Integrating User Performance Time Models in the Design of Tangible UIs

Paul Holleis
Embedded Interaction Research Group
University of Munich
80333 Munich, Germany
http://www.hcilab.org
paul@hcilab.org

Dagmar Kern
Fraunhofer Gesellschaft - IAIS
53757 Sankt Augustin, Germany
dagmar.kern@iais.fraunhofer.de

Albrecht Schmidt
Fraunhofer Gesellschaft - IAIS
53757 Sankt Augustin, Germany
B-IT University of Bonn
53113 Bonn, Germany
http://uie.bit.uni-bonn.de/
albrecht.schmidt@acm.org

CHI 2007, April 28–May 3, 2007, San Jose, California, USA.
ACM 978-1-59593-642-4/07/0004.

Abstract
One of the aspects that are important for judging an application is the time an experienced user needs to complete a task. This can be assessed by a Keystroke-Level Model (KLM) of that task. In this paper we show a method that allows designers to prototype hardware applications entirely in software and still be able to draw conclusions about the time to completion of given tasks on the envisioned hardware implementation. We provide versatile, easily extensible tools and examples that give developers quick access to KLM data for their prototypes and applications.

Keywords
Platform independent prototyping, user performance times, Keystroke-Level Model (KLM), tool support

ACM Classification Keywords
H.1.2 User/Machine Systems, H.5.2 User Interfaces: Prototyping, D.2.2 Design Tools und Techniques: User interfaces

Introduction / Motivation
The Keystroke-Level Model (KLM) is a specialization of the GOMS [3] family of cognitive modeling approaches. In comparison to other such models, it is relatively easy to apply to application ideas and designs since it is stripped down to only those operators necessary to make user performance time predictions. The KLM

Pictures taken from Klemmer et al. [6].

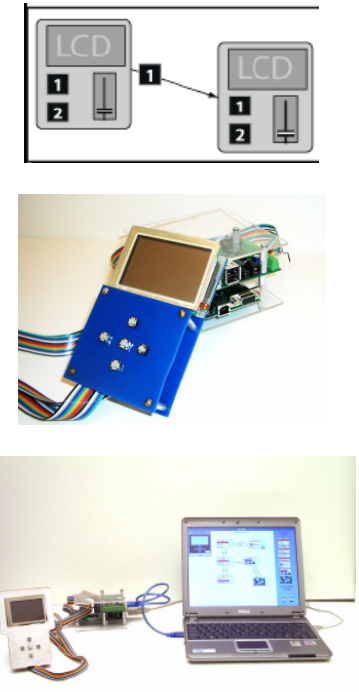

Figure 1. With the d.tools framework one can easily specify what happens when physical events occur (like when the buttons labeled "1" in the top image is pressed). Tangible prototypes (center image) are integrated such that physical changes are immediately reflected in the software environment. The bottom picture shows the prototyping setup with a physical interface connected to a computer running the d.tools system.

splits interactions into unit tasks for which a set of operators is provided. For standard desktop interaction, operators specify, for example, the time needed to execute key presses, system response time, mouse movements, etc. Time predictions for a specific task can then easily be retrieved by modeling its parts and adding the times of each operator.

Still, such approaches are used only by few developers, mainly because of the induced extra cost and missing experience. Tool support which could alleviate those issues is scarce. However, the need for such predictions is prevalent, especially for physical devices. Prototyping hardware user interfaces and appliances is a tough and time consuming job. KLM offers the possibility to also test physical user interfaces with regard to interaction times without the need to build such interfaces in the first place. We present tools and examples how KLM can be used to generate time metrics for interactions on different target platforms using software prototyping only. We offer an extensible system that can be used to convert demonstrated interactions into time values for specific hardware platforms. We currently provide support for two scenarios, namely advanced mobile phone interactions and preliminary for in-car interfaces.

Related Work

Prototyping has a long tradition as a means to try to avoid introducing errors or obstacles in costly systems. Since prototypes are supposed to be created quickly and cheaply, tools and frameworks have been provided to help developers build prototypical versions of their envisioned applications or devices. For software projects, programming languages and prototyping environments have been introduced to enable a quick assembly of non- or semi-functional applications. This includes graphical user interface (GUI) builders such as Borland JBuilder[1] as well as authoring tools like Macromedia Flash[2] that have been primarily designed to support the rapid use of graphical widgets.

