

Providing User Context for Mobile and Social Networking Applications

André C. Santos^a, João M. P. Cardoso^b, Diogo R. Ferreira^a, Pedro C. Diniz^a, Paulo Cháinho^c

^a*IST – Technical University of Lisbon, Avenida Prof. Dr. Cavaco Silva, 2744-016 Porto Salvo, Portugal*

^b*Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal*

^c*PT Inovação, S.A., Taguspark Edifício Tecnologia II, 31, 2780-920 Porto Salvo, Portugal*

Abstract

The processing capabilities of mobile devices coupled with portable and wearable sensors provide the basis for new context-aware services and applications tailored to the user environment and its daily activities. In this article, we describe the approach developed within the UPCASE project, which makes use of sensors available in the mobile device as well as sensors externally connected via Bluetooth to provide user contexts. We describe the system architecture from sensor data acquisition to feature extraction, context inference and the publication of context information in web-centered servers that support well-known social networking services. In the current prototype, context inference is based on decision trees to learn and to identify contexts dynamically at run-time, but the middleware allows the integration of different inference engines if necessary. Experimental results in a real-world setting suggest that the proposed solution is a promising approach to provide user context to local mobile applications as well as to network-level applications such as social networking services.

Key words: Context-aware services, context inference, mobile applications, wearable sensors, decision trees, social networking

1. Introduction

There is a growing desire of telecommunication operators to increase traffic volume by offering value-added services to customers in addition to traditional voice and data communication. Some of these services must be enabled or disabled depending on the actual user context. For example, if the user is driving then a service could automatically provide an estimate for the rush-hour delay to reach the desired destination; or, when given certain sound and lighting conditions, a service could be configured not to disturb or interact with the user. Other examples include anti-theft or near-emergency services, which can be triggered under specific contexts; for example, by using sensors it might be possible to determine whether an elderly has fallen at home and has not moved for some time thus triggering an emergency call.

To enable such kind of services, mobile devices must be able to clearly and accurately identify specific user contexts [1, 2]. For this purpose, mobile devices can be augmented with sensors that yield data such as position, lighting or sound, from which the user context can be determined. Accurate context inference, however, is notoriously difficult as there are various sources of noisy data signals with possibly very distinct

patterns which need to be captured and processed in a timely fashion. Furthermore, the amount of raw sensor data can overwhelm the resources of even the most sophisticated mobile devices. A possible solution would require each mobile device to acquire and transmit sensor data to a centralized server for processing. Although conceptually simple, this centralized solution is infeasible as it would require constant communication and impose network latency on devices that need to operate in real-time. It would also require excessive computing power for the device to constantly transmit a possibly high volume of sensor data, therefore rapidly draining the device's battery. On the server side, fusing sensor data from millions of devices would require tremendous computing power. Instead, each mobile unit should be able to infer user context by processing data originating from its sensors and possibly from communicating with network services to obtain additional complementary information such as traffic or weather conditions.

In this article, we describe the architecture, operation and potential applications of a prototype system developed within the UPCASE project (User-Programmable Context-Aware Services) [3], a research project funded by a national telecom operator. The system provides context inference based on a smartphone augmented with an array of sensors connected via *bluetooth*¹. The system architecture comprises four main layers: (1) the acquisition layer which is concerned with sensor data acquisition and preprocessing, (2) the feature extraction layer which assigns specific value categories to the preprocessed sensor data, (3) the context inference layer which at the moment uses decision-tree induction techniques to determine the user context, and (4) the context publication layer used to provide context information to network-level services.

A preliminary version of the UPCASE system has been presented in [3]. Here, we describe the current system prototype as well as its applications, together with results regarding the real-time performance of the system, including battery lifetime of the devices involved. The system is able to identify user activities and publish this context information on remote servers that can support different kinds of services. In particular, publishing the user context to social networking services provides users with an easy mechanism to keep in touch with their peers. The success of social networking services such as *Facebook*² and *Twitter*³ and their use via mobile devices show an increasing interest and potential in providing users with mechanisms to communicate their daily activities.

The article is organized as follows. We begin with a description of related work in the area of sensing and context inference. We then present the approach and the system developed in the UPCASE project: its sensors, architecture, and operation through data acquisition, preprocessing, feature extraction, and context inference. Afterwards we present a set of experimental measures of operating characteristics. Finally, we describe the application of the system to publish user context to well-known social networking services.

¹<http://www.bluetooth.com/>

²<http://www.facebook.com/>

³<http://www.twitter.com/>

2. Related Work

Context identification has been recognized as an enabling technology for proactive applications and context-aware computing [4, 5]. Sensor networks can be used to capture intelligence (as in the e-SENSE project [6]), providing sensing capabilities from the environment and opening opportunities for context-aware services.

Early context-aware applications were predominantly based on user location defined as typical user places (e.g., “at home”, “in a museum”, “in a shopping center”). Projects such as GUIDE [7] and Cyberguide [8] addressed the use of information about location and situation to guide the user when visiting tourist attractions. Recent work has studied techniques to identify a richer set of contexts or activities. These include simple user activities (e.g., “walking”, “running”, “standing”), environment characteristics (e.g., “cold”, “warm”), or even emotional condition of the user (e.g., “happy”, “sad”, “nervous”).

Usually, the identification of contexts is performed in several stages. Processing raw data from sensors may require a wide variety of techniques such as noise reduction, mean and variance calculation, time- and frequency-domain transformations, estimation of time series, and/or sensor fusion. Data collected from sensors goes through feature extraction so that context-inference stages makes use of features rather than raw data. Context inference itself has been addressed using different techniques such as Kohonen Self-Organizing Maps (KSOMs) [9], k-Nearest Neighbor [10], Neural Networks [11], and Hidden Markov Models (HMMs) [12]. Some approaches also combine several of these techniques, as described in [13].

Regarding the inference of user activities such as “walking” or “running”, there have been a myriad of approaches, ranging from simple processing steps and threshold operations [14, 15, 2] to the use of neural networks as a clustering algorithm [11]; or even using non-supervised time-series segmentation [16]. As an example, the work presented in [1] infers activities such as “walking”, “running”, “standing”, and “sitting” with a single 3-axis accelerometer, claiming an accuracy of 96%.

The SenSay [17] project developed a smartphone prototype able to take advantage of the user context to improve usability. For example, if the user is busy and wishes not to be disturbed, the smartphone can answer/reply automatically using an SMS (Short Message Service). The SenSay prototype uses a smartphone and a sensor unit consisting of a 3-axis accelerometer, two microphones (one to capture sound from the environment and the other to capture voice from the user), and a light sensor. The prototype makes use of simple techniques such as performing an average of sensor readings over a given window and applying a numeric threshold to identify each activity. This and other system prototypes are summarized in Table 1 for comparison.

With the increasing popularity of social networking services, context information finds a new role in enabling interaction between members in a community. In [19], for example, the authors present the Meeting Room Assistant application, a hyper-local facilitator where context is used to enrich communication between participants in real-life meetings, in a similar way to what social networking services achieve on the web. In the CenceMe system [20], the inference of user behavior is used to share this information through a private social

