WHAT DO WE WANT FROM A WEARABLE USER INTERFACE?

Adrian F. Clark,* Neill Newman,* Alex Isaakidis* and John Pagonis†

*University of Essex and †Symbian Ltd

Abstract

Graphical user interfaces are widely regarded as being inappropriate for use on wearable computers. This paper outlines Sulawesi, a user interface framework for wearable computing. Some shortcomings in Sulawesi are analyzed. Two specific areas of improvement, a high-level user interface toolkit and contextual measurement and adaption, are reported upon. Together these contribute towards the development of a proactive personal assistant application, the authors' ultimate aim.

1 Introduction

If you read the popular computer press, you'll quickly see that they can't conceive of any non-graphical way of interfacing the machine to the user. Even the 'battle for the desktop' between Windows and Linux seems largely to come down to which provides the more 'user-friendly' graphical interface. So one could be forgiven for assuming that a graphical user interface (GUI) is essential — that a computer system is totally unusable without a GUI — and that the 'friendliness' of the GUI is paramount.

We beg to differ. While accepting totally that GUIs make certain jobs (*e.g.*, drawing diagrams) easier, the authors believe that, a decade or two hence, we shall regard them in much the same light as the QWERTY keyboard layout: an expedient that was devised to accommodate the technological limitations of the hardware of the day. Why is this? GUIs impose a particular *style of interaction* between man and machine through the use of buttons, menus *etc.*, and applications are constrained to fit in with this style. The low level of existing GUI widget sets and toolkits contributes to this situation too. Although GUIs can be used successfully on handheld devices with low-resolution displays, as evidenced by the excellent interfaces on the Psion 5 series of devices and some mobile 'phones, this interaction style is simply inappropriate for wearable computing [1]; and the paucity of wearable input devices [2] brings this into sharp focus.

So what will replace GUIs? We believe that future user interfaces will support human–computer interaction using a range of modalities. Freed from the tyranny of conforming to a set of GUI guidelines, we are confident that applications will do more work 'behind the scenes' to provide and filter information before presenting it to the user.

This paper outlines some research the authors have carried out in this direction, and reports some preliminary steps in developing proof-of-concept demonstrators. Section 2 describes the Sulawesi user interface framework, our first foray into this area, and Section 3 identifies some of its shortcomings. Section 4 then describes the authors' preferred metaphor for a wearable user interface. Section 5 considers context and Section 6 mechanisms for contextual adaption, as much of our recent research has focussed in that area. Finally, Section 7 give some concluding remarks.

2 Sulawesi

We realized very early in our experiments with wearables that user interface issues would be important; and, as we were not in a position to compete in hardware terms with the big groups, we decided to concentrate on that. It seemed obvious to us that the way in which a user might wish to interact with the computer will depend on the situation: for example, while sitting on a train one might be happy to use visual output but while walking one would prefer spoken output. This means that the user interface must be *multi-modal*, able to use several different modes of input and output. Moreover, it should be able to change modes depending on what the user is doing: *context-sensitive*. A simple example of this is for the computer to monitor the signal from a microphone to see whether the user is speaking, and only to speak to him or her when no-one is speaking — a polite user interface!

In an effort to meet these needs, we designed and built *Sulawesi*, a user interface framework that is flexible enough to encompass a wide range of interaction techniques, can be adapted through well-defined programming interfaces, and can be tailored for a specific purpose. Sulawesi comprises three distinct parts (Figure 1):

- an input stage, on the left side of the figure, which gathers raw data from the various sensors attached to the wearable;

