Enabling multi-device interaction on the go in the MAGI project

Kenny T.W. Choo
School of Information Systems
Singapore Management University
80 Stamford Road
Singapore 178902
kenny.choo.2012@smu.edu.sg

Quentin Roy
School of Information Systems
Singapore Management University
80 Stamford Road
Singapore 178902
quentin@quentinroy.fr

Richard C. Davis
School of Information Systems
Singapore Management University
80 Stamford Road
Singapore 178902
rcdavis@smu.edu.sg

Submitted to CHI 2016 Workshop on Interacting with Multi-Device ecologies "in the wild".

Abstract

We discuss the MAGI project's vision for multi-device interaction and how we plan to support application developers. Key aspects of our vision are the presence of multi-device gestures and adaptation to changing physical contexts and device contexts. Our gesture recognition architecture reduces power consumption and recognition latency through a pipelined HMM approach with early discard of unlikely candidates. To help developers build applications that adapt to changing contexts, we propose the use of *Midgets* (MAGI widgets): context-dependent interaction components that span across devices. Application designers choose the device configurations that an application will support, assign them to physical contexts, and lay out *Midgets* manually. The MAGI system chooses the best configuration based on users' current activity and the input/output channels available.

Author Keywords

Wearable devices; Multi-device interaction; Distributed User-Interface; Gesture; Gesture recognizer; Power consumption; Distributed recognition.

ACM Classification Keywords

H.5.2 [Information interfaces and presentation]: User Interfaces – Input Devices and Strategies.

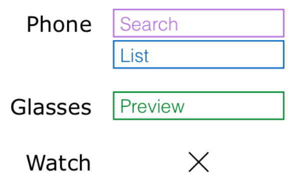

Figure 1: *Midgets* running on each device in scenario 2 part (a): seated, with the phone out.

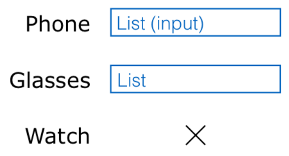

Figure 2: *Midgets* in scenario 2 part (b): walking, with phone in hand.

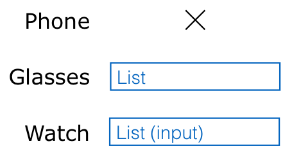

Figure 3: *Midgets* in scenario 2 part (c): sitting, without the phone.

Introduction

Wearable devices such as smart watches are gaining widespread interest from consumers. While wearables provide easy access to content outputs such as notifications, emails or text messages, each device has limited input and output capabilities. We envision a personal multi-device ecosystem where applications span across all of a user's devices; ranging from smart watches, smart glasses, smart bands to smart phones. Inputs and outputs are distributed based on which devices are available and the automatically detected context of use (e.g. walking, sitting or cycling).

This paper briefly outlines our vision, called MAGI (Multi-device Adaptive Gestures & Interfaces) [3]. We aim to enable developers with an Android framework that provides high-performance, multi-device gesture recognition and primitives for interface adaptation. Here, we illustrate this vision with two scenarios. We then show how we are addressing the challenges of multi-device gesture recognition with a novel pipelined HMM approach with early discard of unlikely candidates. Finally, we describe our vision for *Midgets* (MAGI Widgets), which are context-dependent hierarchical interaction components that span across devices.

Multi-Device Scenarios

Consider the following scenarios, which show the types of applications that we intend MAGI to support.

Scenario 1: Steering in a driving game.
While riding on a train, Alice is playing a driving game on her virtual reality display, smart watch, and smart phone. With her watch on one hand and phone in the other, she turns an imaginary steering wheel. She uses the touchscreen buttons on the phone to accelerate or brake. Her phone vibrates, indicating an incoming call, and Alice raises her phone to her ear. When she does this, the game pauses, and Alice takes her call.

