International Journal of Multimedia and Ubiquitous Engineering
Vol. 4, No. 4, October, 2009

Possibilities and Limitations of Context Extraction in Mobile Devices: Experiments with a Multi-sensory Personal Device

Tetsuo Yamabe and Tatsuo Nakajima

Dept. of Computer Science, Waseda University
{yamabe, tatsuo}@dcl.info.waseda.ac.jp

Abstract

This paper describes a context extraction framework for a mobile device that equips a variety of sensors. The framework captures context about a user and her surrounding environment; the information is used to adapt the behavior of applications running on the mobile device. Our framework adopts the blackboard architecture to execute multiple analysis modules that analyze signals from respective sensors. Respective modules implement different algorithms to complement each other's results to retrieve accurate high-level context. Based on experiments with a sample application, we evaluate the feasibility of the framework and point out the possibilities and limitations of context extraction in mobile devices.

Keywords: *context awareness, mobile computing, software framework, sensors*

1. Introduction

Within the past decade, mobile devices (e.g., PDA, mobile phone) have evolved with significant improvements in system performance, battery life, user interface design and wireless communication capability. A variety of feature-rich devices (e.g., smart phone) have been released into market and people can enjoy attractive mobile services (e.g., internet browsing, route navigation) anywhere they want. Mobile devices support our social activities today and the mobile computing technology has already been pervasive in our daily lives.

One important technology trend in the mobile computing evolution is sensors. Various sensors such as GPS receivers, accelerometers and capacitive touch sliders are incorporated to a mobile device, in order to enhance its user interface and functionality. Sensors have been originally used for monitoring the internal states of device (e.g., temperature, battery condition). In these recent years, however, the progress of miniaturization and low cost production of sensors have diversified the purpose; built-in sensors also cover user-oriented usages and applications. In other words, mobile devices have become able to perceive the external world, and they have been reaching the vision of context-awareness [11].

Since mobile devices are very personal and close to users, they are expected to play an important role in ubiquitous computing environments [4,6]. Services can recognize context of a user and her surrounding environments with analyzing collected sensor data [16]. However, to date, sensors have been mainly used for detecting primitive context that relates to the mobile device itself (e.g., posture, motion). To implement richer context awareness, useful sensors have to be identified in empirical studies. Also it is important to discuss effective sensor deployment strategy upon the experiments.

In this paper, we introduce a multi-sensory mobile device named Muffin. Muffin equips fifteen kinds of sensors to sense several types of contextual quantity such as acceleration, orientation, air temperature, a user's heart rate and so on. Muffin is a very unique device in the world, since it has been developed under a grand concept - “implement as many different kinds of sensors as possible into a PDA sized box”. Thus Muffin is a good prototype to perform empirical studies that aims to investigate sensors' characteristics and possibilities in context extraction process.

Empirical studies with Muffin showed practical issues in the mobile context extraction; the validity of sensor data and its analysis algorithms is not stable due to mobility. For example, biological sensor data are available under some limited conditions (e.g., “at a user's hand”). The mobility diversifies use cases of mobile device and we have to select an available sensor set according to the situation. On the other hand, advantages of the multi-sensory device were also identified; context can be detected in multiple ways with changing combination of sensors and analysis algorithms.

We have developed a software framework named Citron to utilize the advantages of multi-sensory mobile device. Citron supports parallel context analysis by employing the blackboard architecture [15]. Software APIs are offered to develop context analysis modules so that developers can easily monitor interesting context. Also we have developed a sample application on top of Citron and evaluated its feasibility. Based on Muffin and Citron development experiments, we discuss possibilities and limitations of context extraction in mobile devices.

In Section 2, we identify characteristics and requirements for realizing context-awareness on mobile devices. In Section 3, we introduce Muffin and point out some difficulties in mobile context extraction. In Section 4, Citron framework is introduced and a sample application is shown in Section 5. In Section 6, feasibility of our approach is evaluated and we conclude this paper in the final section.

2. Context extraction with mobile devices

Mobile devices play a particularly important role in ubiquitous computing environments [13]. Like a partner, people carry them most of the day and use them very frequently in daily scenes. From the interaction point of view, they are medium between a user and context-aware services. Through the device, the user can access several services running in the background. Also the services can display information with or without interactive actuation (e.g., vibration, make sound and light) through the device. Considering from a context extraction aspect, mobile devices are expected to act as a monitor of a user. With logging sensor data and/or interaction history, context of the user can be inferred. It contributes to make the services context-awareness. Thus context extraction with mobile devices is one of key research theme in ubiquitous computing research.

At the early stage of context extraction from mobile devices, location and time information are the main resource of context [1]. To identify the location, GPS receiver is used in outdoor environments and other location systems are used to support indoor positioning [10]. Device status (e.g., network connection) and static personal information (e.g., name) are also processed as additional context resource. Since sensitivity is limited, however, additional sensors are required to extract more complex context.

Furthermore mobile devices' mobility causes two critical issues in the context extraction process. One is physically limited space where sensors can be incorporated. Even though detectable context heavily depends on equipped sensors [2], available sensors on a mobile device are limited due to its portable small package. The other is the dynamics of mobile environments. Since mobile context (e.g., a user's activity, location) changes continuously, it is difficult to track the transition with poor processing power. Thus some limited sensors have been attached for limited context extraction in traditional researches.