System	Goals	Pre-processing	Context Inference	Contexts	Applications	Device	Sensors	Sensor Node	Sampling Rates	Communication Node-Device	Context Publication
SensSys(6) [17]	improve mobile phone usability	average over window of samples (5-30 secs)	two thresholds over 10min of sensor data	sitting, walking and running	context-aware user interface, phone configuration	notebook	3-axis accel., 2 micros, light, temperature	in-house module	1 Hz	RS-232	GPRS
SensSys(b) [13]	improve mobile phone usability	average, SAD, FFT, normalization, PCA	KSOMs, k-means clustering and second-order Markov model	working, walking, cooking, eating, sleeping, reading	context-aware user interface, phone configuration	PDA and Fujitsu LifeBook	microphones, GPS, 2-axis accel., Bluetooth Spoon headset, internal clock	SenseWear armband from BodyMedia	32 Hz	802.11b (SenseWear-LifeBook), Bluetooth (LifeBook-PDA)	GPRS
[1]	physical daily-life activities	12-sample variance; 4-sample average per axis; max. value FFT power spectrum	average, variance, FFT, angles, phone position	walking, running, standing, sitting, leaning + phone position	e-coaching, messages based on physical conditions and exercise	PC	3-axis accel.	Pavenet module physically connected to phone	100 Hz	Bluetooth	—
[18]	daily-life activities	hardware noise filters, mean	thresholds over DSYM; average per axis	standing, lying, sitting, running, walking, falling	monitor activity patterns, recognize emergency situations	PC	3-axis accel.	TIP700CM ZigBee-compatible sensor node	100 Hz	ZigBee	—
[14]	healthcare, medical symptoms	RMS, integral, variance	thresholds	walking, running, sitting	checklist-based Wellness Diary	2 HP iPAQ (PDA)	ECG monitor, 2-axis accel.	in-house board	—	Bluetooth	—
CoCo [15]	user navigation, recommendations, content delivery	average, variance, power spectrum	thresholds	sitting, standing, walking, running, gestures, environment	navigation in large book market	laptop w/ MIB510CA receiver	2-axis accel., GPS, brightness, temperature, humidity, UV, motion, alcohol	Mica2dot module + sensor array	8.3 Hz	proprietary protocol (Mica2dot) + RS-232 (sensor array)	—
Cyberjacket [11]	providing information based on user location/context	RMS, integral (2 secs.)	Neural Network for clustering	walking, running, sitting, standing, upstairs, downstairs	tourist guide, web pages and audio based on user location	Matscom onHand PC	2-axis accel., GPS	Crossbow board, neural network chip, handheld display, audio interface	5 Hz	RS-232	—
MSP [2]	identify mode of transport and significant places	Goertzel algorithm, DFT coefficients w/ second-order filter	threshold over sum of DFT values of accel. signals	driving, running, walking, speed	classify mode of transport; highlight significant places	GSM phone, waist-worn, WiFi-enabled PDA	temperature, humidity, 3-axis accel., compass, light, microphone, barometer	multi-modal sensor board (MSB)	30 Hz	USB cable (MSB-PDA) and Bluetooth (phone-PDA)	—
UPCASE [3]	recognize user context	thresholds, mean, variance, FFT	decision trees	walking, running, standing, lying, inside, outside	publication to server and network services	smartphone	two 3-axis accel., humidity, light, temperature, sound, GPS, time	BlueSentry module	≤ 4 Hz ≤ 20 Hz (internal accelerometer)	Bluetooth	WiFi, Bluetooth, or GPRS

Table 1: Comparison of several system prototypes for context identification. The last row represents the prototype developed in this work.

networking service. The CenceMe system consists in an application running on a mobile phone and a back-end infrastructure hosted on server machines. The phone is responsible for sensing, classifying the raw sensed data, and uploading the user status to the back-end servers. Classification algorithms that infer more complex forms of sensing presence run on the back-end machines.

In the work presented in this article, we extract signal features using techniques similar to those described in [2, 15]. For context inference we combine signal-processing and machine-learning techniques, using decision trees [21] to fuse features and to identify user activities. All data preprocessing and context inference is performed in real-time on the mobile device. The results can be sent to a server in order to aggregate information from multiple users, thus allowing for more advanced and possibly non-local context inferences. Our work focuses in particular on the publication of context information to well-known social networking services.

3. Approach Overview

Context information may be described by a set of objective and subjective facts about the current activity the user is engaged in. If the user is driving then it may be assumed that the user is inside a vehicle regularly moving, which can be ascertained by data captured by a set of sensors. If, on the other hand, the user is working then this context information can only be inferred from a set of sensor readings that are typically associated with that high-level activity. In any case, the underlying assumption is that provided with a sufficient number of diverse sensors it is possible to distinguish between different user high-level activities. The identification is based on the principle that a different action, a different environment, or even the presence of other people will result in a different set of sensor readings for the various kinds of sensors.

Mobile devices such as smartphones and PDAs provide a combination of communication and processing capabilities that can become extremely useful for context-aware applications. First, it is possible to connect other devices, such as sensors, to a mobile phone via bluetooth, for example. Second, current mobile phones have enough processing capabilities to handle sensor data and run simple applications on them. Finally, the phone is connected to a network where users can share information and interact with remote services. The fact that the mobile phone is a personal item that the user is already accustomed to bring along makes it an ideal platform for the deployment of context-aware applications.

Context information can be inferred locally based on the capabilities of the mobile device coupled with a number of sensors. The identified user context can then be published to the network in order to configure, trigger or interact with remote services. Figure 1 shows the main stages involved in capturing the user context and making it available to other applications. Ordinary users make use of standard services provided by the network; advanced users with context-aware platforms can make use of additional services that are automatically customized to their present context, and can share their context with other users on the network, effectively enabling social-networking applications.

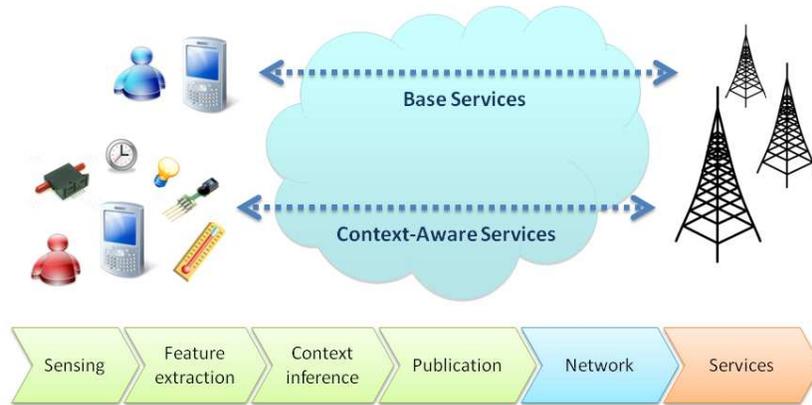


Figure 1: Overview of the context inference approach for mobile applications.

Context inference is based on several stages, from reading sensor values to actually using them to determine user context. In general, as will be discussed in the next sections, sensors can be of different types and they may produce different kinds of signals. These signals must be pre-processed to extract features about the current user activity (e.g., user is moving or not moving, sound is loud, temperature is high). These features are then fed to rule engine that determines user context based on a set of preconditions (e.g., if the user is not moving and the sound level is moderate, then the context is “working”). This context is then published through the network to remote services.

Clearly, the set of possible contexts and the rules that determine the present context must be configured, adapted and tuned for each user. There is hardly a better way to define those rules than to let the user train the system itself during certain period. During the training period, the user will set the context manually and the system will keep record the sensor readings. This will allow the system to collect a set of training examples. After some time, the system has collected enough examples to be able to infer the user context using standard classification techniques. As would be expected, longer and more frequent training periods will allow for more accurate context inference results as well as more personalized tuning for the system user. For the time being, our prototype uses decision trees [21], but the system allows the integration of other inference engines if necessary. In the next section we start by describing the system sensors and prototype.

4. Sensors and Prototype

A key goal for this work is the ability to identify user context using a set of sensors connected to a mobile phone. The sensors should be relatively small in order to be embedded in clothes or personal items such as backpacks or purses. Sensors include accelerometers, light, sound, humidity, temperature and GPS sensors, and also *virtual sensors* to acquire information such as time of day and calendar events, which can be retrieved directly from the mobile device.

Figure 2(a) depicts the main components of the system prototype: the mobile device, a sensor-aggregating node, and a set of sensors. The black box contains the batteries (the 1-Euro coin is shown to provide an idea of scale). Figures 2(b) and 2(c) depict experimental setups where the components are embedded in a backpack and on a vest, respectively. The vest prototype is an evolution from the earlier backpack model. These experimental prototypes were used for testing purposes and therefore have deliberately unconcealed sensors in order to better evaluate sensitivity to the environment by ensuring that the sensors experience approximately the same conditions as the user. In prototypes with concealed sensors one needs only to guarantee that some sensors in particular must be exposed to allow for more accurate measurements.