Scenario 2: News feed consultation.
(a) Clara is sitting at a café, browsing the titles of her news feed on her phone. The list can be filtered by a search field, also shown on the phone. As she scrolls through the list of articles, her smart glasses preview the first lines of the topmost article at the periphery of her vision (see Figure 1). (b) Later, Clara gets up and starts walking to the metro station, holding her smartphone at her side, without looking at it. The list of articles moves to her glasses so she can read while walking, and she scrolls the list by swiping on her phone (see Figure 2). (c) When Clara boards the crowded metro train, she puts her phone in her pocket to keep it safe. Still browsing articles on her glasses, she now begins scrolling by swiping on her watch (see Figure 3).

These scenarios illustrate two key challenges. The first scenario uses a continuous, multi-device gesture that is particularly sensitive to latency. Recognizing such gestures will place high processing demands on wearable devices, which increases power consumption. The second scenario shows a user's changing context. The application must adapt by choosing optimal input and output channels. MAGI will address these challenges.

Gesture Recognition Challenges

Each device employed in the personal space will have to deal with two problems (1) recognition latency, and (2) energy consumption. By nature of their size, the computational capabilities and energy resources of mobile devices are limited. Most previous works adopt a centralized approach [1,4]. Sensor inputs (e.g.

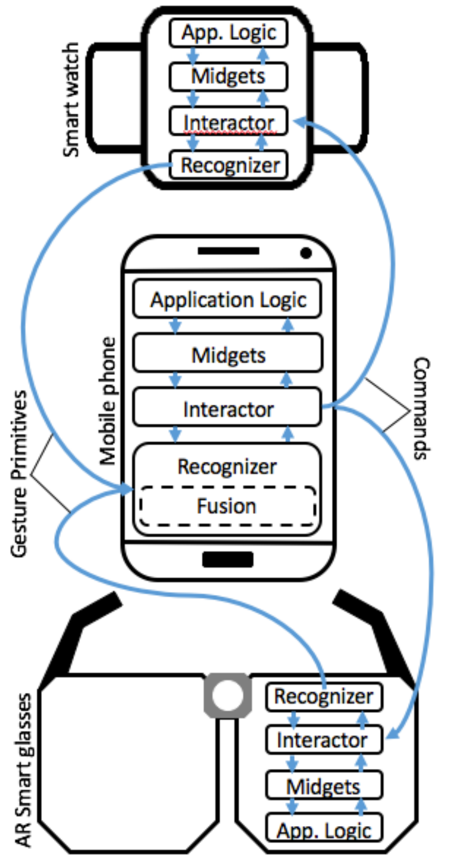

Figure 4. The MAGI architecture supports distributed gestural interfaces that are able to automatically adapt themselves to different contexts of use.

accelerometer and gyroscope data) are all streamed to a single device that handles the recognition process. This approach has two disadvantages. The first is continuous sensor data transmission, which consumes excessive energy in both the sender and the receiver. The second is poor scaling. As the number of devices in the ecosystem increases, a centralized recognizer on a mobile device can rapidly be overwhelmed. We propose to tackle these demands through collaborative and adaptive systems and algorithms.

We have developed an on-device, early-filtering Hidden Markov Model-based (HMM) gesture recognizer that improves the speed and reduces the latency (and hence energy efficiency) of two-handed gesture recognition (see Figure 4) [3]. Gesture recognition is distributed as much as possible to reduce network usage. The method attempts to recognize gestures early. If an entire gesture is recognized with high likelihood by an HMM, then a prefix (an initial part) of the gesture should also have high likelihood. Our approach thus permits *early discard*, avoiding processing samples whose prefix indicate that it is unlikely to be a gesture. Also, our method is heavily *pipelined*, reducing the latency of gesture detection by performing Viterbi decoding concurrently with the generation of subsequent gesture samples. Since our method performs gesture recognition on-device, we significantly reduce the throughput which otherwise would comprise sending multiple raw sensor feeds into a central device and having that perform all the work. In the tests reported previously, our system recognized gestures with 89.86% accuracy in approximately 0.2ms [3].