The most frequently used sensor on mobile context extraction is accelerometers. An accelerometer is cost efficient and sufficiently small to be incorporated into mobile devices. Furthermore acceleration data is effective to recognize a user's activities (e.g., walking, running, walking up/down stairs) and motion (e.g., shaking, rotating, knocking on) [8,12]. Microphones are second-popular sensor after accelerometers, since ambient noise is useful to infer the place where a user exists (e.g., meeting room, restaurant, on the street) [6,9]. Also microphones are used to recognize speaking or talking [12]. Lastly environmental sensors are commonly used on a mobile device; light and temperature sensors are used to extract environmental and a user's context [5].

Due to the issues, however, it has not been sufficiently discussed what types of sensors are useful for mobile context extraction. Also it has not been clarified what kind of context can be efficiently extracted from sensors on mobile devices. In the next section, we introduce Muffin that is a unique mobile device in terms of the unparalleled sensitivity. Based on preliminary experiments on Muffin, we point out practical issues in the context extraction process, and identify requirements to utilize the advantages of multi-sensory mobile devices.

2.1. Related work

Hinckley et al. have added multiple sensors to a handheld PDA for enabling interactive user interface [7]. Developed device has an IR the proximity range sensor for measuring the proximity to a user, a touch sensor for detecting whether a user is holding it, and two axis accelerometers for detecting its tilt. Extracted context is the basic state of device and used only for user interface extension. A static set of context was analyzed, and context extraction over multiple sensor data was not discussed.

Siewiorek et al. have developed SenSay, which is a context-aware mobile phone for recognizing interruptible states [12]. Five kinds of sensors (i.e., voice microphone, ambient noise microphone, accelerometer, temperature sensor and visible light sensor) are mounted on a sensor box. SenSay extracts sensor data from the box and it decides the activity of a user (i.e., uninterruptible state, active state, idle state and normal state). It is similar to our work at the point they inferred a user's state from multiple extracted context. However flexible context extraction on different context was not discussed.

Gellersen et al. have proposed TEA (Technology Enabling Awareness) approach and they incorporated sensors into a mobile phone [6]. The TEA sensor board equips various kinds of sensors, such as a light sensor, microphone, CO-sensor, IR sensor, accelerometer and so on. They proposed preprocessing architecture called cue to reduce the analysis cost on context extraction. The output of each cue is classified into clusters by self-organizing map. Their approach is similar to ours, since abstract context is retrieved by stepwise approach and sensor data is processed in multiple ways. However non-exclusive context representation and parallel context analysis were not discussed.

Laerhoven et al. have shown a practical example of multiple interpretations of the same sensor data [8]. They created a wooden cube with a sensor module as a tangible input device and put an accelerometer into it. Then they extracted “gesture", “orientation" and “top side" from acceleration data with different analysis algorithms. Expanding analysis algorithms is more efficient for embedded devices rather than adding more sensors in terms of power consumption. Thus we take the approach to extract some of the basic states of Muffin.

3. Preliminary experiments with Muffin

3.1. Muffin: a multi-sensory personal device

Muffin is the prototype of mobile device for studying context awareness. It was developed in the collaboration work with Nokia Research Center. The significant characteristic of Muffin is its sensing capability: thirteen kinds of built-in sensors in the PDA sized box and two kinds of externally attached sensors are available (**Figure 1**).

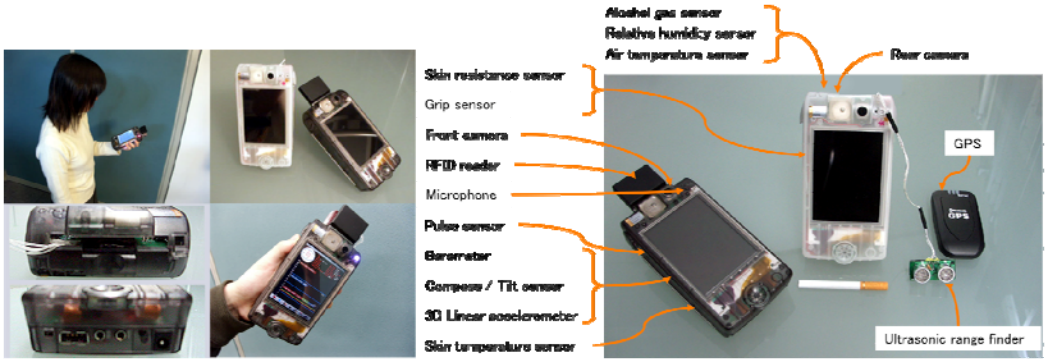

Figure 1. Muffin terminal and available sensors

The sensors can be roughly divided into four groups. First group is environmental sensors: an air temperature sensor, a relative humidity sensor and a barometer. Second group is physiological sensors: an alcohol sensor, a pulse sensor, a skin temperature sensor and a skin resistance sensor. Third group is motion/location sensors: a compass/tilt sensor, a 3D linear accelerometer, a grip sensor, an ultrasonic range finder and a GPS receiver. The ultrasonic range finder and the GPS receiver are externally attached as optional sensors. Last group is remaining sensors: an RFID reader, front/rear cameras and a microphone. Linux operating system runs on Muffin, so each sensor can be accessed as a device file (e.g., /dev/AccelX). Also Muffin equips ordinary user interfaces (e.g., touch screen, micro joy stick, microphone, vibration motor) and connection interfaces (e.g., IrDA, Bluetooth, wireless LAN, USB port).