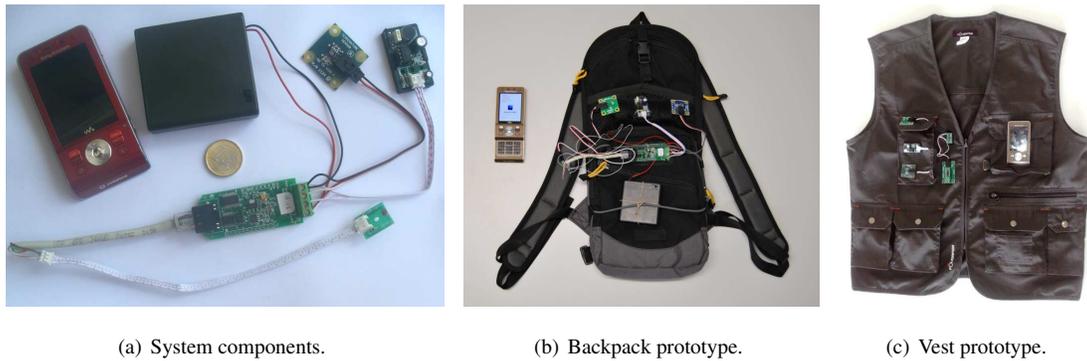


Figure 2: The system components and experimental prototypes mounted on a backpack and on a vest.

The system comprises either a Sony Ericsson W910i mobile phone⁴ or a Nokia N95 mobile phone⁵ and a BlueSentry external sensor node⁶. The BlueSentry sensor node communicates with the smartphone via bluetooth to provide sensor readings, thus avoiding the need for physical connection between the two. With respect to sensors, the vest prototype includes more sensors than the backpack. We used a sound sensor⁷, a temperature sensor⁸, a light sensor⁹, a humidity sensor¹⁰ and a 3-axis accelerometer¹¹, all wired to the sensor node. In addition to these, there are three other sensors being used, two that are internal to the smartphone – the internal accelerometer and a virtual time sensor to provide the time of day – and one external bluetooth GPS receiver¹² node.

⁴<http://www.sonyericsson.com/cws/products/mobilephones/overview/w910i>

⁵<http://europe.nokia.com/phones/n95>

⁶<http://www.rovingnetworks.com/bluesentry.php>

⁷<http://www.inexglobal.com/products.php?model=zxsound>

⁸http://www.phidgets.com/products.php?product_id=1124

⁹http://www.phidgets.com/documentation/Archive/1105.0.Product_Manual.pdf

¹⁰http://www.phidgets.com/documentation/Archive/1107.0.Product_Manual.pdf

¹¹<http://www.dimensionengineering.com/DE-ACCM3D.htm>

¹²http://www.globalsat.com.tw/eng/product_detail_00000039.htm

5. System Architecture

The system makes use of supervised learning techniques to determine user context. During a training period, the system collects a number of manually classified examples in order to induce a decision tree that will be used as a model for context identification. After this training phase the system operates autonomously and unobtrusively by automatically determining the context from the current sensor readings and the previously created model. The overall system architecture is presented in Figure 3.

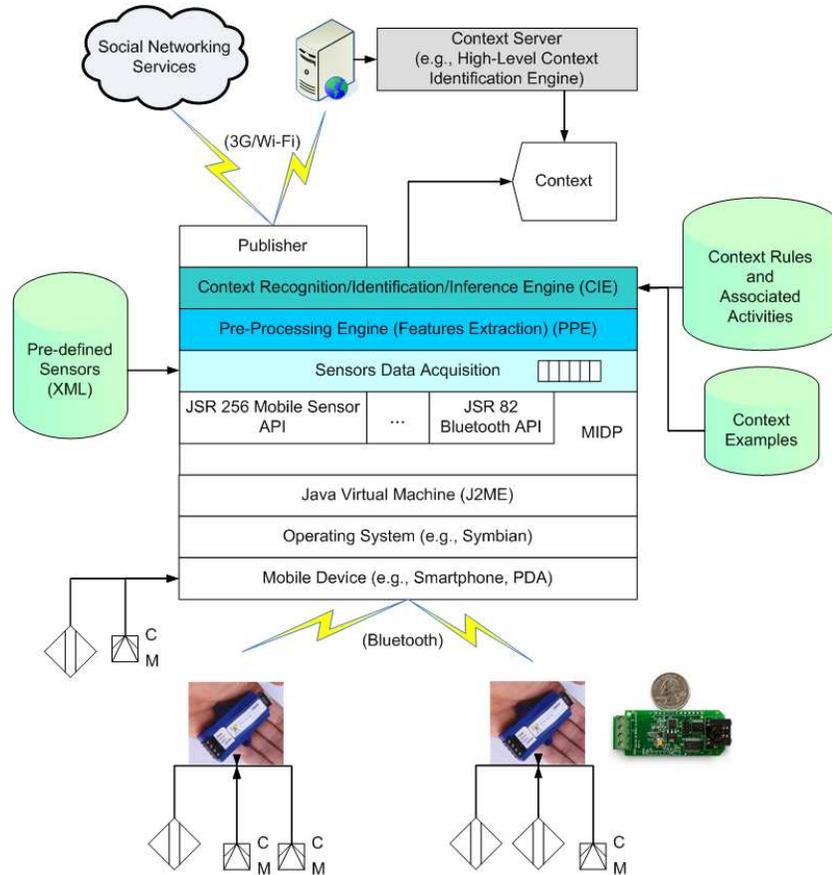


Figure 3: System architecture with layer description and communication connections.

At the lowest level, sensors gather data from the environment and provide it as raw analog signals to the sensor node, which in turn converts them to digital form and transmits them to the smartphone via bluetooth. The application layer has been developed using the Java ME platform¹³, a technology that is widely used due to its recognized portability across many mobile phone devices. The mobile phone runs a proprietary operating system which supports J2ME MIDlets. With the help of a Mobile Information Device Profile (MIDP), the

¹³<http://java.sun.com/javame/>

application acquires raw sensor data from both the internal sensors and external sensor nodes, namely the BlueSentry aggregating node and a bluetooth GPS sensor.

The system operates according to four main stages: a sensor data acquisition stage, a preprocessing plus feature extraction stage, a context inference stage and a publication stage. Sensor data acquired from the available sensors are fed to the preprocessing stage, which is responsible for extracting signal features to be used in the upper layers of the system architecture. The inference stage gathers these features and determines the user context according to a set of rules. These rules are available in the form of a decision tree that has been built during the training phase. Finally, the mobile device, upon having recognized a specific context, can publish that information to a server (subject to user permission) to trigger other actions or custom services, to provide statistical data about user behavior, or to automatically send a status update to social networking services.

These several stages can be seen as a process that maps low-level sensor data into high-level context information. The preprocessing, feature extraction, and the context inference stages can all be configured by specifying the available sensors and the default context rules. In addition, it is possible to add new sensors or replace existing ones by means of an XML configuration file, thus providing an easy and simple way of managing the set of available sensors.

5.1. Application Layer

Figure 4 provides some screenshots of the application running on the smartphone. It presents a simple but effective user interface, allowing different modes to be chosen from a list of available options (Figure 4(a)). It also includes the possibility of editing existing contexts (Figure 4(b)) and printing the decision tree for debugging purposes. Figure 4(c) presents an example of the initial configuration for the continuous context-learning mode which acquires sensor data during a period of time when a certain context is active. In Figure 4(d) it is possible to see the different sensor readings and the identified context, as well as a confidence value calculated as the percentage of total records for the displayed context within a fixed-size buffer window. A suggestive icon, when available, is also presented to the user.

5.2. Sensor Data Acquisition

Aiming at leveraging standard APIs, the system makes use of the JSR-256 Mobile Sensor API¹⁴ whenever it is supported by the mobile device. The JSR-256 API provides developers with a standard way to retrieve data not only from embedded sensors but also from sensors connected via infrared, bluetooth and GPRS. When the JSR-256 is not supported, the JSR-82 Bluetooth API is used¹⁵. At the moment, JSR-256 is being used only for the internal accelerometer in the Sony Ericsson W910i, and JSR-82 for all other data acquisition.

¹⁴<http://jcp.org/en/jsr/detail?id=256>

¹⁵<http://jcp.org/en/jsr/detail?id=82>

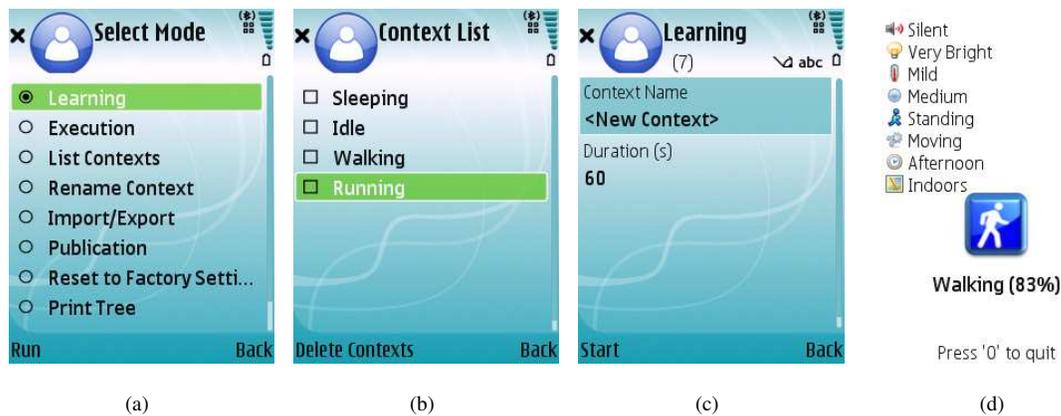


Figure 4: The application in selection mode (a), in context-editing mode (b), in learning mode (c) and in operation mode (d).

