Using the Keystroke-Level Model to Evaluate Mobile Phones

Trenton Schulz
Trolltech ASA
University of Oslo
trenton.schulz@trolltech.com
trentonw@student.matnat.uio.no

Abstract. Developing applications and services for mobile devices can be a challenging task. One classic evaluation method available to desktop machines is the GOMS Keystroke-Level Model that measures the keystrokes, mouse movements, and mental preparation of an expert user performing a task error-free. A model can be used for determining how long it takes to perform a task. We introduce a tool that can generate Keystroke-Level Models and use it to evaluate some mobile phones. Examining the models, we find useful information about the mobile devices, but there could be some adjustments to the tool and the Keystroke-Level Model approach for generating better models of the mobile device interaction.

Introduction

Designing for mobile devices involves dealing with many issues. Mobile Devices typically have smaller screens, limited input, less processing power, and work on a battery for a long time. Typically, standard metaphors and ways of development for desktop applications do not transfer well to mobile devices because they do not take these issues into account. An open question is if methods for user interface evaluation from the desktop would work for mobile devices.

Card et al. (1983) introduced GOMS for evaluating user interfaces. A simplified version was also introduced called the Keystroke-Level Model (Card et al. 1983). Generating Keystroke-Level Models is straightforward, but takes time, is repetitive, and can be error-prone. This seemed like a good opportunity for automation. The tool, KLM-Qt, listens to windowing system events and translates them into operators for the Keystroke-Level Model.

In order to test out how well KLM-Qt works, an evaluation was done with two phones that are available under open source licenses, the Trolltech Greenphone from Trolltech ASA and the Neo1973 from the OpenMoko project. Keystroke-Level Models were also generated for Apple's iPhone for the purpose of comparison and to see how well the Keystroke-Level Model works in general.

In this paper, we first present the problem area. This is followed with a short background on GOMS, focusing specifically on the Keystroke-Level Model. There is also an introduction to KLM-Qt and how it works. We then take a look at the evaluation done on the phones and discuss the results. We conclude with some suggestions for the KLM-Qt and some recommendations for using the Keystroke-Level Model in evaluation of mobile devices. The questions we try to answer are, can methods that originate from desktop machines be used on mobile devices? Specifically, can the Keystroke-Level Model make the transfer to mobile devices? And finally, how well does KLM-Qt aid in the generation of the Keystroke-Level Models?

Problem Area

Software for mobile devices usually shares the same development methods and languages that are used for desktop software. However, when compared to desktop application, mobile devices have less processing power, less memory and storage space, and need to work on battery power. At the same time, an application should have a good level of usability, being effective, efficient, and pleasant to use. Finding the right balance between the user's needs and the capabilities of the device is a challenge for any interaction designer.

Some methods of evaluation work regardless of the system being used. For example, O'Hara et al. (2006) had users keep diaries of users' experiences with video telephony and their patterns of use. Alsos and Svanæs (2006) used a scenario of a doctor talking to a patient to evaluate methods of interaction between a PDA and a patient display.

Another possible method for evaluation is to model users. A model may not be able to behave exactly like a real user nor be able to give other feedback, but it may still provide relevant data in many areas. Modeling can be done quickly and cheaply since it does not involve experts and users. It also makes it possible to evaluate more ideas. As summarized by Heim (2007), there are two types of models, predictive or descriptive. A descriptive model provides a framework for thinking about user interaction and explains how people interact with dynamic systems. GOMS and the Keystroke-Level Model belong to the area of predictive models. They attempt to approximate how a user will use an interface.

Background

GOMS stand for Goals, Operators, Methods, and Selectors and was introduced in Card, Moran, and Newell's *The Psychology of Human-Computer Interaction* (Card et al. 1983). GOMS's focus is on an expert's error-free use of an interface. An expert is someone who knows the task domain well and knows how to perform all the tasks that need to be done. The expert does not have to look up or ask for guidance in doing any task. Ideally, the expert makes few, if any, mistakes. Therefore, GOMS has no built in mechanism for handling mistakes. Specifically GOMS consists of the user's *goals*, (e.g., replacing all occurrences of "England" with "Britain" in a text editor), the *operators* and *methods* used to achieve the goals, and the *selection rules* for choosing the correct method when multiple methods are available for achieving a goal. The result of a GOMS model is the amount of time it takes for an expert to use the interface.