3.2. Context-awareness on Muffin

Muffin has so many kinds of sensors that it can quantify several kinds of physical phenomena. **Table 1** shows examples of context that could be extracted from Muffin's sensors. In the table, context is divided into three categories based on its subject: Muffin terminal, a user, and environments.

Table 1. Examples of context that could be extracted from Muffin's sensors

Class	Description	Sensor
Muffin terminal's context		
Motion	Moving speed	3D accelerometer, ultrasonic range finder
	Trajectory	
Posture	Top side	3D accelerometer, digital compass
	Tilt	
	Orientation	
Placement	At a user's hand	Skin resistance sensor, grip sensor
User context		
Activity	Stationary state (e.g., standing, sitting)	3D accelerometer, ultrasonic range finder, digital compass
	Moving state (e.g., walking, running, going up/down stairs)	
Geographical information	Location	GPS receiver
	Orientation	Digital compass
Physical condition	Stress level	Skin resistance/temperature sensor, pulse sensor, grip sensor
	Alcoholic level	Alcoholic sensor
Emotion	Exciting	Skin resistance/temperature sensor, 3D accelerometer, grip sensor, pulse sensor
	Surprising	
	Fearing	
Environmental context		
Air condition	Air temperature	Air temperature sensor
	Air humidity	Relative humidity sensor
	Air pressure	Barometer
Sound	Ambient noise	Microphone
	Talking voice	

Based on this classification, we performed preliminary experiments in order to confirm feasible cases. As a result, we found important points and design issues in the context extraction with Muffin as follows.

Muffin: Muffin terminal's context could be extracted accurately, because used sensors output valid data. Also the sensors are so responsive that context can be analyzed in real time. Furthermore Muffin's state can be clearly classified into exclusive classes, so that simple algorithms such as threshold analysis could be applied. For example, Muffin takes either "at user's hand" state or "not at a user's hand" state at a certain moment, and it can be analyzed by simply comparing grip sensor data with threshold value.

User: There are three important issues in the process of user context extraction. The first issue is the availability of sensors and context analysis algorithm. User context frequently changes in mobile computing environments, so available sensors and algorithms also change accordingly. In order to extract user context, a user has to carry Muffin in some ways. However, available sensors and algorithms change according to the position or situation in which Muffin is used. For example useful analysis algorithm for detecting "standing or

sitting" with acceleration data changes according to Muffin's context: whether Muffin is held or waist-mounted. Also if it uses an ultra range finder to correct analysis results by measuring the distance from floor, the validity of the sensors also changes.

The second issue is the delay caused from time-consuming context extraction process. In the previous example, detecting "sitting" state is easy at five minutes after the user actually sits. To detect the event instantaneously, we need to analyze the wave pattern of sensor data just in a few hundred milliseconds. This issue should be discussed also about other sensing domains, such as emotion awareness. To retrieve valid physiological sensor data, Muffin has to be grabbed by a user's hand. However, short-term sensor data is not sufficient to recognize physiological context. For example sensor data collected from a skin temperature sensor changes very slowly, and sensor data from a pulse sensor changes too rapidly. In other words it is necessary to log the average of extracted sensor data and compare them in the enough span of time.

The last issue is the complexity and ambiguity of context. The definition of complex context, such as angriness or feeling of hunger, changes according to situation and application. For example the meaning of loud voice differs in different situations (e.g., "in the meeting room", "while talking with friends"). To appropriately recognize context, further information about a user (e.g., location, activity) is required in addition to sensor data (e.g., microphone).

Environment: Environmental context such as air temperature directly relates to raw sensor data, so it is not difficult to calculate them. However, Muffin gets hot internally as time goes on, and the heat affects environmental sensors. As a result of that, sometimes measurement does not work and collected sensor data become unuseful. This problem arises from mechanical design, so we should refine the placement of sensors in Muffin and protect them from the heat affect.

3.3. Design issues in context extraction process

We have pointed out practical difficulties in mobile context extraction. Even about simple states such as "standing" or "sitting", mobility affects the availability of sensors and analysis algorithms. In physical condition or emotion sensing cases, such difficulties become increasingly prominent, because the complexity of context increases. Therefore we should go back to consider simple context extraction cases, and then discuss about design issues for effective and robust context extraction with Muffin.

At first, complex context should be represented as a combination of other simple context. If the complex context can be decomposed into primitive one like Muffin's activity, it becomes easy to reconstruct them and represent higher abstract context. This approach is also important in terms of decreasing the ambiguity of context. Next, we should observe physical phenomena from multiple aspects of view. In order to increase the quality of information, additional sensor data or context are required. In addition, alternative context analysis algorithms should be prepared to ensure robust context extraction, because required sensors could suddenly become unuseful as described in Section 3.2.

