED/VEHICLE/BUS
1	kitchen work	CLASSIFIED/IDLE/STANDING
1	laundry	CLASSIFIED/VEHICLE/CAR
1	lay in bed	CLASSIFIED/VEHICLE/BUS
1	layin down	CLASSIFIED/IDLE/STANDING
1	layin in bed	CLASSIFIED/VEHICLE/BUS
3	laying down	CLASSIFIED/VEHICLE/BUS
1	laying down	CLASSIFIED/VEHICLE/CAR
1	laying down in bed	CLASSIFIED/VEHICLE/BUS
3	laying in bed	CLASSIFIED/VEHICLE/BUS
1	laying in bed with the phone on my stomach	CLASSIFIED/VEHICLE/CAR
1	lie in bed	CLASSIFIED/VEHICLE/BUS
1	ligger pa golvet	CLASSIFIED/VEHICLE/BUS
1	love	CLASSIFIED/VEHICLE/CAR
1	lying in bed	CLASSIFIED/VEHICLE/BUS
1	moving phone	CLASSIFIED/WALKING
1	Nothing at all!	CLASSIFIED/UNKNOWN
1	passenger in car straight road 50mph	CLASSIFIED/VEHICLE/BUS
1	playin gamw	CLASSIFIED/VEHICLE/BUS
1	pooping	CLASSIFIED/UNKNOWN
1	Rolling around	CLASSIFIED/WALKING
1	Rotating the phone	CLASSIFIED/WALKING/STAIRS/UP
1	seating down	CLASSIFIED/VEHICLE/BUS
1	seting	CLASSIFIED/VEHICLE/BUS
1	shake the device	CLASSIFIED/VEHICLE/CAR
1	shaking my leg sitting on my bed	CLASSIFIED/VEHICLE/BUS

Count	**User annotation**	**On-device classification**
1	shaking phone violently!!!	CLASSIFIED/WALKING/STAIRS/UP
1	siq	CLASSIFIED/VEHICLE/BUS
1	sitti	CLASSIFIED/VEHICLE/BUS
1	Sitting	CLASSIFIED/IDLE/SITTING
1	Sitting	CLASSIFIED/IDLE/STANDING
19	Sitting	CLASSIFIED/VEHICLE/BUS
6	Sitting	CLASSIFIED/VEHICLE/CAR
1	sitting at my desk	CLASSIFIED/VEHICLE/BUS
1	Sitting down	CLASSIFIED/VEHICLE/BUS
1	sitting on my ass	CLASSIFIED/VEHICLE/CAR
1	sitting on the couch	CLASSIFIED/VEHICLE/BUS
1	sjxjxgzog	CLASSIFIED/VEHICLE/BUS
1	sleep	CLASSIFIED/IDLE/SITTING
2	sleeping	CLASSIFIED/VEHICLE/BUS
1	Standing	CLASSIFIED/VEHICLE/BUS
1	Standing	CLASSIFIED/VEHICLE/CAR
1	swinging	CLASSIFIED/IDLE/SITTING
1	test	CLASSIFIED/VEHICLE/CAR
2	train	CLASSIFIED/VEHICLE/BUS
1	traveling by bus	CLASSIFIED/VEHICLE/CAR
1	traveling by car	CLASSIFIED/VEHICLE/BUS
1	travelling by bus	CLASSIFIED/VEHICLE/CAR
2	travelling by s line	CLASSIFIED/VEHICLE/BUS
3	UNCLASSIFIED/NOTCORRECTED	CLASSIFIED/DANCING
69	UNCLASSIFIED/NOTCORRECTED	CLASSIFIED/IDLE/SITTING
29	UNCLASSIFIED/NOTCORRECTED	CLASSIFIED/IDLE/STANDING
32	UNCLASSIFIED/NOTCORRECTED	CLASSIFIED/VEHICLE/BUS
14	UNCLASSIFIED/NOTCORRECTED	CLASSIFIED/VEHICLE/CAR
16	UNCLASSIFIED/NOTCORRECTED	CLASSIFIED/WALKING
5	Walking	CLASSIFIED/VEHICLE/BUS
1	Walking	CLASSIFIED/VEHICLE/CAR
1	Walking (downs tairs)	CLASSIFIED/WALKING/STAIRS/UP
1	walking in a store	CLASSIFIED/VEHICLE/CAR
1	walking in circles	CLASSIFIED/VEHICLE/CAR
1	watching amovie!	CLASSIFIED/VEHICLE/BUS
1	watching TV	CLASSIFIED/VEHICLE/BUS
1	work in the airport	CLASSIFIED/VEHICLE/CAR
1	z	CLASSIFIED/VEHICLE/BUS

Tab. 4: User-annotated SENSOR LOGGER results

References

[1] http://www.changewaveresearch.com/articles/2010/01/smart_phone_20100104.html.

[2] http://www.computerworld.com/s/article/9139026/.

