Liquid: Designing a Universal Sensor Interface for Ubiquitous Computing

TECHNICAL REPORT

Submitted to the Computing Department
Lancaster University
December 11, 2003

[†]Kiel Mark Gilleade, [†]Jennifer G. Sheridan, [◊]Jen Allanson

[†]Computing Department, Lancaster University, Lancaster LA1 4YR
`{gilleade, sheridan}@comp.lancs.ac.uk`

[◊]School of Computing and Mathematical Sciences, Liverpool John Moores University, Liverpool L3 3AF
`allanson@livjm.ac.uk`

Liquid: Designing a Universal Sensor Interface for Ubiquitous Computing

[†]Kiel Mark Gilleade, [†]Jennifer G. Sheridan, [◊]Jen Allanson

[†]Computing Department, Lancaster University, Lancaster LA1 4YR
`{gilleade, sheridan}@comp.lancs.ac.uk`

[◊]School of Computing and Mathematical Sciences, Liverpool John Moores University, Liverpool L3 3AF
`allanson@livjm.ac.uk`

Abstract. This paper describes the specification of a universal sensor interface (USI) called Liquid, which allows for the collection and representation of sensor readings from a wide range of different sensors. We illustrate how it is possible for Liquid to collect data from a broad spectrum of sensors using a select method of sensor classification and to present this data within a common environment. We explore how this approach can extend itself to include sensors not yet conceived of with relative ease. Finally we explain how the Liquid USI provides developers of ubiquitous systems with a general-purpose toolkit for the development of sensor-based applications.

1 Introduction

Miniaturization and a dramatic decrease in the cost of sensors are beginning to turn Mark Weiser's vision of ubiquitous computing [1] into reality. A growing interest in ubiquitous computing [2] coupled with a rise in the demand for ready-to-use sensors [3] suggest that sensors will play an increasingly important role in ubiquitous computing and other sensor-based domains.

The sensor market is brimming with an assortment of ready-to-use sensors and data acquisition devices. The graphical user interfaces (GUI) [4] used to manipulate them are just as numerous. Sensor GUIs tend to be hardware-specific and as a result, when users purchase a new type of sensor they need to familiarise themselves with a different operating environment before they can concentrate on the more important job of capturing and interpreting data.

This problem is not unique to data acquisition. Users often struggle to learn new interfaces, such as when upgrading or purchasing domestic appliances (VCR, DVD player). The amount of time spent on this task fluctuates depending on the complexity and usability of the interface and in many cases, consumes more time and resources than a user is willing to spend. One proven and widely used solution to this problem is to create a domain standard so that skills learned using one device can be transferred to another device.

In this paper we discuss the design of a universal sensor interface (USI) known as Liquid [5]. The Liquid USI addresses the needs of an increasing demand for ready-to-use sensors and is the first step towards identifying a USI standard. From the this standard, we've created a toolkit that allows users to handle sensors much like they would with plug and play objects thus eliminating difficult installation procedures and the need to continually familiarise oneself with new sensor interfaces.

The following sections detail the design and implementation of the Liquid USI and its corresponding sensor toolkit. Our conclusions suggest possibilities for future USI research.

2 Design

In order to include the range of sensors our USI would need to support we decided to develop ours using a component-based architecture. This allowed us to isolate the basic elements that define a sensor and turn them into a series of expandable modules. Therefore it is possible for anyone to increase the library of sensors our USI supports.

The problem with designing a viable USI is deciding how to encompass sensors past, present and future. This wide range of sensors often leads to a narrow specification that only covers sensors from one particular domain because the property of one sensor may not necessarily exist for another. Therefore in order to categorize dissimilar sensors under the same classification scheme we need to identity some common properties that are inherent in all sensors but do not promote any ambiguity between similar sensors e.g. IR sensors can be designed to operate as either a switch in an alarm system or as a thermometer.

2.1 Sensor Classification

At the beginning of this project, we attempted to class sensors according to their primary and secondary signals (thermal, electrical, magnetic, etc) which is common practice in Engineering [6]. Our problem with this classification system is twofold.