This approach also contributes to avoid the time-consuming process issue. In some cases, optional sensors or alternative analysis algorithms are more effective to recognize context than a default approach. For example an ultrasonic range finder is useful to detect whether a user sit down, in the case that she is holding Muffin and the sensor can measure the distance

from floor. While one sensor data offers multiple meanings, one context can be recognized in multiple ways.

At last, it is required to specify relationship and dependency among decomposed context. Our experiments show dependency relations among context. For example, available analysis algorithms or sensors change according to the situation, because there are several styles to use Muffin. In **Figure 2** possible three cases of context dependency are identified. These relationships are classified based on a hierarchical context modeling. The resources are inputs (i.e., context or sensor data) to context analysis modules. The context analysis modules work for the extraction of predetermined context with analysis algorithms.

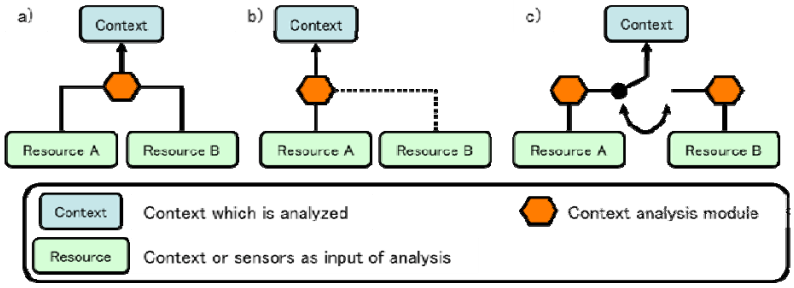

Figure 2. Relationship and dependencies among context

The case a) is a basic hierarchical context abstraction with the combination of Resource A and Resource B. The case b) and the case c) are its variations. In case b), context is mainly represented as a result of Resource A processing. Resource B is supplementary used to configure the analysis module's algorithm. In the case c), context is represented as a processing result of Resource A or Resource B. When the availability of one analysis module is not sufficient, other alternative module works instead of the original one.

As a result of the discussion, design issues in context extraction with Muffin have been pointed. They are not separated issues, but related to each other as shown in **Figure 3**. In the figure, there are four basic context that represent Muffin's context: "held or not", "top side", "moving or stop" and "activity". Every subject takes an exclusive states and the state of Muffin can be clearly classified into one of them.

However the relationship among context is non-exclusive, so more complex context can be represented with a context relationship as described above. For example, if the display of Muffin turns up ("top side") and a user holds it ("held or not"), the user might look into Muffin's display ("under observation or not"). This is one of the context relationship represented as the case a).

In order to extract a wide variety of context, one sensor should be analyzed from multiple aspects of view. In **Figure 3**, three different analysis modules analyze acceleration data, and it is processed into different context and meanings. Furthermore context is also observed from multiple viewpoints. In the figure, "Activity" is recognized from two different sensors: accelerometer and ultra range finder. One of them is selected according to the availability of sensors. This example shows the case c) that dynamically changes its analysis module and

required resources. Also we can find the case b): the analysis module of “walking or running or not” changes its threshold according to the posture of Muffin terminal.

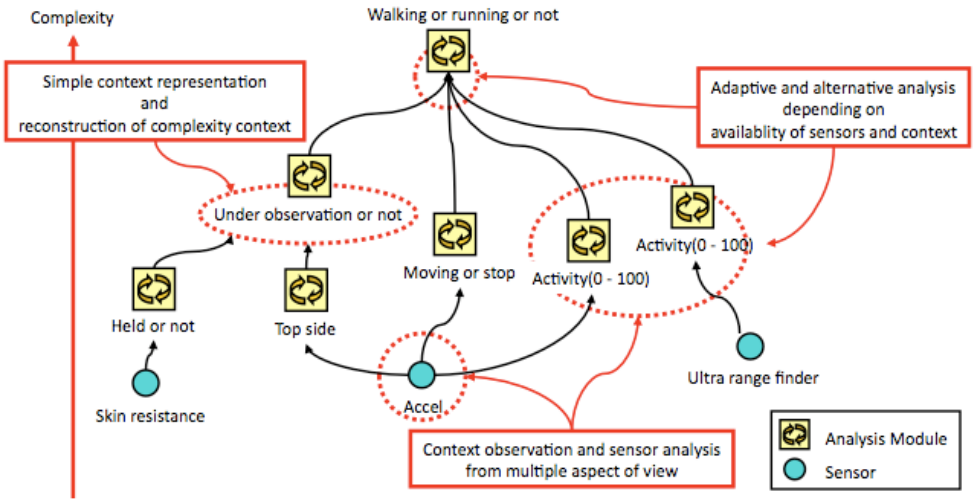

Figure 3. Context-processing diagram with the design issues

We have refined this context-processing diagram into a software framework named Citron, which offers a software API for context analysis module development. In Section 4, we introduce Citron architecture and its features.

4. Citron: context information acquisition framework for Muffin

4.1. System architecture

To implement the design issues discussed in Section 3.3, we implemented:

1. A context analysis module framework that allows developers to specify relationship among modules.
2. A shared space for module communication that stores extracted context.

In this implementation, we employed the blackboard architecture to coordinate context analysis modules. Furthermore we defined each context analysis module as a worker for exclusive context extraction.