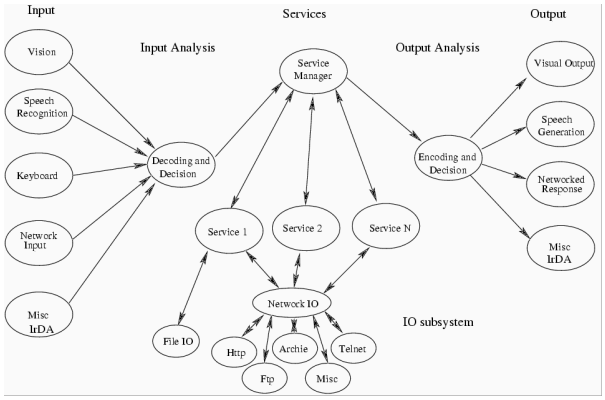

Figure 1: Architecture of the Sulawesi framework

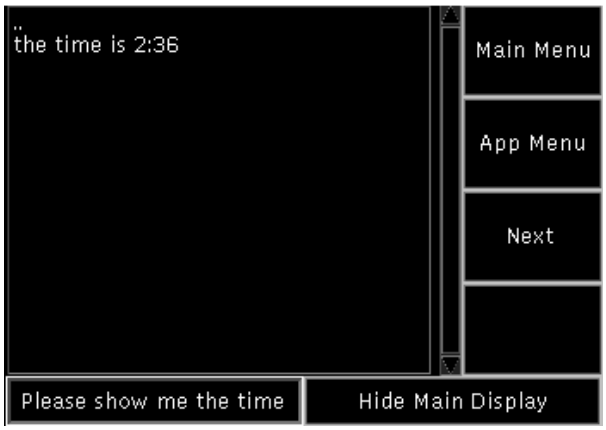

Figure 2: GUI used within Sulawesi

- a core stage, the middle region in the figure, which contains a natural language processing module and service agents (applications) to process information gathered from the input stage and produce, where possible, a mode-neutral output;
- to the right of the figure, an output stage which decides how to "render" the results from the service agents.

The agents in the system can monitor the input and output streams of data and autonomously retrieve information depending on how the system perceives the user's environment.

At the heart of Sulawesi lies a very simple natural language parser which converts a sentence (either typed or spoken, the latter using IBM's ViaVoice recognition engine) into a command stream. This speech module can communicate spoken words, sentences, or even just the knowledge that somebody is talking, to the system core. Two pieces of information are extracted from a sentence: the service to invoke and how the output should be rendered. The list of services is constructed dynamically by Sulawesi during its initialization, while the verbs controlling the renderers are stored in a look-up table, which makes for easy customization.

This semi-natural language processing is perhaps best explained by a simple example. The two phrases

```
could you show me what the time is?
please tell me the time
```

both result in the user being informed of the time, as the word `time` is recognized as being the name of one of the services available in Sulawesi. The first example uses `show`, which causes Sulawesi to render the output visually, while the second uses `tell`, which implies audible output. The important point about this approach is that the machine infers meaning from a relatively natural sentence, rather than the user having to adapt to the machine and remember complex commands or manipulate a user interface.

With visual output, we have found that a simple GUI is worthwhile (Figure 2). This has been designed for low graphical resolution and color depth (only 320×240 grey-scale pixels). A high-contrast GUI is essential here, as head-mounted displays are transparent, superimposing their output on the wearer's view of the real world. The main graphical widgets have been placed out of the user's foveal field of view. A text entry box is at the bottom left of the user interface, and a row of menu-selection buttons are on its right-hand side (for a right-eye view; the buttons can be switched to the other side for a left-eyed user). The buttons provide some form of visual feedback by flashing black-on-white for a brief period when pressed. The main panel is used by applications for their displays. A mechanism is provided within Sulawesi to allow multiple applications to be 'stacked,' only one being visible at any one time. The buttons provide the mechanism to flip between applications, a scheme that was developed to keep the field of view as uncluttered as possible. When the desired application is in focus, the actions of the buttons may be overridden by it to allow some simple forms of user interaction. The last button ('Show'/'Hide') gives the user the ability to turn off the main panel, thus freeing up most of the field of view.

Some of the more interesting applications built on Sulawesi make use of the wearable's location, found using GPS outside and infra-red beacons within our building. Some effort was expended so that applications were independent of the source of the location information. This was achieved via an *information abstraction layer* (IAL) which converted the information produced by sensors into meaningful locations such as "Home."

The IAL approach also provides a mechanism for Sulawesi itself to control its responses. For example, the

wearable's movement sensor tells Sulawesi when the user is sitting, standing or walking. A simple use of this is to blank the GUI when the user stands up and walks. However, more sophisticated behaviours are possible. For example, if the user asks Sulawesi to show him or her the time, this would normally cause visual output. However, if the wearer then stands up, Sulawesi will normally override the requested rendition and render the information audibly. Information from several sensors may be involved in this decision-making: if the user is sitting but the GPS unit determines that he or she is moving (and is hence presumably travelling in a vehicle), audible output is again used. The rules that govern the switching of rendering are configurable.

With the ability to sense position, it is natural to provide a Sulawesi application that produces reminders depending on where a user is — a *spatial reminder* service. A user enters a reminder (either by typing, speaking, or using any other available input device) with phrases such as

```
when I leave work, remind me to
do some shopping
```

and

```
when I arrive at the shops, remind
me to get some bread and milk
```

Here, "work" and "the shops" are locations that have been stored in Sulawesi. As the user leaves the location "work", the corresponding reminder is triggered; and as he or she arrives at "the shops", the shopping list of bread and milk is output — the particular mode of output being selected by the system depending on what the user is doing.