In the future, we plan to let devices assist each other in the recognition process by notifying each other when it is safe to halt HMMs for candidate gestures and conserve power. Any one of the recognition streams may provide enough information about the activity to trigger early termination of some gesture candidates. In Scenario 1, for example, raising the smart phone to the ear is out of the norm for steering gestures, and other devices can be notified to stop processing the steering gesture.

Context Adaptation Challenges

Previous researchers have studied the automatic layout of distributed applications depending on device context (e.g. Panelrama [4]). Such approaches reduce the amount of manual design required from the designer, but can lead to unpredictable behavior. We prefer to give designers more precise control over the input and output channels that their application will use while still allowing easy coordination of components and adaptation to context. This led us to develop the concept of *Midgets*, which are context-dependent interaction components that span across devices. *Midgets* are shared across devices in an application, but they can have different behavior depending both on a user's device context and physical context (recognized automatically, as in [2]).

Consider the application of Scenario 2 in three contexts. At first (Figure 1), a phone and a pair of glasses are used in conjunction. The *Midgets* of the application are spread across the two devices. When the user starts walking and stops looking at her phone, the phone screen switches to input only and the glasses hold all visual information (Figure 2). Showing all *Midgets* at once on the glass may not be a good design choice as it is likely to occlude a large portion of the user's vision. Thus, in this context, a designer may prefer to give access to only one *Midget* at a time. Interaction on the glass may be cumbersome but because the phone is still at hand, the phone remains available for eyes-free input. Finally, when the phone is put away, (Figure 3) input switches to another device.

Our vision gives designers more control, because designers will specify all the device configurations allowed by an application and will have an opportunity to design each one separately. The designer will also specify the context for which each configuration is preferred. Scenario 2, for example, uses three configurations. When sitting with the phone and glasses available, configuration (a) in Figure 1 is preferred. When walking with phone in hand, configuration (b) in Figure 2 is preferred. When sitting with the phone unavailable, configuration (c) in Figure 3 is preferred. MAGI will determine the user's context and choose the best configuration, allowing the user to override when context recognition errors occur.

Inspired by Panelrama's method of automatically arranging output layouts [4], we are also considering adding an automatic layout engine so that a designer can set some *Midgets* to automatically re-arrange themselves when an unplanned context is encountered.

Conclusion

We presented our vision of a multi-device ecosystem used on the go. We addressed system-related issues with a new distributed recognition approach, presented the architecture of our multi-device framework and discussed how it can be used by a designer to create applications that automatically adapt their inputs and outputs to their context of use.

This first work opens several challenges we are interested to address in the future. What *Midgets* will a designer need to create robust applications? How can we manage multi-application contexts and the allocation of inputs and outputs between concurrent applications? How can we support multi-user applications and allow applications to span across personal ecosystems?

Acknowledgements

This work was supported by the Singapore Ministry of Education Academic Research Fund Tier 2 under research grant MOE2014-T2-1063. All findings and recommendations are those of the authors and do not necessarily reflect the views of the granting agency or Singapore Management University.

References

[1] Houben, S. and Marquardt, N. WatchConnect: A Toolkit for Prototyping Smartwatch-Centric Cross-Device Applications. *Proc. of CHI'15*, ACM (2015), 1247–1256.

[2] Lee, Y., Iyengar, S.S., Min, C., et al. MobiCon: A Mobile Context-monitoring Platform. *Commun. ACM 55*, 3 (2012), 54–65.

[3] Tran, V.H., Choo, K.T.W., Lee, Y., Davis, R.C., and Misra, A. MAGI: Enabling Multi-Device Gestural Applications. *PerCom'16 Workshops*, IEEE (2016).

[4] Yang, J. and Wigdor, D. Panelrama: Enabling Easy Specification of Cross-device Web Applications. *Proc. CHI'14*, ACM (2014), 2783–2792.