The blackboard architecture is a data centric processing architecture that has originally been developed for speech understanding and artificial intelligence. There are one shared message board and multiple worker modules for collaborative data analysis. Each module reads information from the message board as a resource, and after processing the data it writes the result to the board. Thus module communication is established without the knowledge about other modules. Furthermore extracted data are analyzed and complimented in the communication process.

To implement the blackboard architecture, we adopt tuple space based programming model [3]. The tuple space is one of implementations for inter-process communication among

independent processes. The tuple space refines a shared space with a very flexible data type called tuple, and it enables applications to search tuples with a template-based query. The important characteristics of the tuple space based system are 1) loose coupling of worker modules, 2) information sharing among worker modules and 3) flexible data representation.

Especially in ubiquitous computing environments, devices and services should work without specific knowledge (i.e. IP, port) about others, since the environment dynamically changes. Moreover context for several applications could be represented in a unified format because of tuple's flexible data type. Winograd discussed the characteristics and trade off between the blackboard architecture and other frameworks on the purpose of multi-process coordination [15]. We have also employed the blackboard architecture for designing Citron, because it is an adequate architecture to implement the design issues. **Figure 4** shows the overview of Citron architecture.

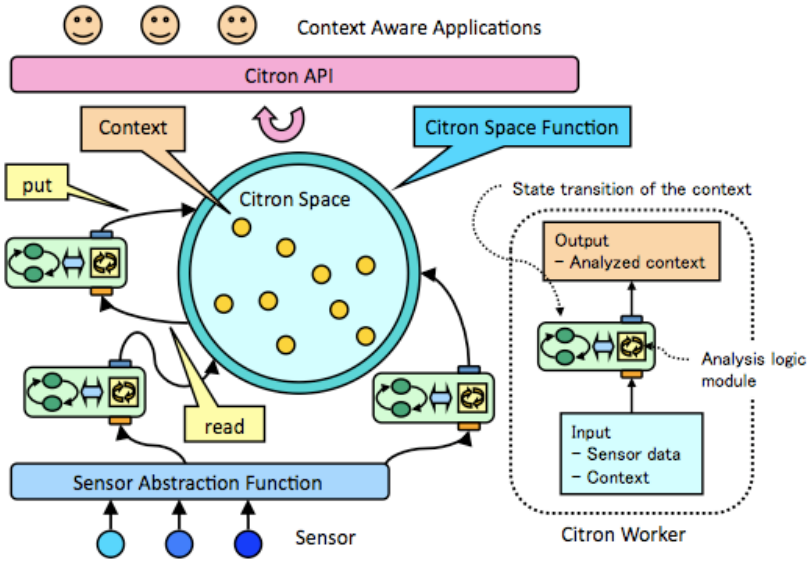

Figure 4. Citron architecture overview

Citron consists of two software components: Citron Worker and Citron Space. Citron Worker is a sensor data analysis module. Each worker collects sensor data from Muffin's sensors and it retrieves context from Citron Space. Also each worker takes responsibility on single context extraction. For example, a worker that recognizes "held or not" observes the value of a skin resistance sensor in order to detect "held" or "free" with threshold analysis.

Citron Space is a tuple space and it stores tuples that represents context analyzed by Citron Workers. Citron Space handles data management requests from both Citron Workers and the applications running on top of Citron. Context is represented as a set of meta-information, such as subject, state and time. Detailed explanation will be given in Section 4.2.

There are two internal functions (i.e., Sensor abstraction function and Citron Space function) in the system, and a software API is offered to develop external applications. The sensor abstraction function is just a simple wrapper function for accessing device files of

sensors, thus Citron Workers extract sensor data with this function. Citron Space functions allow accessing Citron Space with three types of methods as shown in

Table 2.

Table 2. Brief description of Citron Space functions

Method	Description
put	Insert context into Citron Space.
read	Read context from Citron Space with template matching.
get	Read and remove context from Citron Space with template matching.

Lastly Citron API is published to enable applications to retrieve context from Citron Space. More details are described in Section 4.3.2.

4.2. Context representation

In Citron architecture, context is represented as a set of meta-information:

Context := {ID, Subject, State, Time, Lag, Interval}

Also **Table 3** explains each meta-information that makes up tuples.

Table 3. Explanation of Tuple fields and corresponding meta-information

Field	Description
ID	Citron Worker's unique identifier
Subject	Subject of the context (e.g., "orientation")
State	Verb of the context (e.g., "NW")
Time	Time when the context is analyzed
Lag	Time lag in the analysis
Interval	Update interval of the context

For example context that is written as "a user is not walking (i.e., just standing)" is represented as follows:

{"ID_walk", "walking", "at_rest", 1107005245, 0, 100}

In this case, the state could be either "walking" or "at_rest". On the other hand the subject field is fixed, so Citron Worker extracts only one specified context. The lag and interval field are the meta-information of context, which are inherent in its analysis algorithm. Some analysis algorithms such as FFT analysis require a certain amount of data and time for buffering. Thus the lag field shows the time lag to applications so that it can be handled in appropriate manner. The interval field shows the freshness of context. Also the ID field specifies the identifier of Citron Worker that created the tuple. Tuples in Citron Space are updated every its polling interval by the Citron Worker that is associated with same ID. Thus applications can examine the freshness of context by looking up the interval field and the time field.