Sensor data are acquired at a fixed rate. At regular intervals, the smartphone sends a request to the sensor module in order to retrieve data from the sensors connected to it. The sensor readings are buffered in the smartphone in order to allow a time-frame window preprocessing stage. It is worth noting that sensors may have different acquisition rates. The difference in sampling frequency may force the acquisition to run at the slowest rate, or at individual rates for each sensor. Currently, the system is using the same acquisition rate for all sensors, except for the internal accelerometer of the smartphone, which is being sampled at twice the rate of the other sensors. Figure 5 depicts examples of raw sensor data acquired from sound, light, humidity, temperature and the two accelerometer sensors, together with the features extracted during the preprocessing stage described in the following section.

5.3. Preprocessing and Feature Extraction

The preprocessing stage prepares the raw sensor data to be converted into a set of finite value features or categories. While for some sensors the raw sensor value can be mapped directly to a category, other sensors such as the accelerometer need more elaborated preprocessing. Preprocessing typically involves a set of basic techniques such as averaging, filtering or transforming values, depending on the particular sensor the data came from. Rather than using instant values, the system captures a window of sensor readings and then pre-processes them in order to minimize jitter and to provide a more accurate categorization.

Table 2 presents the categories used for each sensor, as well as the value range for each category. Clearly, a limited number of categories places some boundaries on the total number of contexts that can be identified. However, a too large number of categories can also divide the perception of the environment in too many different states. According to the user contexts to be identified and the specific application domain, the categories of sensor values may have to be adjusted. In general, however, most applications will make use of only a limited set of contexts. We have found that the limited categories presented here are rich enough to support several practical scenarios.

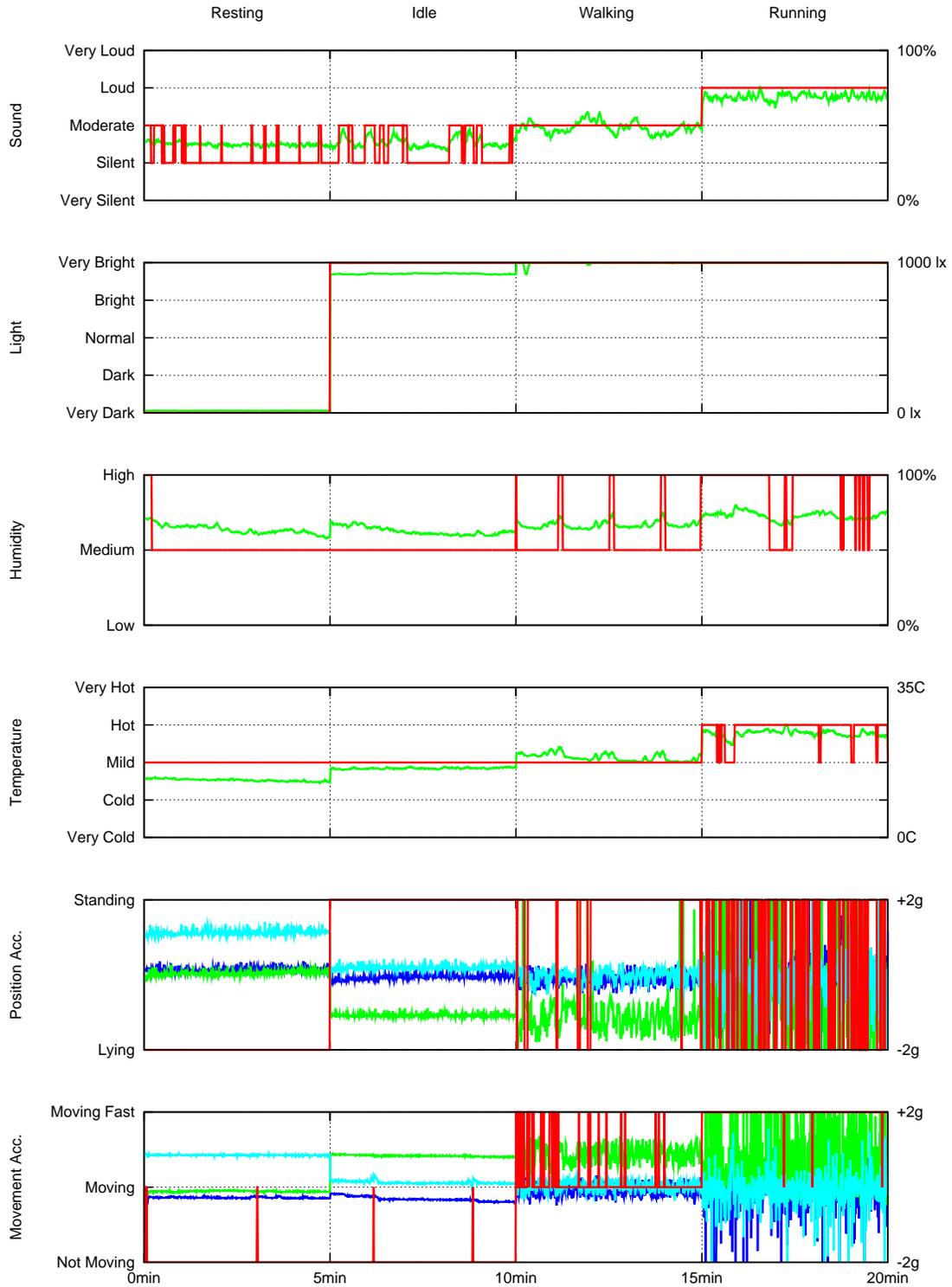


Figure 5: Raw sensor data and preprocessing result collected from several sensors during the training of different contexts. The rectangular waveforms denote the preprocessing result on the raw sensor data. The two y-axes on left and right are independent.

Sensor	Category	Calculation
sound	very silent	0% — 20%
	silent	20% — 40%
	moderate	40% — 60%
	loud	60% — 80%
	very loud	80% — 100%
light	very dark	0 lx — 200 lx
	dark	200 lx — 400 lx
	normal	400 lx — 600 lx
	bright	600 lx — 800 lx
	very bright	800 lx — 1000 lx
GPS	indoor	no signal
	outdoor	signal
position accelerometer	lying	x- or y-axis
	standing	z-axis

Sensor	Category	Calculation
temperature	very cold	-50° — 0°
	cold	0° — 15°
	mild	15° — 25°
	hot	25° — 30°
	very hot	30° — 150°
time	dawn	0h — 5h
	morning	6h — 11h
	afternoon	12h — 17h
	night	18h — 23h
movement accelerometer	not moving	low variance
	moving	high variance
	moving fast	variance- and FFT-based
humidity	low	0% — 30%
	medium	30% — 70%
	high	70% — 100%

Table 2: Categorization of sensor values after acquisition.

For most sensors, the preprocessing is based on applying thresholds to the mean value over an 8-sample buffer window. For the 3-axis movement accelerometer, the system computes the variance for each axis within a buffer window which includes the last 16 samples. The three variances obtained are compared to a threshold in order to identify “moving” and “not moving” activities. When movement is detected, the system performs an FFT (Fast Fourier Transform) over the last 32 samples and adds up the amplitude of the harmonics for frequencies within the range from 0.5Hz to 2Hz. If the resulting value is greater than a specific threshold value then a category of “moving fast” is reported. Otherwise, the output will be reported as “moving”.