Hardware devices are still harder to develop than software. However, toolkits have been developed to provide basic components for hardware as well. Phidgets (Greenberg and Fitchett, [5]) provide physical building blocks to create devices that can easily be controlled from a PC through USB using a multitude of systems including C, .NET, JAVA and Flash. Stanford's iStuff toolkit (Ballagas et al. [2]) also provides some basic hardware components like buttons or card readers. In addition, they offer a software framework to dynamically map devices to applications. Similar to the EIToolkit[3] framework, there exist small pieces of software for each hardware or software component supported by the framework responsible for the communication between components. Several such systems that especially focus on enabling the use of hardware devices in software applications with only little programming knowledge necessary have been developed in the last years. One example is the EQUATOR Component Toolkit (ECT, Egglestone et al. [4]) which provides an event-based communication mechanism between components. This lets developers define the way software and hardware items connected to networked computers communicate. A graph editor allows connections between components to be drawn

[1] Borland JBuilder: http://www.borland.com/jbuilder

[2] Macromedia Flash: http://www.macromedia.com/

[3] EIToolkit: http://www.eitoolkit.de

Figure 2. User interface of an ultrasound system with a standard keyboard, knobs, sliders, buttons and a trackball.

Figure 3. User interface of a professional proportioning pump with a plastic foil keyboard.

such that it is easy to, e.g., display the value of a hardware slider in a software text field. Another, recently introduced tool that follows a state-based paradigm is called d.tools (Hartmann et al. [6]), implemented as a plug-in to the programming environment Eclipse[4]. A blueprint of the device to be prototyped can be drawn and widgets representing hardware buttons, sliders, displays, etc. can be placed on the drawing. In another editor, a state graph can be created that specifies into which state the device should be transferred on a specific action (like a button press). See the description of Figure 1 for more details. In a later part of this paper, we describe how we extended this system to integrate our time metrics (Figure 5).

Cognitive architectures like ACT-R [1] have been used in HCI to evaluate different designs with respect to human behavior. However, the initial effort to generate a thorough understanding of those architectures is very high and is likely to pay off for large or critical projects only. One of the very few efforts to reduce this large cost is the CogTool Project by John and Salvucci [12]. It uses storyboards much in the same way as the state diagram in d.tools to mockup an application. From a demonstrated task, a time model can be automatically generated using a KLM-like language called ACT-Simple. This is, however, a standalone application that, with the exception of HTML, does not allow developers to use their own prototyping environment. Besides an additional operator introduced by the authors to model Graffiti input with a stylus on a PDA, it is also specialized to the original operator values of KLM useful for office desktop settings only.

[4] Eclipse, Open Development Platform: http://www.eclipse.org/

Applications and Scenarios

A variety of applications with physical user interfaces need hardware prototypes to get information about the time to completion of a given task. This particularly applies to specialized devices for specialized tasks. In the medical domain, e.g., there are many different devices with a lot of interaction elements and possibilities like an ultrasound system with physical elements like handles, sliders, keys and trackballs (Figure 2) or a proportioning pump (Figure 3). The usual user interface of a proportioning pump consists of a plastic foil keypad. With a corresponding KLM, a software prototype can quickly be created to compare the interaction performance with this keypad to other functionally equivalent interfaces using a couple of different buttons or knobs.

Apart from such specialized devices, many designs of mobile phones try do optimize interaction speed as well. In this work, we picked up two specific examples, a mobile phone and a car air conditioning system to illustrate the benefit of using a software prototype with KLM instead of hardware prototypes.

In previous work [9], we created a KLM Model for advanced mobile phone interactions. This type of interaction includes tasks like taking pictures, gesture input or near field communication (NFC). As part of an ongoing project, an application to buy tickets for public transport which can be controlled with a mobile phone should be developed. We used physical prototypes and applied our KLM model by hand. The results of an evaluation showed that the estimates given by the mobile phone KLM are well comparable with the time testers needed for the given task. The creation of the models, however, was in cases difficult and took

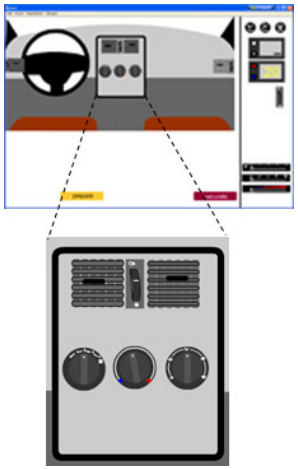

Figure 4. The prototyping tool "Car Air Conditioning" (CAC), which supports designers in the arrangement of heating interaction elements in a car interior.

Figure 5. Two different car heating systems with knob and sliders (top) and with buttons and a feedback display (center). The picture at the bottom shows two ventilation slots.

significant time. Also, whenever we changed a small part in the prototypes, we had to put considerable work into adapting the models. This proved the high potential of such an approach, but also showed that methods are required to reduce the initial effort.

For the other application area of in-car interfaces, we pursued the other method of first developing a software prototype instead of a hardware prototype. Interaction sequences can be demonstrated in a similar way to tangible prototypes, only with mouse keyboard. We are currently preparing user studies to get specific time measurements for the KLM operators.