4.3. Implementation

Citron is written in C language, and it offers 1) a framework for Citron Worker development and 2) software API for application development.

4.3.1. Citron Worker framework: This framework offers the abstraction of context analysis module so that developers can easily implement Citron Workers. The framework provides common functions of context analysis modules, such as connection management to Citron Space and sensor data retrieval from Muffin. Thus developers can concentrate on implementing analysis algorithms. When Citron Worker is initialized, the specification of the analysis, such as the type of sensor and the subject of context, has to be declared. First, a developer has to set value to the tuple fields. Then Citron Worker starts to run and invokes an analysis function at every specified interval. This analysis function executes the analysis algorithm with retrieving sensor data and context. The analysis result is returned as context and the Citron Worker shares it in Citron Space. In the current implementation, Citron Worker only puts the result of analysis when the state of context is changed, in order to reduce the load of Citron Space.

4.3.2. Citron API: Applications running on top of Citron access to Citron Space with invoking below functions. Context_t is the data structure implementing the context representation shown in Section 4.2.

- char* libcitron_get_state(char* subject);
- context_t* libcitron_get_context(char* subject);
- int libcitron_add_event_handler(const char* subject, void (*handler)(char*));
- void libcitron_remove_event_handler(void);

Citron Space is implemented based on LinuxTuples, which is a tuple space implementation on Linux operating system [14]. Originally tuple space allows flexible wildcard-based query for tuple search, but in Citron it is wrapped as a simple function: only the subject field is used as a key for search. This design is sufficiently useful for many applications and it decreases the workload of Citron Space. Furthermore Citron API also provides callback function management interface. Developers can register/remove callback functions to handle state change events in Citron Space.

We have developed several Citron Workers and context-aware applications to evaluate the Citron framework. In the next section, we introduce one sample application and Citron Workers developed for the application.

5. Sample application

We developed a sample context-aware application named “RouteTracer”, which uses a user's context extracted by Muffin. This application displays the track of walking route in real time with a user's state. RouteTracer also shows walking speed and the duration that a user stayed at the same point. The walking speed is divided into five levels and it can be distinguished with different colors. The point where the user stopped is represented as a circle and its radius becomes larger as time advances. **Figure 5** shows a sample map image generated by RouteTracer (right) and the user's states that are used to draw the map (left).

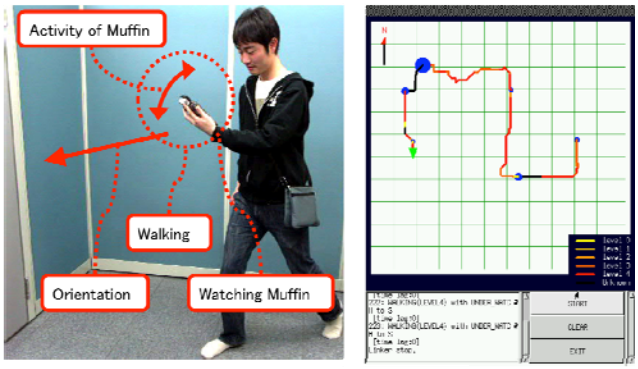

Figure 5. RouteTracer

In this application, three kinds of context are required: “walking state”, “walking direction” and “walking speed”. To draw the track of route, at least “walking direction” and “walking state” are necessary. The “walking speed” context is optional, however, it is necessary to speculate the distance for creating more accurate map. The “walking speed” context is inferred with the activity level of Muffin. In preliminary experiments we found that the activity correlates with the walking speed of a user, while she is walking with looking Muffin’s display. Thus “watching” context is required to determine walking speed. Six Citron Workers run to extract corresponding context in total: “orientation”, “walking”, “activity”, “watching”, “holding” and “top side”. **Figure 6** shows the relationship among context required by RouteTracer application.

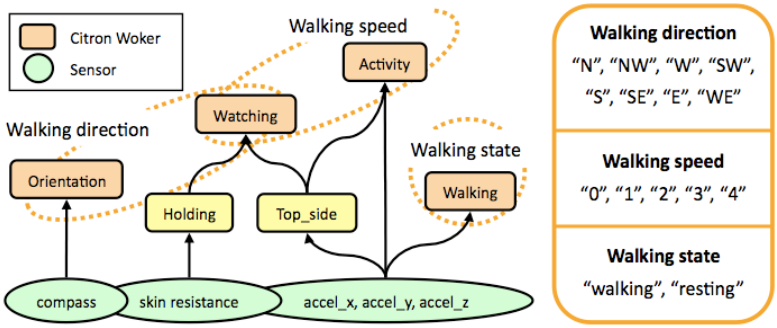

Figure 6. Required context and Citron Worker in RouteTracer

The “orientation” worker retrieves sensor data derived from a compass, and then it analyzes which orientation Muffin is heading. This orientation can be regarded as the direction of walking, when a user is watching Muffin. The “watching” worker recognizes whether the user is watching Muffin or not, based on a result of “holding” and “top side” worker analysis. If the user holds Muffin and the display of Muffin looks towards the user’s face, the “watching” worker recognizes the state as “the user is watching Muffin”. The “holding” worker analyzes sensor data derived from a skin resistance sensor, and the “top side” worker analyzes gravity acceleration. The “activity” worker also retrieves acceleration data and it recognizes the activity level of Muffin with FFT analysis. This worker requires context generated by the “top side” worker, since the axis for motion detection changes

according to the topside of Muffin. As described above, activity status is divided into five levels and treated as the walking speed. The "walking" worker decides whether the user is moving or resting with acceleration data: it is used as the "walking state" in RouteTracer.