6. Context Inference

At any given moment, the set of sensor readings will bear some relationship to the current user activity and the surrounding environment. The purpose of context inference is to discover this relationship so that when a similar set of readings occurs, the device will recognize the same context. While different contexts can lead to the same set of sensor readings, the converse is also true: different readings may correspond to the same context, as there will be naturally some variation in sensor readings during each activity. Context inference must therefore identify the range of readings that typically correspond to each particular context. It is important to note that although we have opted for the use of decision trees for context inference, the software architecture we have developed supports other classifiers as well.

6.1. Context Inference with Decision Trees

Our current prototype uses decision trees for the context inference engine. Decision trees are structures that fit the purpose of context inference. They are fast to build and process, making them attractive for im-

plementation on mobile devices. Provided with a set of training examples, decision tree induction produces a classification of contexts according to a set of features extracted from sensor readings. Our implementation is based on the ID3 algorithm [21], which uses information entropy to build the smallest tree that can correctly represent all branches. For comparison, we have also implemented the C4.5 algorithm [22].

The decision tree is first built from a set of default context rules. These initial rules are represented in XML so that they can be provided as a starting point for different user profiles (*e.g.*, a student profile, a sports profile, a worker profile). The tree is then updated in real-time as the user trains the system with new contexts. As the system is being trained, the whole tree is recomputed and updated with the contexts learned up to that point.

Figure 6 depicts an example of an induced decision tree. The tree has paths from root to leaf nodes that provide the rules for context identification based on features extracted from sensor readings. For example, if the user is moving, and the sound level is moderate, and the light is bright, then the context is “walking”. In this case, the movement accelerometer provides the most distinctive feature, followed by the sound sensor and only then by the light sensor. In general, the structure, the branches, the placement of sensors, and the particular contexts that appear in the leaf nodes depend on the training data that were used to induce the decision tree, and thus Figure 6 reflects the scenarios experienced by the user during training.

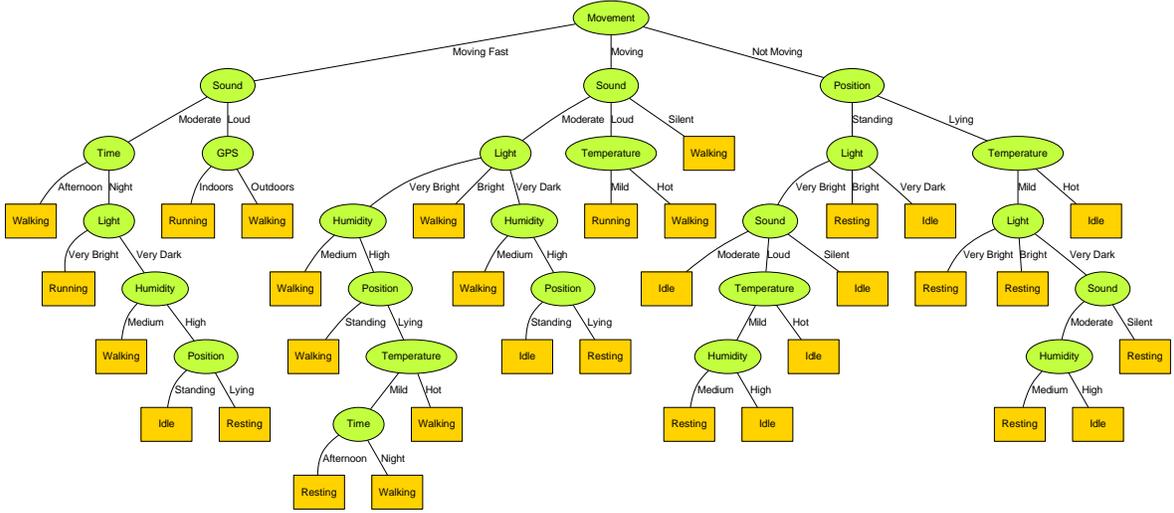


Figure 6: Example of an ID3 decision tree generated from training data for four different contexts: running, walking, resting and idle.

Learned contexts can be renamed or deleted, as the system provides means for the user to manage existing contexts. This way the user can erase a learned context and train the system again, or simply remove an unused context. Even if the context has been learned, the user can update the set of rules to better illustrate the conditions under which it should be detected. Whenever it becomes necessary, recomputing the decision tree is sufficiently fast to be done in real-time without the user noticing it.

The context identified via the decision tree is stored in a buffer which gathers a finite number of readings

and returns the context that has been recorded more often within that time window. This avoids momentary conditions that would make the context change unexpectedly, and also provides a mechanism for the system to assign a confidence level to the context that has been inferred. Together with sensor data buffering, this provides another layer to mitigate errors due to faulty sensor readings or due to activities that are detected for only a brief moment within a different context.

It is important to note, however, that the use of decision trees is just one possible approach for context inference. Although more advanced inference approaches can be used, decision trees have suited our needs as a proof of concept. The architecture of the middleware has been designed to allow an easy integration of other classification techniques.

6.2. Training Phase

The decision tree shown in Figure 6 can be generated from a set of training examples collected during a training period. When the user sets a context and a learning period as in Figure 4(c), the system collects sensor readings during that period and classifies the readings as training examples for the specified context. The training examples are then integrated into the inference rules if they do not yet exist. If the set of sensor readings is already associated with a context then we have a *collision*, which is handled by means of a counter for each rule. If collision occurs and the context is the same, then the counter for that rule is incremented; otherwise, if the contexts are different, then the new rule will be added to the system, thus overriding the previous one, but only if it has a higher counter value. The counter value for a new rule is calculated using the number of occurrences of that rule relative to the duration of the learning stage. Rules added to the system imply the update of the decision tree.

In one experiment [23] we have trained the system with four contexts – “walking”, “running”, “idle”, and “resting” – as depicted in Figure 7. These activities have been performed in four different environments, namely in the basement of the university building, in an empty classroom, in the main hall, and outside in open air and full daylight. A training period of five minutes was used to collect training data for each of the four activities in each of the four environments. Some of these data have already been shown in Figure 5. In five minutes, about 500 complete sensor readings have been collected, meaning that there are about 500 examples for each activity in each environment. Each example contains the set of sensor readings and the associated context, such as: *sound = moderate | light = very_bright | temperature = mild | humidity = medium | position = standing | movement = moving | time = afternoon | gps = indoors* → *context = walking*.

The set of examples for each activity in each environment was divided into a training set and a testing set, with roughly 300 examples for the first and 200 examples for the second. Given that there are four environments, there is a training set of $300 \times 4 = 1200$ examples for each activity, and a testing set of $200 \times 4 = 800$ examples for each activity. The training set was used to generate the decision tree, both with ID3 and C4.5. We have observed that the decision trees generated by ID3 are in general larger and deeper when compared to their C4.5

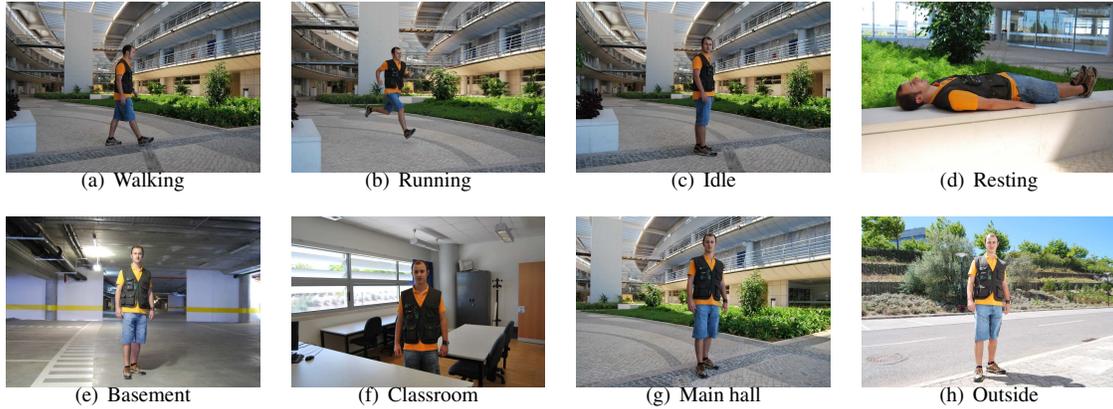


Figure 7: Activities representing the contexts to be recognized and the environments where they have been performed.

counterparts, and they tend to overfit the training data. Figure 6 shows the decision tree generated by ID3.