3 Sulawesi's shortcomings

Although there were a few innovations in Sulawesi, there were also a few shortcomings. For example, there were surprisingly many problems with the implementation language, Java. This was partly because the JDK's class library was undergoing significant changes as Sulawesi was implemented; but there were also a number of cases where we had to implement low-level code in C and interface that to Java. The multi-threaded nature of the application also made for a few frantic debugging sessions. However, there were more significant problems.

Firstly, it was difficult to switch from speech rendition during output, largely due to shortcomings in the control of the speech synthesis engine. We have experimented in supplying the engine with one word at a time

Figure 3: Selecting from a number of choices

but this causes the output to become stilted; we settled on a phrase (delimited by comma, semi-colon, full stop, *etc.*) as the practical minimum that had to be spoken as a chunk.

Secondly, Sulawesi provided no mechanism for disambiguating commands. This problem does not arise in GUIs as, being programmed explicitly, there is no possibility of ambiguity. What is needed is a way of engaging the user in dialogues that may be in any of the supported modes.

To address the latter issue, we are in the process of devising a high-level dialogue toolkit that can switch between input and output modalities. Written in Perl, dialogues are delimited by `begin_dialogue` and `end_dialogue` calls, within which are invocations of a series of procedures, each of which is for a particular type of input. A simple example is

```
begin_dialogue ();
yes_or_no (\$ans, "Are you sure?", 1);
end_dialogue ();
```

where the final, optional, argument to `yes_or_no` indicates that the default is "yes." For textual interaction, the prompt is generated and the user enters the reply; essentially the same approach works for spoken interaction. If a full GUI is in use, the entire dialogue is contained within a pop-up window.

This approach to dialogue construction is also amenable to other forms of interaction. For example, when the user has to choose one of a number of options, the toolkit is able to display a number of arches (Figure 3); the user then just has to nod towards the arch corresponding to his or her choice; the choice is determined by the azimuth measured on the HMD and the selection by the way elevation changes.

However, by far the most important shortcoming in Sulawesi was the rule-based nature of its adaptation to context. Either the developer had to work out a series of rules to accommodate context, or the user had to be able

to identify situations in which the system should adapt to changes in context and express them in the rule base. Wearables would be much better if they were able to adapt to patterns of use automatically, a subject we shall return to in Section 5.

4 A better metaphor for a WUI

The authors are of the opinion that, rather than the desktop metaphor of a GUI, the right metaphor for a WUI is a *personal assistant* (PA). A good (human) personal assistant is unobtrusive, predicts what information is needed and prepares it in anticipation of its need, schedules meetings and appointments, *etc.* — precisely the *desiderata* of a WUI. Although there is no need why a "wearable PA" should be anthropomorphic, this may be desirable as human–human interaction is much more natural than human–computer dialogues. Achieving human characteristics may involve aspects of affective computing [3]; it is certainly desirable to embue the software with the ability to adapt to the wearer's mood. For example, immediately after the author has given a lecture, we might envisage a dialogue between him and his wearable PA of the form (where the wearer's inputs are emboldened):

> **PA wakeup.**
> While you were teaching, there were two telephone calls to your office and six incoming emails. None of the emails were marked urgent but one of them is from the head of department. Would you like to process it now?
> **No. Did the telephone callers leave messages?**
> Yes. I can play them to you; but since that lecture was to first-year students, why don't you have a cup of coffee first, like you usually do after seeing them?
> **OK. Where is Alex?**
> Alex is outside the campus coffee bar.

Note that this exchange would work equally well audibly or in visual form — but it is very different from the types of interaction that occur with a GUI. We believe that these kinds of dialogues can just about be handled with simple natural-language parsers such as that encountered in Sulawesi. Note however that even this level of PA support requires a network connection, a computer in the wearer's office capable of controlling a telephone, access to the wearer's incoming email, and position-sensing technology in the wearable computers. It may even require a multi-processor architecture, so that decoding and playing out continuous media does not inhibit interaction with the user. (Indeed, such tasks may be carried out most appropriately by specialized co-processors.) Furthermore, the wearable PA has monitored the wearer's habits and is trying to schedule things to accommodate them, something that is just about possible to achieve on today's systems.

5 Measuring Context

Before one can consider adapting to context, one must be able to determine it. It is useful to distinguish between two classes of context: **environmental context**, the situation in which the wearable finds itself; and **system context**, what is happening on the wearable system. Environmental context includes *e.g.*, location, time, and whether the user is standing, sitting or walking. This type of context appears to be the most useful for characterizing information and can usefully be stored along with application data for, say, subsequent searching. Several authors have suggested using an XML-based scheme to integrate the contextual information with the application-level data. Acquiring environmental context involves the use of explicit sensors: GPS, real-time clock, and so on.