Firstly, while this system might work well in Engineering, our preliminary user study suggested that the complexity of this classification system would be too complicated for our intended USI audience (developers of sensor-based applications). Secondly, in many cases what constitutes as a sensor's primary signal and secondary signal is unclear [6].

This uncertainty makes groupings difficult in that many sensors could be classified in several different primary signal categories. We concluded that what we needed was a method of classification that didn't focus too much on the technical side of the sensors method of data processing.

Rather than focusing on primary and secondary signals, we decided to define a series of general descriptors, which we could populate with existing sensors. There are three general descriptors by which Liquid classes a given sensor:

- *Purpose of Sensor:* context in which the sensor is used.
- *Property Measured:* property measured within the context of the sensors purpose.
- *Means of Sensing:* means by which the property being measured is monitored.

Each subsequent descriptor is a sub-descriptor of the previous one and helps to form a clear and concise sensor class. This makes adding new sensors to the USI library relatively easy; developers simply have to create a new instance of a purpose, property or means (in the latter two instances you would create a new instance under an already existing purpose or property). We define this method of categorization as Purpose, Property, Means (PPM) classification. Table 1 illustrates how the electrocardiograph of an I330-C2 sensor and a Nintendo 64 videogame controller can be categorised using this method.

Table 1: Illustration of Sensor Classification using PPM

Sensor Model	**Purpose**	**Property**	**Means**
I330-C2	Physiological Monitor	Heartbeat Rate	ECG
N64 Controller	Videogame Control	2D Axis Movement	Photocell

Using PPM classification, we can categorise literally any type of sensor under the same framework. However, before we can begin construction of a USI toolkit we need to determine an appropriate data format for each instance of a PPM combination in which to return sensor readings.

2.2 Data Formats

Sensors that belong to a given class will always share similar data properties between one another e.g. all digital cameras will return data as an image. Our problem then becomes how to support all the known formats a given class can return data in. For example, a digital camera can support several different image formats including GIF, JPEG and PCX all of which can be collected at various resolutions.

Our original intention was to use the USI to develop a sensor toolkit providing such features as the ability to fairly compare a sensors output with another of its class and a development studio for sensor-based applications. However if a particular format were not supported by some of the sensors in the given class then the former feature would be limited to those sensors that share the same data formats.

In order to prevent this, we decided that each sensor class must be assigned a single data format in which to present sensory data. The chosen data format must be the one that returns the most empirically useful interpretation of the collected data and is the most common data format that specific class. Other data formats the sensor returns should be ignored. Sensors that cannot return data in the specified format would require developers to make provisions for data conversion in an implemented USI. Sensor formats must be continually updated to ensure that the USI does not become outmoded. This method prevents the USI from invalidating itself when new sensors become available.

3 Implementation

A major concern for most developers of sensor-based applications is learning how to manipulate both a sensor's GUI and its application programming interface (API). Since there are very few standards covering the combined design of both GUIs and APIs, such a task can consume an inordinate amount of time and resources. The creation of a viable USI framework can produce a development toolkit that frees users from this constraint.

The toolkit we created using the USI framework encapsulates the command and control functions of an individual sensor's API within a single operating environment. This makes the toolkit well suited for the rapid prototyping of sensor-based technologies.

3.1 Toolkit Modules

The Liquid USI toolkit is divided into three modules (Figure 1). When combined, these modules define the sensor library (classification index) that the toolkit supports.

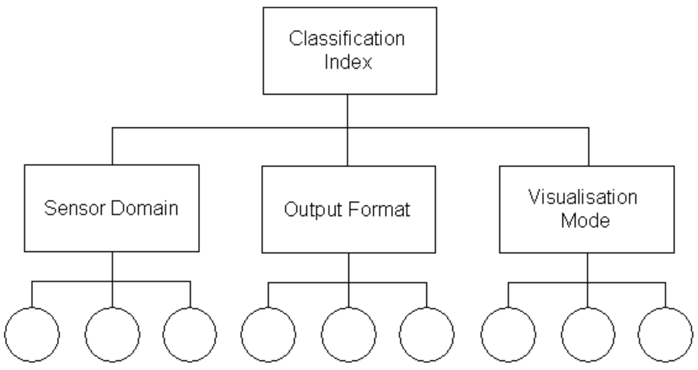


Figure 1. Illustration of the Liquid USI toolkit modules.