Since its creation, GOMS has evolved and other versions have spun off from the original version (later called CMN-GOMS for Card, Moran, Newell). Kieras (1988) introduced Natural GOMS Language (NGOMSL), which is a formalized natural language for representing GOMS models. Gary et al. (1993) developed the Critical Path Model or Cognitive Perceptual Motor version of GOMS (CPM-GOMS), which can model things happening concurrently.

The Keystroke-Level Model

Another version that was created early on was the Keystroke-Level Model by Card et al. (1983). Also known as KLM-GOMS or simply KLM, it attempts to simplify GOMS by ignoring the goals and selectors and instead focuses on the keystrokes and mouse presses of the expert user. Even though it is simpler than the other models, it is also very elegant and has been used in a wide variety of situations, from text editors to a database of outer space operations (John and Kieras, 1996, pages 307 and 308).

A KLM consists of a stream of operators. There are five operators that most modern applications are interested in: *P*, *K*, *H*, *R(t)*, and *M*. These operators are presented in table I.

Operator	**Description**	**Time in Seconds**
P	Pointing with a pointing device	1.10
K	Key or button press and release	0.20
H	Move from the mouse to keyboard (or back)	0.40
R(t)	Waiting for the system to become responsive	*t*
M	Mental preparation and thinking time	1.35

Table I: The KLM operators with times determined by Card et al. (1983).

The *P* operator represents pointing with a pointing device to a position on the screen, excluding button presses. The *K* operator represents a pointing device button press and release or a keyboard key press and applies to a single key. This value is very dependent on how fast a user can type. The value for a person typing at 55 words per minute is approximately 0.20 seconds. Points where the hand moves from the keyboard to the pointing device is represented by the *H* operator and is approximately 0.40 seconds for a keyboard and mouse combination. The *M* operator represents time spent mentally preparing to execute an operation, including deciding how to invoke a command, how a command should be terminated, or which options to choose. Thinking time was originally shown by Card et al (1983) to be around 1.35 seconds. Olson and Nilsen (1987) have shown this to be a valid upper bound with their time being around 1.20 seconds.

Finally, there are times when the computer is unresponsive because it is busy doing some processing, and the user must wait before they can interact with the system. This is indicated by the *R(t)* operator where *t* indicates the time in seconds that the user has to wait. Most of the operators can be defined by following the physical movements of the user. The exception is the *M* operator. Card et al. (1983) developed a list of heuristics. The heuristics start by adding in *M*s in all the locations where there could be a potential for mental preparation and then start removing them where they can be eliminated; for example, a click after pointing is done without any hesitation by expert users.

Introducing KLM-Qt

KLM-Qt works on the assumption that a lot of the information about the operators that are part of the keystroke-level model (e.g., the key and mouse presses and mouse movements) can be derived from the events that are delivered to an interface. Other things, such as hand movement, can be inferred by looking at the places where a mouse event is followed by a key event or vice-versa. Therefore, all that is needed is to examine the information when it is delivered to the application. This also works for mobile devices. The way of interacting may be different, but the information is the same. If the information is used to derive a model, it results in a KLM defined for a task the same amount of time it takes to demonstrate the task in the interface. This can save a lot of time and avoid errors instead of creating the model by hand.

The *M* and *R(t)* operators do not provide their information through system events nor can they be easily derived. The *R(t)* operator is difficult to evaluate because a computer switches from busy to being available again many times in a second. On mobile phones *R(t)* is important due to their constrained resources. An acceptable workaround is to add the operator manually by editing the model.

One solution for *M* operators is to look at pauses between operators and determine if an *M* operator should occur. However, as mentioned above, a user evaluating an interface with KLM-Qt may not be an expert user of the interface, so this may result in insertion of more *M* operators than there should be. KLM-Qt can place *M*s by looking at the time between events, but does not do it by default. Another approach would be to automatically apply the heuristics mentioned in the previous section to add the operators, but this requires more information than KLM-Qt has. Like *R(t)*, the safest option is to place the *M* operators manually.

KLM-Qt has two parts, a library and the main KLM-Qt application. The library is linked into the application that is being tested, listens for events, and sends the information back to the KLM-Qt application. The main KLM-Qt application providing an overview of all currently running applications that can be evaluated, allowing a user to select which application they will evaluate, and controlling the recording.

