Supporting Distributed User Interfaces in Mobile and Wearable Device Ensembles: the 2WEAR Experience

Spyros Lalis  
University of Thessaly  
Volos, Greece  
lalis@inf.uth.gr

Anthony Savidis  
University of Crete & ICS-FORTH  
Heraklion, Crete, Greece  
as@ics.forth.gr

Constantine Stephanides  
University of Crete & ICS-FORTH  
Heraklion, Crete, Greece  
acs@ics.forth.gr

ABSTRACT
The 2WEAR project explored the concept of multi-device personal computing where different wearable, portable and fixed devices communicate with each other in an ad-hoc fashion. This paper summarizes the work done to support dynamic yet controlled system formation and dynamic distributed user interfaces, highlighting the most challenging issues. For an overview of 2WEAR the reader is referred to [6].

1. INTRODUCTION
A rich variety of sensing, processing, storage and actuating nodes are embedded in different personal devices and artifacts or even clothes worn by people. In addition, computing functionality is becoming pervasive, integrated into appliances, furniture, buildings, public spaces and vehicles.

Motivated by the proliferation of digitally-augmented elements in conjunction with cheap short-range wireless technology, the objective of the 2WEAR project was to develop and experiment with technology that would enable the flexible and dynamic formation of a mobile personal computing system out of several individual parts. While 2WEAR focused primarily on devices worn or carried by the user, the idea was for such a system also to exploit non-portable devices and infrastructure in an opportunistic fashion.

This vision of a dynamically composed personal computing system radically departs from the “one-device-does-it-all” approach of the desktop world, exhibiting several challenging characteristics. Most importantly:

- **Distribution.** Instead of revolving around a single device, the personal computing system is a collection of elements that communicate with each other over wireless.
- **Heterogeneity.** The elements that make up the system can be widely heterogeneous in terms of sensing, actuating, computing and user interaction resources.
- **Dynamics.** The number of elements can change at any point in time, and it is generally not possible to rely on a fixed and a priori known configuration.

Precisely due to these characteristics, the potential benefits of such a multi-device personal computing are manifold:

- **Resource sharing.** One may exploit the resources of several devices and even combine them to achieve a result that would not have been possible when using any single device in isolation.
- **Flexible customization.** The user can combine in a flexible way different elements to produce a system configuration that is most appropriate for a given task at hand.
- **Fault tolerance.** If an element that provides a specific function fails, it can be replaced by another element that provides the same or a similar function.

However, converting this potential to useful functionality is non-trivial. In particular, the main challenge in this case is to:

*Enable useful system and application adaptation, as a function of resource availability, without requiring significant effort on behalf of the programmer or frequent explicit input from the user.*

In terms of user interaction, 2WEAR strived to achieve this goal as a combination of work on two different but related fronts. On the one hand, the short-range of cheap wireless communication (Bluetooth) was exploited to drive/control system configuration in an implicit fashion, based on physical proximity [5]. On the other hand, a user interaction framework was developed to support distributed user interfaces, transparently managing and engaging the UI resources provided by the devices that are part of the system [7]. The next sections describe these aspects in more detail and discuss key highlights and challenges.

2. DRIVING SYSTEM FORMATION VIA PHYSICAL PROXIMITY
Given the short range of cheap wireless technology, 2WEAR investigated the possibility of controlling the formation of the (mobile) personal computing system (or personal area network) based on the physical proximity of devices.

In essence, the idea is to let the user add or remove a device to/from his personal area network merely by approaching respectively by distancing himself from it. A similar result can be achieved by picking up and carrying/wearing a device, respectively by leaving it behind when it is no longer needed.
Relieving the user from having to manually control system formation via explicit commands becomes very important when the addition and removal of devices is the rule rather than the exception. Indeed, the personal computing system advocated in 2WEAR seamlessly grows or shrinks each time the user changes environment, from the house to the car, to the office, to the car, to the restaurant, to the car and finally back home.

Such an implicit system formation and enabling/triggering of functions is even more crucial when the system comprises devices with limited user interfaces. As an extreme example, imagine a credit-card sized device with no user interface whatsoever which is used to run one or more personal applications and another device such a wristwatch that has some display capability. While it is not possible for the user to explicitly tell the former to direct application output to the latter, this can be achieved indirectly, by bringing these devices close to each other.

To support proximity-driven system formation 2WEAR implements a resource discovery layer and protocol on top of Bluetooth, letting a device advertise its own local resources and functionality as well as be informed about other remotely available resources and functionality. This discovery layer is used by higher-level system services or directly by applications.

3. SYSTEM SUPPORT FOR DYNAMIC DISTRIBUTED USER INTERFACES

The ad-hoc nature of the envisioned system and the variety of devices that can become part of the system may lead to unusually dynamic and heterogeneous configurations in terms of user interaction resources. While it is possible to let applications engage UI resources in a direct way, in this case the programmer becomes responsible for explicitly discovering and combining these resources to perform the desired interaction. Also, failure to communicate with a (remote) UI resource must be handled, e.g. by replacing it with another resource of equivalent or similar type.