The *sensor domain module* defines all the PPM instances in which a sensor can be grouped. The *output format* defines how each sensor domain formats the data it collects. The *visualization mode* defines how each output format is visualized e.g. time domain data is rendered as a graph. These modules are not hard coded into the toolkit itself; therefore, the library can be extended by anyone.

Such third-party support is a vital feature of this toolkit because the number of sensors currently on the market makes it difficult to produce all the necessary information a USI would require to support an extensive sensor library. By allowing users to extend the library, we make the process easier and less time consuming for a central organization to administer.

3.2 Device Drivers

In order to directly support a given sensor we designed a way of encapsulating a sensor's API within a windows dynamic link library (Fig. 2) which we refer to as the Liquid Device Driver (LDD). Each LDD supports the same key command and control functions (which encompasses the necessary sensor-specific function calls) from which users need to develop an application e.g. activate/deactivate sensor. Because command and control is passed to the LDD, which in turn passes it to the toolkit, the user no longer has to learn that particular sensor's packaged API. As a result of this, the majority[1] of supported sensors mimic plug and play in that the user simply obtains the LDD specific to the sensor they wish to use, copies it to the LDD folder, attaches the sensor to the computer and boots the toolkit. The sensor-specific interface (both its GUI and API) is replaced by the Liquid interface.

The LDD schematic has been open-sourced [5] to allow third party developers to construct their own device drivers. Our intention is that the toolkit is treated as an online community project so that we can distribute the task of supporting the USI sensor library.

[1] Some sensors require that the API acts via a proxy and so additional software may need to be installed before use.

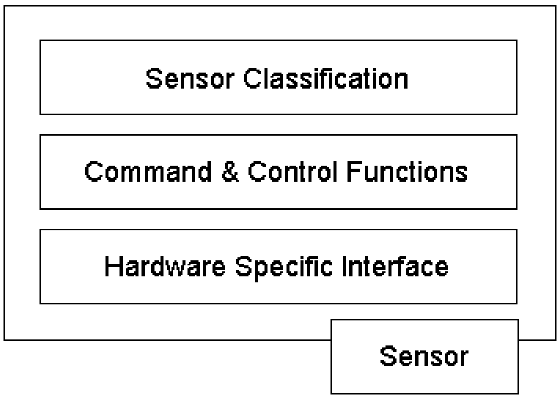
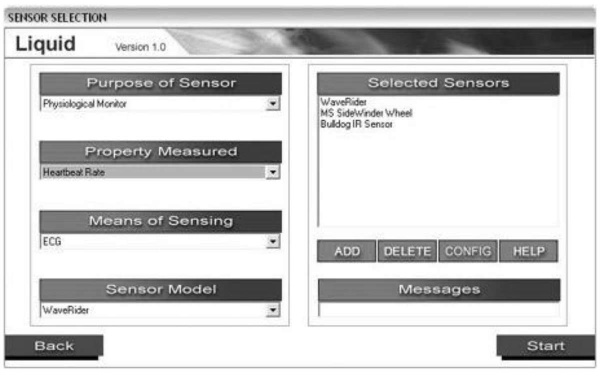


Fig. 2. Illustration of the encapsulated sensor components in an LDD

3.3. Toolkit Interfaces

As mentioned previously, a sensor's interface usually consists of a GUI and an API. The Liquid toolkit provides access to both of these interface components. The toolkit GUI (Fig. 3 and Fig. 4) allows the user to see a visual representation of the data the sensor is collecting. While in this mode, the user can compare the data collected from sensors within identical classes or identical purpose-property data format combinations in order to gauge the appropriateness of that particular sensor for a specific task. For example, using the Liquid toolkit, we could measure the responsiveness of two videogame controllers (N64 controller/Dreamcast controller) that measure motion by two different means (photocell/magnets).