System context includes indicators such as which programs are active in a particular time period, the network load, the rate of input events, and so on. It can be estimated, at least under Linux, by reading parts of the `/proc` filesystem or by monitoring X events. One can identify how busy the user is by monitoring the rate of X input events and screen updates (even the simple GUI in Figure 2 is built on top of X). One can assess what the user is doing to some extent by monitoring which programs are using processor time or have focus — though with multi-functional programs such as Emacs, this is far from clear. Network activity and open files help determine the user's task too.

Our belief is that system context is less useful then environmental context for characterizing data. As you have seen, however, it is invaluable in identifying what task the user is doing — which gives us the source information for being able to develop so-called *proactive* applications that anticipate the user's needs, precisely what the wearable PA needs.

6 Achieving Contextual Adaption

Several approaches to contextual adaption have been explored. One of these, genetic programming, works but is too unwieldy to use on resource-limited machines such as wearables [4]. Our work has focussed most recently on cluster analysis, a well-established pattern recognition approach. It has been applied to a wide range of problems, including some types of contextual adaption. However, some care is needed in applying any type of cluster analysis to estimating system context. This is best illustrated by an example.

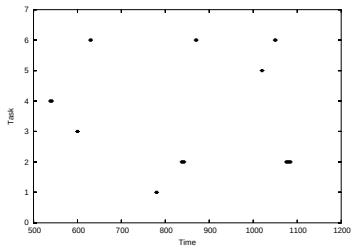

Figure 4: Typical system usage data

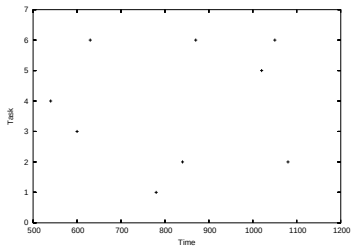

Figure 5: Clusters of activity in the data of Figure 4

Let us consider a fairly typical set of usage data (Figure 4), which shows how one person's activities varied over time during part of a particular day. (One can, of course, produce similar graphs relating task to other types of context such as location.) The values along the ordinate axis represent the tasks (reading mail, browsing the web, *etc.*) and are purely arbitrary. If one performs cluster analysis directly on these data, the algorithm will attempt to group together points that are similar in time *and* task, which is obviously not meaningful: there is no sense in which, say, "reading mail" is closer to "web browsing" than to "program editing." Instead, one must perform cluster analysis for each of the tasks independently.

The particular cluster analysis schemes we have considered are variations on k-means, known as PAM and CLARA [5]. Our preference is for PAM, which improves robustness to outliers by using medoids rather than means and considers all samples, but there is little practical difference between their results. The result of applying PAM to the tasks in Figure 4 is shown in Figure 5.

Having shown that it is possible to identify the combination of contexts in which particular tasks are performed, and that one can measure some types of context explicitly and others by inference, we are now bringing the two together in a "context dæmon" for Linux-based wearables which encapsulates all context monitoring and information abstraction tasks. However, we are still trying to work out whether it is possible to hide context adaption in the dæmon and interface toolkit, or whether it has to be coded explicitly into applications.

7 Concluding Remarks

Sulawesi was a reasonable first stab at a user interface framework for wearable computing. Experience with it has identified a number of shortcomings, the most significant of which are its inability to perform sensible dialogues with the user and its rule-based contextual adaption mechanism.

In response to the first of these, a high-level user interface toolkit is being developed which is able to perform limited textual, graphical and spoken dialogues with the user. This will soon support better natural language parsing than Sulawesi offered, including automatically asking for missing information and disambiguating instructions. While much more work remains to be done in this area, our initial results are promising.

Measuring environmental context is fairly well established, largely relying on the provision and integration of suitable sensors. Estimating system context automatically is more of a challenge, and we have made tentative first steps along that route. However, we have

shown that, with a little care, conventional clustering techniques can be used to identify patterns of usage. This is the essential first step towards our ultimate goal of developing truly proactive applications that learn what the user wants in certain situations and anticipates his or her needs.

References

[1] Bradley Rhodes. WIMP interface considered fatal. In *Proceedings of VRAIS98*, 1998.

[2] Neill J. Newman. *Systems and Services for Wearable Computers*. PhD thesis, University of Essex, 2001.

[3] Rosalind W. Picard. *Affective Computing*. MIT Press, 1997.

[4] Alex Isaakidis and Adrian F. Clark. Contextual adaption in wearable user interface. In *Eurowearables'02*, 2002.

[5] Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley, New York, 1990.