Most car's interiors consist of different tangible interaction systems the driver has to handle, e.g., for the radio, navigation or air conditioning. Here, we concentrate on the design of the car air conditioning system and created the Car Air Conditioning (CAC) prototyping application (see Figure 4). After sampling several cars, we found two major different ways of controlling the temperature: pushing buttons with visual feedback or setting a slider or knob to the desired temperature (Figure 5). Currently, the CAC application supports the interaction element types 'button', 'slider', and 'knob'.

Imagine the following scenario: *Bob is driving on a cold day through a valley and the front window gets steamy. He wants to quickly get the window clear while driving. Bob adjusts the air conditioning system to the front window setting a high temperature, high fan power and the arrangement of the open ventilation slots to the front window. After a short time, the window is free and Bob can reset the adjustments to normal (20°, middle fan power and a centered ventilation slots).*

As a technically more advanced example, consider a couple using the same car but a different set of standard settings for mirrors, seat, radio, etc. If it cannot be completely automated, a long time to initially configure the settings would quickly become annoying.

The interaction time strongly depends on the type of interaction elements, their locations, and their way of presenting feedback (tactile, audible, or visual). Thus, a lot of physical prototypes would be needed to get information about the different operating times. With the use of a corresponding KLM, only one software prototyping tool is needed to get such time information.

Operator times

KLM was designed to apply mostly to desktop computers with a standard mouse, keyboard and monitor. However, several projects have already employed it to model a variety of application domains like visual and auditory perception (John, [11]) or a tool selection technique for tablet computers (presented by Hinckley et al. at CHI'06 [8]). Others again have made slight additions or modifications to fit new areas of interaction. Examples are stylus input on handheld devices [13] or special pointing operations for manual map digitizing [7]. There also exist several studies concerned with different techniques for text input on mobile phones, see, for example, [10] for studies and references. In addition to those projects, we were able to provide a new and evaluated set of operators as well as updated, study based time measurements for the original operators to model state of the art physical mobile phone interactions in [9].

In the same way, we are currently planning studies to measure average times necessary to create Keystroke-

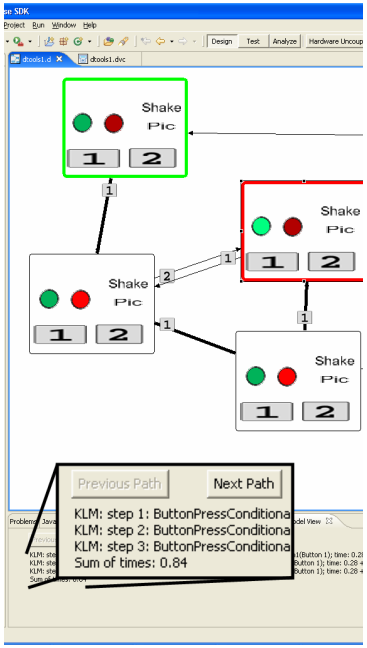

Figure 6. Extended d.tools interface with a state diagram. One path has been highlighted (from top left to top right component) and the corresponding KLM time predictions have been automatically calculated and are displayed at the bottom.

The user can browse through all possible paths to reach the desired target state. The times are automatically recomputed.

Level Models for tangible interfaces. Although we will specialize on devices mentioned in the scenarios like car and device interfaces, we will emphasize the possibilities to generalize these observations to general ubiquitous physical interfaces.

Implementation and Use

Following the prescribed component oriented design, we programmed KLM semantics into an EIToolkit component with a well defined interface. Thus, it is practically platform-independent and can even run on another networked computer. Using specific control messages, the type of KLM can be chosen, e.g., using times for mobile phones or those for another set of controls. As soon as the amount of supported devices and interaction types has grown to a certain amount, we envision a web based, database backed interface for adding and selecting sets of KLM operators.

To illustrate the simple integration of this KLM component in a software prototype we developed the CAC prototyping tool (see Figure 4), which supports designers in the arrangement of heating and ventilation interaction elements in a car interior. Because the KLM component is independent of a specific programming platform, we chose Macromedia Flash to develop this tool. With CAC, designers can arrange the elements for the air conditioning system in the car in different ways and can get the real user time performance of each arrangement using the provided KLM component. The tool shows a car interior and several interaction elements like buttons, knobs, and sliders for adjusting temperature, direction of air flow (feet, front, window) and the fan's power as well as adjustable elements for ventilation slots. The prototype has two modes. The preparation mode to design the car's interior lets the designer choose and place interaction elements. After preparing the desired scenario, the application sends messages to the KLM component containing the dimensions of the car's interior as well as the arrangement of the elements with their corresponding sizes. In the second interactive mode, participants interact with the elements to adjust the air conditioning system by demonstrating the sequence of actions for a specific task. For each action, a message is sent to the KLM component containing the element id, type of action as well as adjustment information of the element. After completing the interaction, the KLM component calculates the performance time participants would need if they had interacted with physical interaction elements and returns the result to the prototype which displays the total interaction time.