To relieve the application programmer from this task 2WEAR developed a framework that supports dynamic and reconfigurable user interface elements while taking into account the characteristics of small wearable/portable devices with restricted UI capabilities and slow wireless communication. The approach is to let the application use abstract dialog objects that are internally implemented based on one of many possible bindings to concrete primitive UI resources. Two such object classes were supported: (i) the selector class, allowing the user to select one of a list of textual options; and (ii) the text-entry class, enabling the user to enter/edit textual data.

The selector and text-entry classes are implemented using a 2-level hierarchy for the dynamic management of UI resources. Each class may engage different combinations of alternative input and output styles, and comes with the runtime logic for shifting between them in a transparent fashion. In turn, each style binds only to a specific primitive UI resource type at runtime, and encompasses the code for accessing it (over the network).

Different input and output styles can be combined to produce a large number of instantiation options for the same UI object class.

Each application may employ an arbitrary number of such UI objects to interact with the user. When invoked at runtime, a UI object will perform the respective dialog with the user as long as sufficient UI resources are available for instantiating at least one of all possible I/O style combinations. Else, user interaction (and the application) is suspended until appropriate resources are discovered, in which case a feasible instantiation is chosen and user interaction resumes.

The state of user interaction is kept within the UI object. It is updated each time an input event is received from the UI resource bound to the current input style, before visualizing the effects, if any, via the UI output resource bound to the current output style. More specifically, the selector object records the option the user has last focused on, while the text-entry object records the edits performed by the user so far and the current cursor position. Since the object state is preserved locally, as a part of the application runtime state, it can be used to initialize newly engaged UI resources in case of UI reconfiguration.

4. HIGHLIGHTS & CHALLENGES

Post-reflecting on the work done in 2WEAR and based on some, admittedly small-scale and rather informal, experiments that were conducted using the system prototype, it is worth identifying and discussing certain aspects that affected the overall user experience in perhaps subtle but important ways. To our view, some of them remain open and can inspire future work. Also, even though 2WEAR focused primarily on the lower end of wearable and embedded devices and cheap wireless technology with limited bandwidth/throughput, and supported rather simple interaction functionality (based on buttons, keypads and text displays) we believe that several of the problems tackled could be relevant for more complex distributed interfaces as well. The Spartan bodynet [2] also investigates the usage of wireless UI components and employs a wristwatch to interact with embedded applications, yet there seems to be little support for adaptive distributed interfaces.

4.1 Setting the system “boundary”

With a few exceptions (see below), the personal system should seamlessly form itself out of devices that belong to the (same) user, while ignoring other devices that happen to be around (in range of the radio). Still, given that the system can change its configuration (composition) numerous times during a few hours, it is not desirable to let the user specify or confirm the system boundary in an explicit fashion, e.g., by having to acknowledge the addition or removal of devices owned by him.

For this reason system formation is driven based on a user identifier which devices advertise together with their resource information. This identifier needs to be assigned to a device via an offline initialization process. When searching for resources, a device considers an advertisement (and proceeds to interact with the respective device) only if it has the same user id. A special identifier is reserved to denote public devices which may become part of the personal system of any user.

Device sharing between different users is more tricky to support, in particular because this must be done on a per-device (rather on a per-person) basis. This is achieved via a list of “friendly” devices that is consulted at runtime to decide whether to consider a remote device/resource that does not have the same user or the public id. Conversely, a request from a remote device that does not bear the same user id is accepted only if this device and user id is found in the list.
One problem was that this list has to be maintained by hand at both parties. For instance, if A wants to make his device D1 accessible for device D2 of B, A must edit this list at D1 to add D2) and B must edit this list at D2 to add D1. This is rather inconvenient, especially if such a pairing is temporary and must be undone (again, by editing this list). Also, no support was provided for editing the list in a remote way, e.g., for devices that do not feature an appropriate user interface.

While in the general case some means for explicitly pairing devices that belong to different users is indeed required (if only for security reasons\(^1\)), it is important to think of approaches that will simplify this process. For instance, a device could be admitted in one’s personal system via a special device that transmits security credentials (or a shared secret) via near-field communication. An approach in this spirit is described in [9] for adding devices in a smart home system.

### 4.2 On-body vs. peripheral resources

Besides the functional type of UI resources that are provided by devices, 2WEAR explicitly differentiates among two categories of devices: (i) “on-body” mobile or wearable personal devices the user carries with him; and (ii) “peripheral” devices that remain fixed at their locations and do not move with the user.

This distinction is relevant because it can be used to infer the expected availability of a certain device and resource. On-body devices are part of the personal system that moves together with the user; hence the corresponding resources are likely to be rather “stable” (from the perspective of the mobile user). In contrast, peripheral devices are left behind when the user moves between different locations thus their resources appear to be more “volatile”. The application UI logic can be designed to exploit this knowledge, e.g., by assigning controls to on-body devices and opportunistically (re)directing output to peripheral devices.

In addition, such a distinction can be used as a rough assessment of certain qualities of a UI resource being advertised, without requiring a detailed meta-data description. For instance, a peripheral display is probably bigger and supports a finer resolution than a display provided by an on-body device. Conversely, an on-body display is probably better suited for a private communication compared to a peripheral, even within one’s own office, car or home.