The testing set was used to assess the accuracy of the two classifiers in recognizing the user context. Table 3 shows the results. Both classifiers achieve a significantly high accuracy on the testing set. In general, the ID3 decision tree achieves a higher accuracy, and this has been confirmed in other similar experiments. Most of the misclassifications in ID3 are due to overfitting to the training set. This has happened in some “walking” examples which have been misclassified as “idle”. In C4.5 the tree is simpler and more robust, but that comes at a cost of accuracy in some cases, such as when some “resting” examples are confused with “idle” and vice-versa. The easiest activity to recognize in both decision trees was “running”.

Obtained →	Walking		Running		Idle		Resting		Accuracy	
	ID3	C4.5	ID3	C4.5	ID3	C4.5	ID3	C4.5	ID3	C4.5
Walking	735	789	11	11	51	0	3	0	91.88%	98.63%
Running	3	3	797	797	0	0	0	0	99.63%	99.63%
Idle	14	14	0	0	764	738	2	28	97.95%	94.62%
Resting	2	2	0	0	37	75	761	723	95.13%	90.38%

Table 3: Accuracy in recognizing the user context according to the decision trees induced by ID3 and C4.5.

6.3. Context Publication

Contexts can be published, upon user permission, to external servers. This has several advantages. First, it is possible to enable, disable or change the behavior of value-added services that depend on user context. Second, contexts can be augmented with information available at the network level, such as traffic conditions or specific services available at the user location. Third, the publication and availability of user context at the network level opens up the way to other services and applications, such as social networking, remote monitoring, health

assistance, etc. Context publication also provides the network operator with the ability to gather aggregated data on multiple users to study different user profiles.

Figure 8 illustrates the kind of results that can be obtained by analyzing such data. In this example the results were obtained by simulating the behavior of four different users, where context information was published to a server only when there was a change to a different context. In this scenario, several sequences of context changes were recorded for each user, during the time periods when the user was wearing the vest and had the smartphone turned on. The sequences of context changes for all users were then clustered together, without user identification, in order to check that it would be possible to rediscover the different user profiles. The dataset was stored in a Microsoft SQL Server database and then analyzed with SQL Server Analysis Services, in particular the sequence clustering algorithm which has been used in the past by our group to conduct other case studies as well [24].

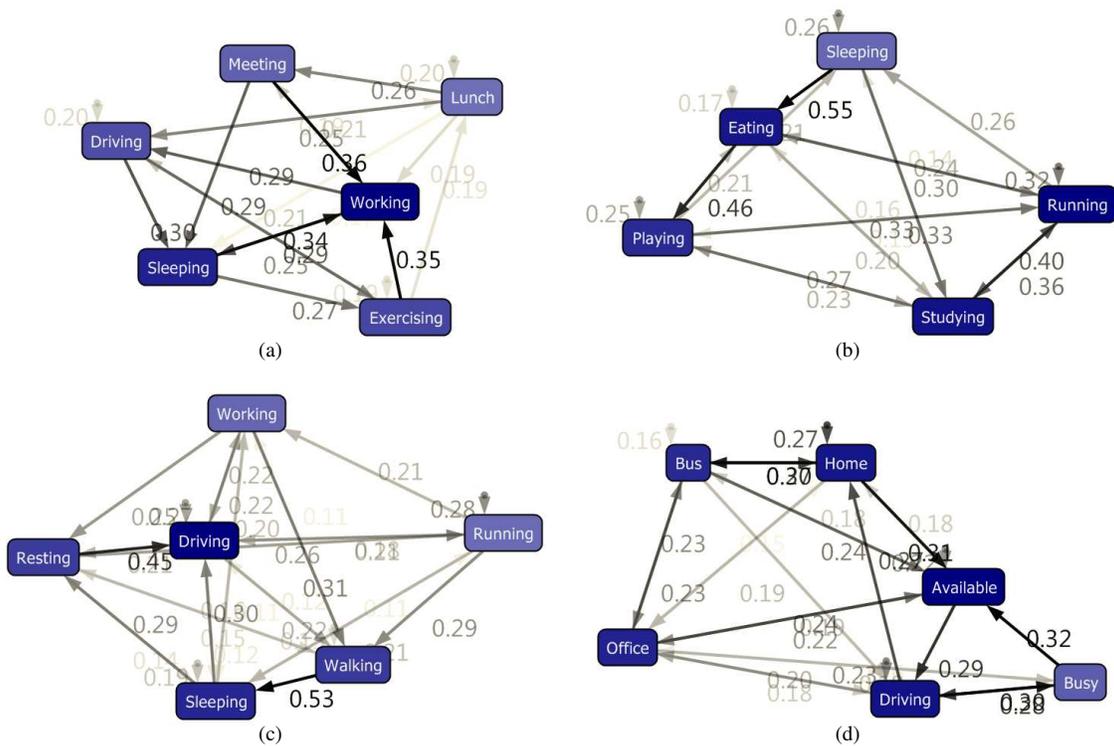


Figure 8: Rediscovery of user profiles from the sequence clustering analysis of context changes in a study with 4 users.

Figure 8 shows the most meaningful clusters obtained when processing the data in this study. Each cluster model is represented by a Markov chain where only the strongest transition probabilities are displayed. These selected clusters effectively have a direct relationship to the original behavior of the four users. For example, the cluster in Figure 8(b) represents the behavior of a child, with a 55% probability of going from context “sleeping” to “eating” and a 46% probability of going from “eating” to “playing”, with other possible transitions

in between. The clusters in Figures 8(a) and 8(c) both represent the behavior of working citizens, with different habits. Figure 8(d) shows the behavior of a user who has defined the set of contexts mostly based on location and availability status.

It should be noted that users are free to define their own context names and therefore, in general, there will be a small but varying degree of overlap between the contexts defined by different users. In a practical scenario, it will be important to first pre-process the dataset in order to retain only the most recurrent context names, whether in overlap or not, and to discard very specific or rare contexts that can only make it more difficult to get a general picture of the different user profiles. These and other preprocessing steps, such as those described in [25], will be required to improve the results that can be obtained from analyzing data from multiple users.

7. System Performance

With the goal of evaluating the relative cost and performance of the various stages of our prototype system, as well as to provide a benchmark for other similar systems, we have conducted a study on system performance, scalability and energy consumption. Figure 9 shows the execution times for the main processing stages. The times recorded are for non-optimized code executed on a Nokia N95 smartphone. Each stage can be described as follows:

- *Context Inference* – Consists in determining a context by searching through the decision tree based on the features obtained from sensor readings.
- *Feature Extraction* – Consists in assigning a feature (category) for each sensor value (or set of values) that have been acquired, according to what has been shown earlier in Table 2. This stage also includes all preprocessing steps such as buffering, averaging and computing signal characteristics.
- *Sensing (Reading)* – This is the process of acquiring data from each sensor. For external sensors, the application queries the sensor node via bluetooth, which then collects readings from all sensors.
- *Decision Tree Induction* – Builds the decision tree from the complete set of available training examples gathered from the learning periods when user contexts are recorded.
- *Persistence Storage (Writing)* – Writes training examples to persistent storage using record stores under control of the Record Management System (RMS) available in the smartphone.
- *Persistence Storage (Reading)* – Reads training examples to persistent storage using record stores under control of the RMS available in the smartphone.
- *Publication (Bluetooth)* – Publishes context information to a server machine via bluetooth communication. In this type of publication, the last feature readings from each sensor are also included.

- *Publication (HTTP + Wi-Fi)* – Publication of context information to a server via HTTP using a Wi-Fi connection. In this case, only the current context is published.
- *Publication (HTTP + 3G)* – Publication of context information to a server via HTTP using a 3G connection. Only the current context is published.

Results show that the main system loop comprising sensing, preprocessing, feature extraction and context inference takes around 0.415 seconds which allows the context to be updated at a maximum frequency of 2.4 Hz, if the decision tree is kept unchanged. This is rather independent of the actual size of the decision tree, as the time it takes to determine the context by traversing the decision tree is negligible. On the other hand, the publication of context information is a slow process (194ms for publication via bluetooth, 1s for publication via HTTP and Wi-Fi, and about 1.2s for publication via HTTP and 3G). These execution times support the initial decision of implementing all processing stages in the smartphone itself.