As a second way of using the KLM component, we used the functionality of the d.tools Eclipse plug-in (presented above and in Figure 1). To the graph view that shows possible transitions between all possible states, we added a tool that lets a user choose an initial state A and a target state B (see Figure 6). The user can browse through the list of all computed paths from A to B which are highlighted in the view. For the currently selected path, the KLM component is used to calculate the predicted interaction time for this sequence of transitions. Transitions are mapped to KLM operators and are augmented with properties that specify if and how long additional times should be used, e.g., for separate thinking or system response time. These values are then added by the KLM component.

Summary and Conclusions

We presented an approach to use Keystroke-Level Modeling (KLM) to calculate predictions on skilled user

performance time for interactions with tangible interfaces. This method considerably lowers the hurdle to employ such models by allowing developers to use their own rapid software developing tools and can spare them from the need to create physical prototypes. This is one of the lessons learned from a project that applied KLM to advanced mobile phone applications. Besides phones, our work currently focuses on supporting specialized devices and interfaces like those found in cars. Two ways of using the provided KLM component were presented: through a specifically designed application for prototyping in-car interfaces and through a rapid prototyping environment for general tangible user interfaces from the Stanford HCI Group.

Future Work

In the present work we concentrated on user interaction time predications with KLM. However, since we designed our system in a component oriented way, we plan to extend it and integrate support for other non-functional parameters. Envisioned functionality for the CAC prototyping tool, for example, includes the degree of driver distraction while manipulating controls. The overall goal is to enhance our prototype developing approach to general tangible user interfaces with new components and types of interactions. It could then check interfaces against existing guidelines and deliver a variety of additional metrics for such user interfaces.

References

[1] Anderson, J., Lebiere, C., The Atomic Components of Thought. Lawrence Erlbaum Associates, Inc. 1998

[2] Ballagas, R., Ringel, M., Stone, M., Borchers, J. iStuff: a Physical User Interface Toolkit for Ubiquitous Computing Environments. In *Proc. CHI '03*. 537-544 ACM Press. 2003

[3] Card, S. K., Moran, T. P., and Newell, A., The Keystroke-level Model for User Performance Time with Interactive Systems. Comm. ACM 23, 7.396-410. 1980

[4] Egglestone, R. S., Boucher, A., Greenhalgh, C., Humble, J., Law, A., Pennington, S., Rodden, T. Supporting Collaboration in the Deployment of Ubicomp Experiences. In *Proc. UbiSys'06*. 2006

[5] Greenberg, S. and Fitchett, C. 2001. Phidgets: Easy Development of Physical Interfaces through Physical Widgets. In *Proc. UIST '01*. 209-218. ACM Press. 2001

[6] Hartmann, B., Klemmer, S.R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., Gee, J. Reflective Physical Prototyping through Integrated Design, Test, and Analysis. In *Proc. UIST'06*. 2006

[7] Haunold, P., Kuhn W., A Keystroke Level Analysis of a Graphics Application: Manual Map Digitizing. In *Proc. CHI'94*. 337-343. 1994

[8] Hinckley, K., Guimbretière, F., Baudisch, P., Sarin, R., Agrawala, M., Cutrell, E., The Springboard: Multiple Modes in one Spring-loaded Control. In *Proc. CHI'06*. ACM Press. 181-190. 2006

[9] Holleis, P., Otto, F., Hußmann, H., Schmidt, A. Keystroke-Level Model for Advanced Mobile Phone Interaction. To appear in *Proc. CHI '07*. 2007

[10] James, C. L., Reischel, K. M., Text Input for Mobile Devices: Comparing Model Prediction to Actual Performance. In *Proc. CHI'01*, 365-371. 2001

[11] John, B. E., Extensions of GOMS Analyses to Expert Performance Requiring Perception of Dynamic Visual and Auditory Information. In *Proc. CHI'90*. ACM Press. 107-115. 1990

[12] John, B. E., Salvucci, D. D. Multi-Purpose Prototypes for Assessing User Interfaces in Pervasive Computing Systems. IEEE Pervasive Computing 4 (4), 27-34. 2005

[13] Luo, L., John, B. E., Predicting Task Execution Time on Handheld Devices Using the Keystroke-Level Model. In *CHI'05*. ACM Press. 1605-1608. 2005