[3] L. Bao and S. S. Intille. Activity recognition from user-annotated acceleration data. *Pervasive Computing*, 3001:1–17, 2004.

[4] P. Bellavista, A. Kupper, et al. Location-based services: Back to the future. 7(2):85–89, April–June 2008.

[5] V. Bellotti, B. Begole, et al. Activity-based serendipitous recommendations with the magitti mobile leisure guide. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pp. 1157–1166. ACM, New York, NY, USA, 2008. ISBN 9781605580111.

[6] J. Carós, O. Chételat, et al. Very low complexity algorithm for ambulatory activity classification. In *European Medical & Biological Engineering Conference and IFMBE European Conference on Biomedical Engineering*. 2005.

[7] T. Choudhury, S. Consolvo, et al. The mobile sensing platform: An embedded activity recognition system. 7(2):32–41, April–June 2008.

[8] S. Consolvo, D. W. Mcdonald, et al. Activity sensing in the wild: a field trial of ubifit garden. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pp. 1797–1806. ACM, New York, NY, USA, 2008. ISBN 9781605580111.

[9] S. Dornbush, K. Fisher, et al. Xpod - a human activity and emotion aware mobile music player. In *Proc. 2nd International Conference on Mobile Technology, Applications and Systems*, pp. 1–6. 2005.

[10] N. Eagle and A. Pentland. Mobile matchmaking: Proximity sensing and cueing, 2004.

[11] A. Garakani. *Real-Time Classification of Everyday Fitness Activities on Windows Mobile*. Master's thesis, University of Washington, 2009.

[12] J. Han and M. Kamber. *Data mining: concepts and techniques*, chap. 6, pp. 348–350. Morgan Kaufmann, 2006.

[13] A. Hein and T. Kirste. Towards recognizing abstract activities: An unsupervised approach, 2008.

[14] J. Hightower, S. Consolvo, et al. Learning and recognizing the places we go. In *UbiComp 2005: Ubiquitous Computing*, pp. 159–176. 2005.

[15] S. Hudson, J. Fogarty, et al. Predicting human interruptibility with sensors: a wizard of oz feasibility study. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 257–264. ACM Press, 2003.

[16] T. Huynh and B. Schiele. Analyzing features for activity recognition. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pp. 159–163. ACM, New York, NY, USA, 2005. ISBN 1-59593-304-2.

[17] L. Liao, D. J. Patterson, et al. Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311–331, April 2007.

[18] M. Mathie, B. Celler, et al. Classification of basic daily movements using a triaxial accelerometer. *Medical and Biological Engineering and Computing*, 42(5):679–687, September 2004.

[19] U. Maurer, A. Rowe, et al. ewatch: a wearable sensor and notification platform. In *Proc. International Workshop on Wearable and Implantable Body Sensor Networks BSN 2006*, pp. 4 pp.–145. 2006.

[20] E. Miluzzo, J. Oakley, et al. Evaluating the iphone as a mobile platform for people-centric sensing applications, 2009.

[21] T. Nicolai, N. Behrens, et al. Exploring social context with the wireless rope. In *In Proc. Workshop MONET: LNCS 4277*. 2006.

[22] P. Nurmi and J. Koolwaaij. Identifying meaningful locations. In *Mobile and Ubiquitous Systems: Networking & Services, 2006 Third Annual International Conference on*, pp. 1–8. 2006.

[23] M. Raento, A. Oulasvirta, et al. Contextphone: A prototyping platform for context-aware mobile applications. *IEEE Pervasive Computing*, 4(2):51–59, April 2005. ISSN 1536-1268.

[24] F. Reynolds. Camera phones: A snapshot of research and applications. 7(2):16–19, April–June 2008.

[25] B. Schilit, N. Adams, et al. Context-aware computing applications, 1994.

[26] A. Schmidt, K. A. Aidoo, et al. Advanced interaction in context. *Lecture Notes in Computer Science*, 1707:89–??, 1999.

[27] —. ilearn on the iphone: Real-time human activity classification on commodity mobile phones, 2008.

[28] J. Shen. Machine learning for activity recognition, 2004.

[29] D. Siewiorek, A. Smailagic, et al. Sensay: a context-aware mobile phone. In *Wearable Computers, 2003. Proceedings. Seventh IEEE International Symposium on*, pp. 248–249. 2003.

[30] K. Song and Y. Wang. Remote activity monitoring of the elderly using a two-axis accelerometer. In *CACS Automatic Control Conference*. 2005.

[31] T. Stiefmeier, D. Roggen, et al. Wearable activity tracking in car manufacturing. 7(2):42–50, April–June 2008.

[32] M. Tentori and J. Favela. Activity-aware computing for healthcare. 7(2):51–57, April–June 2008.

[33] Y. Wang, J. Lin, et al. A framework of energy efficient mobile sensing for automatic user state recognition. In *MobiSys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services*, pp. 179–192. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-566-6.

[34] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and. Techniques with Java Implementations*, chap. 3, pp. 72–75. Morgan Kaufmann, 2000.