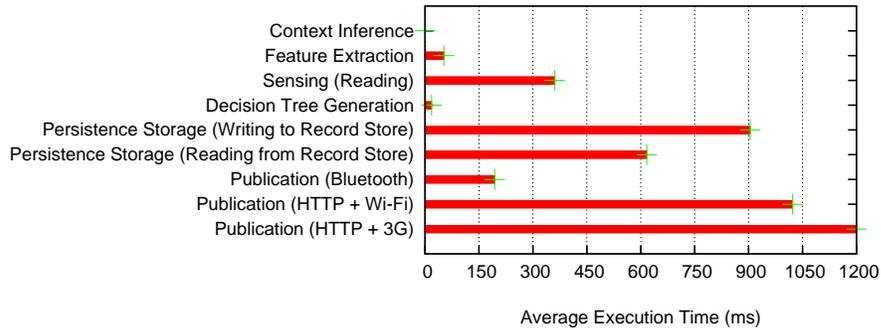


Figure 9: Execution time for different stages of the application.

Table 4 shows the execution time for some of the preprocessing operations. These execution times were obtained using a Sony Ericsson W910i smartphone. As can be seen in Table 4, the FFT implementation is more than 5 times faster than the use of a non-optimized DFT implementation. For the FFT implementation we evaluated two versions: one using float data types and the other using double data types. The performance of the two versions was similar. As expected, the preprocessing operations such as mean and variance are very fast over a window of just 16 samples.

To compare between different smartphones and also to study system performance with increasing number of sensor nodes, we have used up to three additional sensors connected via bluetooth. These happened to be GPS sensors because they were the type of sensor that we had available in the laboratory. The goal was to use these additional sensors to assess the behavior and evaluate the execution time needed for the sensing and feature-extraction stages with increasing number of external nodes. Figure 10 presents the execution times for these stages considering the use of the additional sensors. As expected, the addition of new sensor nodes increases the execution time for both the sensing and feature extraction stages, but this increase is not dramatic and in

Preprocessing operation	Execution time (ms)
Mean-16	1
Variance-16	1
DFT-32	165
FFT-Float-32	30
FFT-Double-32	32

Table 4: Execution time for some pre-processing functions.

general it is perfectly tolerable, at least for the applications we describe in this paper.

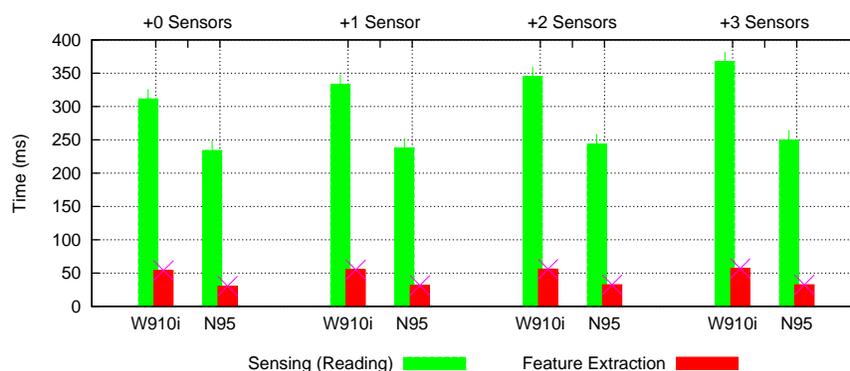


Figure 10: Execution times for the sensing and feature extraction stages using a Sony Ericsson W910i and a Nokia N95, with a varying number of additional sensors externally connected to the system via bluetooth.

An additional experiment tested the battery lifetime during continuous execution of the system main loop with the context being updated every second and with sensor data acquisition at intervals of 500 ms. Context publication was not considered. Taking as reference a baseline battery level, we conclude that the complete system can operate safely for at least 15 hours without the need for any battery recharge, and is limited by the shorter lifetime of some external nodes battery (such as the external GPS sensor node). The BlueSentry node with five sensors operated without battery recharge for 48 hours and the smartphone for 20 hours.

8. Application to Social Networking

Social networking has become increasingly popular with the rise of web-based communities in which users interact with each other on a regular basis. In [19] a descriptive definition for social networking sites is provided as a virtual community in which users have a personal web page (called a profile or space) that allows other users to know more about the interests and activities of the profile owner and to interact with each other in many ways, mostly using public channels.

Social networking websites such as *Facebook*¹⁶, *Hi5*¹⁷, *LinkedIn*¹⁸, *Twitter*¹⁹ and instant messengers like *Skype*²⁰, *GTalk*²¹ and *SAPO Messenger*²² nowadays bring together large numbers of users and are therefore responsible of a significant amount of Internet traffic.

Fortunately, most of these mainstream social networks provide public available APIs with the means to access, configure and update the user profile, status, etc. This is a key enabling feature for other systems which can help automate these updates and relieve the user from having to do them constantly.

Context information can play an important role in this scenario. Being able to retrieve and publish context information automatically to social networking sites will allow users to update their status effortlessly and will promote even more interaction between peers. Context information can be used, for example, to:

- Cope with user mobility.
- Update the current user status message with the current context.
- Enable actions associated with the current context (online/offline mode, available/busy/away status).
- Tag content with the current context.

In the following sections we describe applications where our context-aware system provides information on the user activity and current situation for usage in social networking. We show how the system publishes the user context to three different social services: first for *Twitter*, a micro-blogging social network that allows users to post short messages, then for *Hi5* where users can create profiles and share content, and finally for the instant messaging client *SAPO Messenger*, where user profiles hold a status message.

8.1. *Twitter and Hi5*

Twitter is today a well known social networking and micro-blogging service on which users can interact using short messages known as *tweets*. These tweets are simple text-based messages, that cannot be longer than 140 characters, which aim at answering the question “*What am I doing?*”. We use the inferred context information to publish the current activity and situation to a *Twitter* account. Periodically, the context is submitted via HTTP through a Wi-Fi or GPRS/3G connection (see Table 5) to the *Twitter* servers. The publication is accomplished using the REST API²³ provided by *Twitter* itself. Figure 11(a) presents a screenshot from the *Twitter* website, showing contexts published automatically by the system.

¹⁶<http://www.facebook.com/>

¹⁷<http://www.hi5.com/>

¹⁸<http://www.linkedin.com/>

¹⁹<http://www.twitter.com/>

²⁰<http://www.skype.com/>

²¹<http://www.google.com/talk/>

²²<http://messenger.sapo.pt/>

²³<http://apiwiki.twitter.com/>



(a) Twitter website.

(b) Hi5 Personal user profile.

(c) Hi5 status messages of user and friends.

Figure 11: User context publication to *Twitter* and *Hi5*.

We have also experimented with *Hi5*, one of the most successful web-based social networks with an estimate total of around 60 million users. The service allows users to create an online profile and present to friends or other viewers information such as interests, age and location, along with personal photo albums and music preferences. Similar to *Twitter*, the *Hi5* service allows the user to post a message to the network with information about the current activity the user is doing. Our system automates this status updates by publishing the user context periodically, so that the user profile is always up-to-date. Context information is published over a Wi-Fi or GPRS/3G connection, whichever is available at the moment, using the official *Hi5* REST API²⁴. Table 5 presents the time it takes to publish to the *Hi5* servers. Figure 11(b) and Figure 11(c) presents two screenshots from a profile page, where user accounts are being updated with context information published automatically by the system.

8.2. SAPO Messenger

SAPO Messenger is an instant messaging (IM) client where different users can chat through text messages in real-time. This IM client allows each user to assign a status message that is displayed to all the contacts. In a similar way to the previous scenarios, here the system publishes context information in order to update the user status message in *SAPO Messenger*. Screenshots of the application are shown in Figure 12 with status messages updated using the information sent by the system. Publication to *SAPO Messenger* is accomplished via HTTP through Wi-Fi or GPRS/3G (see Table 5) and using the Extensible Messaging and Presence Protocol (XMPP)²⁵. This is the standard protocol used by *SAPO Messenger* to update user status in their presence servers.

²⁴<http://api.hi5.com/>

²⁵<http://xmpp.org/>



Figure 12: User context publication to *SAPO Messenger*.

Figure 13 shows an example of an XML message that was created by the system to deliver the information to the context server over HTTP and according to a REST API. The message includes the context and its confidence level as well as information about the features for each sensor connected to the smartphone.

```
<?xml version="1.0" encoding="UTF-8" ?>
<p:contextmessage entity="upcase@sapo.pt"
  startedAt="2001-12-31T12:00:00"
  xmlns:p="pt.ptinovacao.contextinformation"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <p:contexts>
    <p:context>
      <p:resting />
      <p:confidence>100.0</p:confidence>
    </p:context>
  </p:contexts>
  <p:devices>
    <p:device id="Light Sensor">
      <p:value>0</p:value>
      <p:type>Light</p:type>
      <p:feature>Very Dark</p:feature>
    </p:device>
  </p:devices>
</p:contextmessage>
```

Figure 13: Example XML message for context publication on the context server.