### 4.3 UI configuration preferences

When several devices with UI resources that belong to the same person (or are part of the friend list) are in range of each other, different UI instantiations are feasible. In this case, it is unclear (to the system) which option to engage; this is also referred to as the association problem in [3]. Somewhat disappointingly, but not very surprisingly, even the short range of Bluetooth (roughly about 10m) often proved to be too “inaccurate” for the purpose of implicitly (re)directing the application UI as desired. Hence, additional information, besides the mere availability of UI resources, is required to drive UI (re)configuration.

This problem is addressed by letting the user define his personal preferences regarding the ranking of different instantiations for each abstract UI object class. While this reduces undesirable (and unexpected) behavior, the management of such preferences can become quite tedious, especially when one has to deal with numerous styles and primitive UI resource bindings. Fortunately, this can be performed offline. In the real world one would also use an existing profile as a starting point rather than specifying everything from scratch. To deal with exceptions, the user can change the current UI instantiation at runtime, by picking a different (feasible) option.

However, in 2WEAR it was not possible for the user to introduce a new style, in fact this required a modification of the UI software. A more flexible approach, but more appropriate for powerful devices, would be to support styles as plug-ins, perhaps even based on a declarative specification (e.g., in XML) that could be provided by the user himself, via a GUI, also at runtime.

Learning techniques that have been successfully applied to other areas of ubiquitous and pervasive computing [1] could be employed to further reduce the amount of explicit input needed. Also, especially for the purpose of output redirection, it is worth investigating mechanisms that enable a more targeted engagement of UI resources in the spirit of the “point-and-shoot” metaphor. For instance, a resource could be taken into account only if the device that offers it is “properly oriented” with respect to the device that wishes to exploit it, which can be determined based on infrared or ultrasound sensors [4].

### 4.4 UI reconfiguration latency

The speed of UI reconfiguration obviously affects the user experience. Notably, users seemed more sensitive about this when they expected the system to engage a new device (that is turned on, or is approached by the user). Reconfiguration due to a device that becomes unavailable (the user turns it off, or walks away from it) was less crucial; probably because people have a natural allowance for “repairs”.

A large part of the resource discovery overhead was due to the delay for setting up a Bluetooth (L2CAP) connection, which was used as a transport mechanism for the actual discovery protocol. The fact that Bluetooth placed a small upper bound on the number of simultaneous connections with remote devices did not help much either. To accelerate discovery, the discovery protocol was complemented by an additional mechanism that exploited the class-of-device (CoD) field in Bluetooth inquiry packets sent as a part of the Bluetooth native discovery protocol. More specifically, the CoD was used to encode hints about the device ownership, type and resources/functionalities. As a result, it became possible to infer whether a device should be considered as a part of one’s own personal area network as well as which functionalities are (not) supported by it, as a side-effect of low-level device discovery. This reduced discovery times, enhancing the system’s reaction in terms of UI reconfiguration. In fact, the improvement was most notable in the presence of many devices that provided different resources, in which case conventional higher-layer discovery was especially problematic because it incurred a large delay while resulting in very few “positives”. A similar, more elaborate, approach for reducing discovery latency also supporting the fragmentation and re-assembly of service advertisements is discussed in [8].

---

\(^1\) The security problem is less crucial for devices that belong to the same user because they can be pre-configured to share a secret; just as this must be done for the user id.
A similar issue arose when performing a UI reconfiguration to individual reconfiguration scenarios (transitions between UI delay could be specified by the user, perhaps also separately for communication with the device will be re-gained soon. This reconfiguration when the network reports a problem, in the hope for this reason the default system behavior was to simple, corresponding to rather short-lived dialogs.

A positive side-effect of this approach was that peripheral UI resources were engaged only at the user's explicit consent, thereby also protecting the user's privacy.

A mutual exclusion scheme was adopted to ensure that each primitive UI resource will be used by at most one application at a time. This required the implementation of a locking protocol between UI resources and their clients. The discovery protocol and CoD-based mechanism also had to be modified to stop advertising a UI resource as soon as it was locked by a client. Notably, a device might attempt to access a UI resource based on outdated (cached) information, in which case unavailability is discovered when the client attempts to lock the resource.

Of course locking introduces other problems. Namely, deadlocks or livelocks can occur when two or more applications invoke composite UI objects that compete for the same primitive UI resources. 2WEAR did not address this issue. One solution is to lock resources in increasing order using a globally common numbering scheme, e.g., the address of the device featuring the resource combined with a local resource identifier. Nevertheless, an application may stall for a long time, waiting for a particular UI resource to be unlocked; unless a different UI instantiation is feasible that does not rely on that resource. As an additional refinement, one could introduce priorities, letting more urgent applications interrupt the dialogs of less urgent ones.

5. ACKNOWLEDGMENTS

The 2WEAR project (2000-2003) was funded by the EC as a part of the IST/FET programme. During that period Spyros Lalis was a research associate of ICS-FORTH through which he received funding for the 2WEAR project. The UI framework of 2WEAR was implemented by the HCI group of ICS-FORTH.

6. REFERENCES