SENSOR SELECTION

Liquid Version 1.0

Purpose of Sensor: Physiological Monitor

Property Measured: Heartbeat Rate

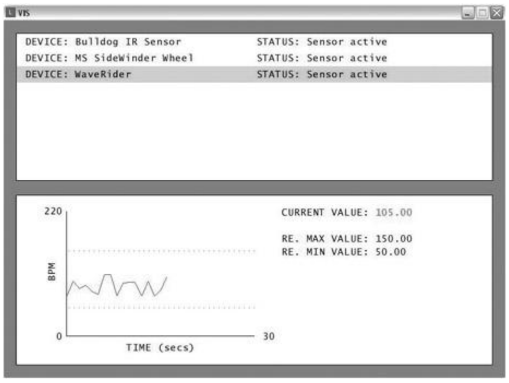
Means of Sensing: ECG

Sensor Model: WaveRider

Selected Sensors:
WaveRider
MS SideWinder Wheel
Bulldog IR Sensor

Fig. 3. Screenshot of the toolkit's sensor selection process.

The toolkit API (development studio) allows the production of prototype sensor-based applications. When using the API, the toolkit transfers sensory data across a network connection (TCP) to a prototype application. To initiate this process, users must select the appropriate sensors from the toolkit's GUI (Fig. 3).



```
DEVICE: Bulldog IR Sensor        STATUS: Sensor active
DEVICE: MS SideWinder Wheel      STATUS: Sensor active
DEVICE: WaveRider                STATUS: Sensor active

CURRENT VALUE: 105.00

RE. MAX VALUE: 150.00
RE. MIN VALUE: 50.00
```

Fig. 4. Screenshot of the toolkit's sensor readout.

Once all the necessary sensors are selected, the toolkit loads the respective device drivers and waits on the designated port for a communiqué from the users application which signals when it is ready to receive command and control of the designated sensors. When the link is established, the prototype application controls the assigned sensors through a series of simple commands e.g. activate/deactivate, getSensorOutput, getSensorStatus. Since programming complexity is contained within the DLL wrapper, the developer can control any class of sensor with the same command set. Once again, this has huge advantages in that users only need to learn one interface (whether the GUI or API).

4 Conclusions

Time permitted that we complete an informal user study with several Computer Scientists from within our own department. Users were presented with the USI and asked to complete a few simple tasks, which were orally presented to them. Our preliminary results suggest that a USI would be extremely valuable for developers in that it would eliminate the huge amount of time users normally spend on learning how to install, configure and use sensors are their sensor-specific interfaces.

As well, users suggested that such an application would be even more valuable if it was available for real-time streaming of data over the web. A central sensor repository would allow developers to share LDDs, sensor data and any knowledge they have about specific sensors and their applications. Due to the open sourced nature of the toolkit, we expect that such a system will have an extremely positive impact in the ubiquitous computing community.

5 Acknowledgements

This project was developed with the support of the EPSRC Project GR/R45253.

References

1. Abowd, G.D. and Mynatt, E.D.: Charting Past, Present, and Future Research in Ubiquitous Computing. In ACM Transactions on Computer-Human Interaction, Vol. 7(1). (2000) 29-58
2. Rodden T., and Benford S.: The evolution of buildings and implications for the design of ubiquitous domestic environments. In Technical Report Equator-02-014 (2002)
3. Intechno Consulting: Sensor Markets 2008: Worldwide Analysis and Forecasts for the Sensor Markets until 2008. Basle (1999) http://www.intechnoconsulting.com/
4. Dix, A., Finlay, J., Abowd, G., Beale, R.: Human-Computer Interaction (2nd ed.). Prentice Hall, Essex (1998)
5. Liquid: Toolkit for Rapid-Prototyping Sensor-Based Applications. Gilleade, K. M. (2003) http://sensors.comp.lancs.ac.uk/liquid
6. Gopel, W., Hesse J., Zemel, J.N. (eds.): Sensors: a comprehensive survey. Weinheim, VCH (1996)