8.3. Publication Time

To understand the impact of the communication overhead needed for publication of the user context information on the above social networking services, Table 5 presents the publication times for both 3G and Wi-Fi connections. Using this metric it is possible to get an estimate of the cost of communicating with external

sources (the social networking servers). For each time the system needs to publish the context we can estimate that it will take up to 2s to accomplish such task, regardless of the social network service being used.

Communication Technology	<i>3G</i>	<i>Wi-Fi</i>
<i>Twitter publication time (ms)</i>	1204	1098
<i>Hi5 publication time (ms)</i>	1171	1063
<i>SAPO Messenger publication time (ms)</i>	2081	1778

Table 5: Average publication times via 3G or Wi-Fi connections using the Nokia N95 smartphone.

9. Conclusion

Being able to gather information about user context is a key enabling feature for a new generation of context-aware services and applications. In the approach described here, we addressed the problem of distinguishing between a number of daily activities by means of a prototype system based on a set of general-purpose, inexpensive sensors connected to a regular smartphone via a bluetooth-enabled sensor module. Also, we described a layered architecture for the design and development of such context inference systems. The system architecture is based on a MIDP (Mobile Information Device Profile) application running on top of Java ME.

Experimenting with several techniques for the acquisition of raw sensor signals, for the classification of signal features, and finally for the identification of user contexts led to an understanding of both the complexity associated with these processes and the problems associated with having such system running on a mobile platform with limited resources. The implementation of the prototype system using a backpack, a vest, various sensors and a smartphone is fully operational and demonstrates the technical validity of the proposed approach. The prototype provides a proof of concept and operates in an unobtrusive manner after an initial training period.

The possibility of publishing the user context to an external server enables a wide range of context-aware services and applications. Examples include the publication of contexts to social networking websites such as *Twitter* and *Hi5* as well as to instant messaging programs such as *SAPO Messenger*. These examples show the opportunity of using context-aware systems to provide new user experiences. Ongoing work is focusing on different context inference approaches and on extending the experimental setup with additional sensors in order to accurately identify a vast number of daily-life activities.

Acknowledgments

This work was partially funded by PT Inovação. The views and findings reported in this article are those of the authors and do not necessarily reflect the views of PT Inovação. We would like to thank João Peixoto from PT Inovação for developing a REST API to allow context publication to the *SAPO Messenger* servers. We would like to acknowledge also the work of André Portela [23] in acquiring data and testing system accuracy.

References

- [1] Y. Kawahara, H. Kurasawa, H. Morikawa, Recognizing user context using mobile handsets with acceleration sensors, in: (IEEE) Intl. Conf. on Portable Information Devices (PORTABLE'07), 2007, pp. 1–5.
- [2] E. Welbourne, J. Lester, A. LaMarca, G. Borriello, Mobile context inference using low-cost sensors, in: Location- and Context-Awareness, Vol. 3479 of Lecture Notes on Computer Science (LNCS), Springer-Verlag, 2007, pp. 254–263.
- [3] A. C. Santos, L. Tarrataca, J. ao M. P. Cardoso, D. R. Ferreira, P. C. Diniz, P. Chaínho, Context inference for mobile applications in the UPCASE project, in: Mobile Wireless Middleware, Operating Systems, and Applications, Vol. 7 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer, 2009, pp. 352–365.
- [4] J. Coutaz, J. L. Crowley, S. Dobson, D. Garlan, Context is Key, *Communications of the ACM* 48 (3) (2005) 49–53.
- [5] R. Hull, P. Neaves, J. Bedford-Roberts, Towards situated computing, in: Proc. of the Intl. Conf. on Wearable Computers (ISWC'97), 1997, pp. 146–153.
- [6] M. Presser, A. Gluhak, D. Babb, L. Herault, R. Tafazolli, eSENSE - capturing ambient intelligence for mobile communications through wireless sensor networks, in: Proc. of the 13th Intl. Conf. on Telecommunications, 2006, pp. 27–32.
- [7] K. Cheverst, N. Davies, K. Mitchell, A. Friday, Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project, in: Proc. of the Sixth Annual Intl. Conf. on Mobile Computing and Networking, ACM Press, 2000, pp. 20–31.
- [8] G. Abowd, C. Atkeson, J. Hong, S. Long, R. Kooper, M. Pinkerton, Cyberguide: A Mobile Context-Aware Tour Guide, in: Proc. of the Intl. Conf. on Mobile Computing and Networking (MobiCom'96), 1996, pp. 421–433.
- [9] K. V. Laerhoven, Combining the kohonen self-organizing map and k-means for on-line classification of sensor data, in: Artificial Neural Networks (ICANN 2001), 2001, pp. 464–470.
- [10] K. V. Laerhoven, O. Cakmakci, What shall we teach our pants?, in: ISWC '00: Proceedings of the 4th IEEE International Symposium on Wearable Computers, IEEE Computer Society, Washington, DC, USA, 2000, p. 77.
- [11] C. Randall, H. Muller, Context awareness by analyzing accelerometer data, in: Proc. 4th Intl Symp. on Wearable Computers (ISWC'00), 2000, pp. 175–176.
- [12] S. Skaff, H. Choset, A. Rizzi, Context identification for efficient multiple-model state estimation, in: Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), 2007, pp. 2435–2440.
- [13] A. Krause, A. Smailagic, D. Siewiorek, Context-aware mobile computing: Learning context-dependent personal preferences from a wearable sensor array, *IEEE Trans. on Mobile Computing* 5 (2).
- [14] J. Healey, B. Logan, Wearable wellness monitoring using ecg and accelerometer data, in: Proc. of the Ninth IEEE Intl. Symp. on Wearable Computers (ISWC'05), IEEE Computer Society Press, Washington, DC, 2005, pp. 220–221.

- [15] H. Si, Y. Kawahara, H. M. H. Kurasawa, T. Aoyama, A context-aware collaborative filtering algorithm for real world oriented content delivery service, in: Proc. of ubiPCMM, 2005.
- [16] J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmäki, H. Toivonen, Time series segmentation for context recognition in mobile devices, in: Proc. of the 2001 IEEE Intl. Conf. on Data Mining (CDM'01), IEEE Computer Society Press, Washington, DC, 2001, pp. 203–210.
- [17] D. Siewiorek, A. Smailagic, J. Furukawa, A. Krause, N. Moraveji, K. Reiger, J. Shaffer, F. Wong, Sensay: A context-aware mobile phone, in: ISWC '03: Proceedings of the 7th IEEE International Symposium on Wearable Computers, IEEE Computer Society, Washington, DC, USA, 2003, p. 248.
- [18] D.-U. Jeong, S.-J. Kim, W.-Y. Chung, Classification of posture and movement using a 3-axis accelerometer, International Conference on Convergence Information Technology (2007) 837–844.
- [19] A. Joly, P. Maret, J. Daigremont, Context-awareness, the missing block of social networking, International Journal of Computer Science and Applications 4 (2).
- [20] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, A. T. Campbell, Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application, in: Proceedings of the 6th ACM conference on Embedded Network Sensor Systems, ACM, 2008, pp. 337–350.
- [21] J. R. Quinlan, Induction of Decision Trees, Machine Learning 1 (1) (1986) 81–106.
- [22] J. R. Quinlan, C4.5: programs for machine learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
- [23] A. Portela, Inferência de contexto para aplicações móveis (in Portuguese), Master's thesis, IST – Technical University of Lisbon (2009).
- [24] D. R. Ferreira, M. Zacarias, M. Malheiros, P. Ferreira, Approaching process mining with sequence clustering: Experiments and findings, in: Proceedings of the 5th International Conference on Business Process Management, Vol. 4714 of Lecture Notes in Computer Science, Springer, 2007, pp. 360–374.
- [25] D. R. Ferreira, Applied sequence clustering techniques for process mining, in: J. Cardoso, W. van der Aalst (Eds.), Handbook of Research on Business Process Modeling, Information Science Reference, IGI Global, 2009, pp. 492–513.