It is also possible for RouteTracer to recognize the walking state only with "walking speed". Performance in the analysis is not good, however, because the "activity" worker has to take about 6.4 seconds, in order to collect 128 samples into buffer and analyze them at every 50 msec. On the other hand, the "walking" worker analyzes the user's walking status with simple zero cross detection in real time. Thus it is expected that the parallel analysis using these two Citron Workers enable RouteTracer to be more responsive to recognize the walking state. In the next section, we evaluate Citron with measuring the overhead in invoking API. Also we compare the accuracy of map drawn by RouteTracer with changing Citron Workers.

6. Evaluation

6.1. Performance

We have measured the overhead in accessing Citron Space with invoking Citron Space functions described in Section 4.1. The dependency relation between the execution overhead and the number of running Citron Workers is also evaluated. Each worker invokes a Citron Space function ten times, and the execution time is measured as the mean time of all workers that run in parallel. **Figure 7** shows the result of experiment.

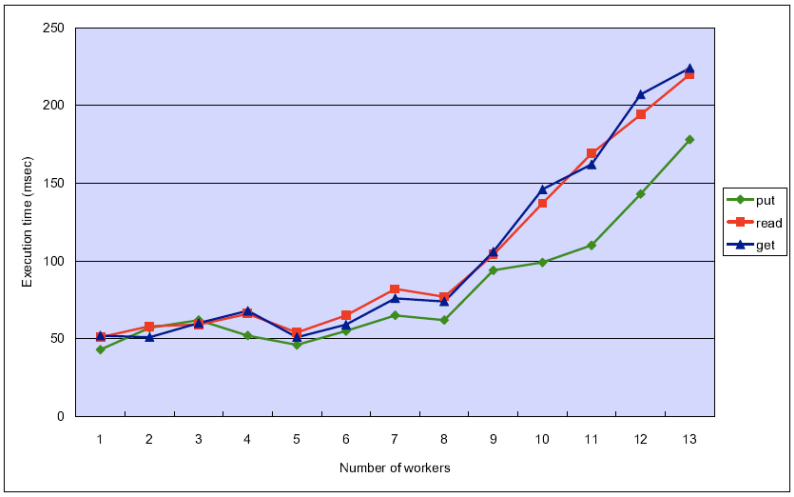

Figure 7. Relationship between execution time and the number of running

Figure 7 clarified that the overhead in Citron Space access increases as the number of Citron Worker increases. Especially the execution time remarkably increases when the number of Citron Worker goes over eight. Also due to template-matching search, "read" and "get" functions recorded longer execution time than "put" function.

6.2. Experiences with the sample application

In this section, we evaluate the effect of Citron Worker coordination using RouteTracer. We compared drawn maps in three cases: using only "walking state" (case 1), using only "walking speed" (case 2), and using both for hybrid analysis (case 3). **Figure 8** shows the walking route in the left side. A participant walks the route with holding Muffin in her hand. The participant was instructed not to pay attention to the display in order to remove the effects of intentional map creation. To clarify Citron Workers' effect, the participant intentionally changes walking speed. The maximum walking speed in this examination is about 5 km/h and normally it is 3 km/h. Also two stop points were instructed on the route, so the participant had to stop for 10 seconds at the points.

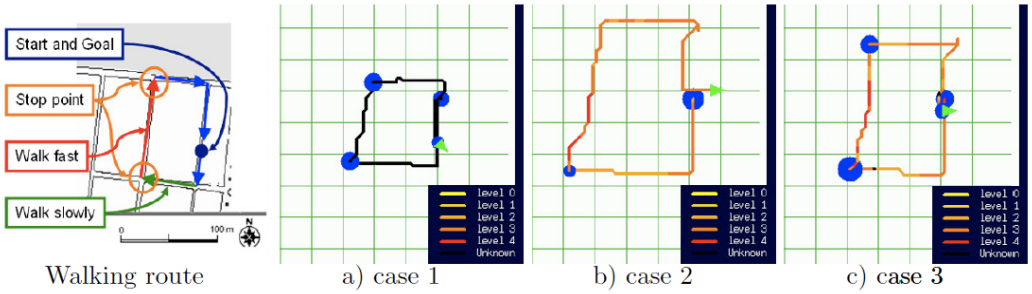

Figure 8. Walking route and drawn maps by RouteTracer in each experiments

Figure 8 also shows the result of each examination case. Figure a) is drawn with the time of walking and its orientation. Turning point and time that the participant has stopped can be clearly confirmed. This is because that the "walking" worker responds quickly when it recognizes the walking status change. However the walking speed is not reflected, so the length of each edge is inferred based on walking time. As contrasted with Figure a), Figure b) speculates the walking distance based on the walking speed. As a result, the drawn map becomes more similar to the actual map than Figure a). Figure b) also shows that the "activity" worker could not detect the stop points, because 10 seconds are not sufficient time for the FFT analysis to recognize the state change. Figure c) shows a more accurate map than Figure a) and Figure b). This case exploits the advantage of each analysis method and it also shows the effectiveness of hybrid context extraction with multiple types of analysis methods. The stop points were detected clearly, and the shape of map is the most accurate.

7. Conclusion

In this paper, we have addressed possibility and importance of mobile devices in ubiquitous computing environments. In order to clarify possibilities and limitations in context extraction with a mobile device, we have introduced Muffin that is a prototype of multi-sensory personal device. Based on preliminary experiments, we pointed out design issues and proposed a software framework named Citron. A sample application was developed and evaluated feasibility of our approach.

We found two further issues in the experiments. One is Citron's performance issue caused from the blackboard architecture. Parallel context analysis with multiple sensors heavily

burdens mobile devices. Thus we should optimize the performance and reduce the load with re-designing Citron. Moreover it is assumed that the limitation of workable Citron Worker heavily depends on the implementation of Citron Space. As described in Section 4.3.2, Citron Space is implemented as a wrapper function of LinuxTuples. The performance can be improved if the implementation is optimized for context processing.

The other one is limitation of context extraction on Muffin. Most of physiological sensors on Muffin require some constraints to be used, such as the position of a finger and the style of holding, to measure valid data. Thus accuracy of such sensors changes so frequently according to the situation. It follows that other sensor devices (e.g., wearable sensors) are required as the alternative resource of context analysis. Citron can coordinate such remote devices easily, since the blackboard architecture is suited to dynamically add/remove knowledge resources and corresponding analysis algorithms.

Acknowledgement

Most of this work was conducted in the collaboration project with Nokia Research Center Tokyo, and we thank them for great contributions on Muffin development and fruitful discussions.

References

[1.] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A mobile context-aware tour guide. In ACM Wireless Networks, pages 421–433, 1997.
[2.] Michael Beigl, Albert Krohn, Tobias Zimmer, and Christian Decker. Typical sensors needed in ubiquitous and pervasive computing. First International Workshop on Networked Sensing Systems (INSS) 2004, pages 153–158, Jun. 2004.
[3.] N Carriero and D Gelernter. Linda in context. Communications of the ACM, 32(4): 444–458, Apr. 1989.
[4.] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.
[5.] Patrick Fahy and Siobhan Clarke. Cass – middleware for mobile context-aware applications. In Second International Conference on Mobile Systems, Appli- cations, and Services (MobiSys2004), 2004.
[6.] Hans W. Gellersen, Albercht Schmidt, and Michael Beigl. Multi-sensor context-awareness in mobile devices and smart artifacts. Mob. Netw. Appl.,7(5):341–351, 2002.
[7.] Ken Hinckley, Jeffrey S. Pierce, Mike Sinclair, and Eric Horvitz. Sensing techniques for mobile interaction. In UIST, pages 91–100, 2000.
[8.] Kristof Van Laerhoven, Nicolas Villar, and Hans-Werner Gellersen. Multi-level sensory interpretation and adaptation in a mobile cube. In In Proc. of the third workshop on Artificial Intelligence in Mobile Systems (AIMS), Ubicomp 2003, pages 111–117, 2003.
[9.] Jani Mantyjarvi, Johan Himberg, Petri Kangas, Urpo Tuomela, and Pertti Huuskonen. Sensor signal data set for exploring context recognition of mobile devices. In Workshop: Benchmarks and a database for context recognition in conjuction with the 2nd Int. Conf. on Pervasive Computing (PERVASIVE 2004), pages 18–23, Apr. 2004.
[10.] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In Mobile Computing and Networking, pages 32–43, 2000.
[11.] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, US, 1994.
[12.] Daniel Siewiorek, Asim Smailagic, Junichi Furukawa, Neema Moraveji, Kathryn Reiger, and Jeremy Shaffer. Sensay: A context-aware mobile phone. In Proceedings of 2nd International Semantic Web Conference (ISWC2003), pages 248–249, 2003.
[13.] Mark Weiser. The computer for the twenty-first century. Scientific American, pages 94–104, Sep. 1991.

[14.] Ware Will. Linuxtuples. http://linuxtuples.sourceforge.net/.
[15.] Terry Winograd. Architectures for context. HCI Journal, 2001.
[16.] Tetsuo Yamabe, Kaori Fujinami, and Tatsuo Nakajima. Experiences with building sentient materials using various sensors. In In Proceedings of 24th International Conference on Distributed Computing Systems Workshops, 2004.

Authors

Tetsuo Yamabe is a Ph.D. candidate at Distributed and Ubiquitous Computing Laboratory in Waseda University. His research interests are mobile computing, pervasive services with context awareness and economic incentive design in persuasive technologies.

Tatsuo Nakajima is a professor of Department of Computer Science and Engineering in Waseda University. His research interests are distributed systems, ubiquitous computing, operating systems and interaction design. He is currently a leader of Distributed and Ubiquitous Computing Laboratory. (http://www.dcl.info.waseda.ac.jp).