It is also possible to make alterations to the models after they are generated. Notes can be added for each individual operator. Operators can also be removed or added. This is typically done to add *M* or *R(t)* operators. When instructed to, KLM-Qt will go through the model and look for breaks of time that are greater than a threshold (by default 1.20 seconds, but adjustable in the preferences) and then inserts an *M* operator, with a note to indicate that it was placed there by KLM-Qt. This simplifies the process of inserting *M*s and may work in cases where a real expert is using the system.

KLM-Qt is not the only attempt to automate GOMS model generation. Examples include: QGOMS (Beard et al. 1996), CATCHI (Baumeister et al. 2000), USAGE (Byrne et al. 1994), GLEAN (Kieras et al. 1995), CRITIQUE (Hudson et al. 1999), Apex (John et al. 2002), and CogTool (John and Salvucci 2004). CRITIQUE is the most similar in its idea to KLM-Qt, but it uses a toolkit that is only available on Sun Workstations and is not available to the public.

The Evaluation

The evaluation served as a test of KLM-Qt, and to see how KLM works on mobile phones. If the models that were generated could be used to draw conclusions about the interfaces or to find issues that should be addressed in the evaluation, it would show that the methods could be applied to mobile phones. We could judge the effectiveness of KLM-Qt by the quality of the models it generated and the amount of work required using KLM-Qt versus doing the models by hand.

The phones evaluated were the Trolltech Greenphone, the Neo1973 (also known as the OpenMoko phone), and the Apple iPhone. The phones are depicted in figure 1. The iPhone uses its own user interface. The Greenphone and Neo1973 both run Qtopia Phone Edition, an open source platform for mobile phones that is tailored to handle the specific features of each phone. For example, the Neo1973 uses a touchscreen and stylus for input and has a special onscreen keyboard, while the Greenphone relies more on the keypad with a predictive T9-like input method.

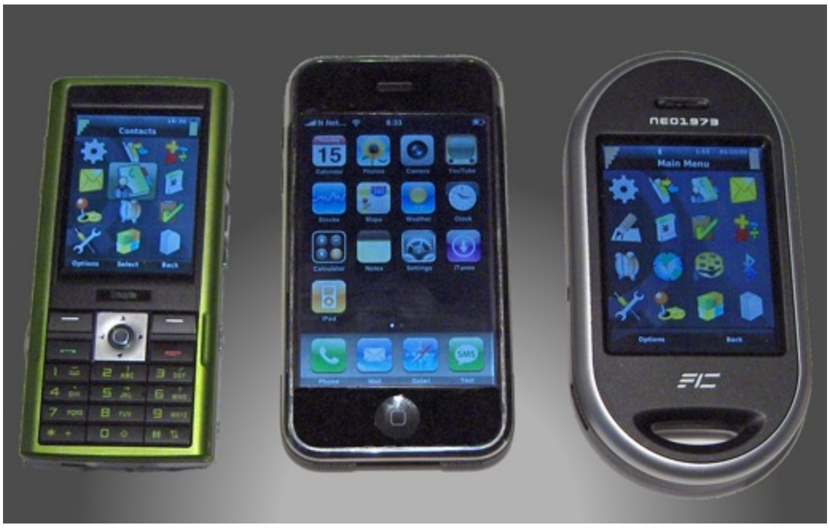

Figure 1: Trolltech Greenphone from Trolltech ASA, iPhone from Apple, and Neo1973 from FIC.

The evaluation was carried out as part of a group project for a class at the University of Oslo. The Keystroke-Level Models for the Greenphone and Neo1973 were constructed using KLM-Qt using the methods explained in the above section since KLM-Qt can work with Qtopia and manually for the iPhone since KLM-Qt cannot be used directly on it. The tasks for the phones are presented in figure 2.

For the iPhone, the models were all generated by hand from blank KLM-Qt documents. Most of the tasks could be accounted with *K* operators that originated from a mouse. For the on-screen keyboard, *K* operators from the keyboard were used. To simulate the slide, a combination of $K_{press}PK_{release}$ was used to indicate the press, slide, and release. Times were not measured precisely for this, but the slide usually takes about a second. No new timings for the operators were taken; instead the original timings from Card et al. (1983) were used. None of the tasks that were tested used “multi-touch” gestures.

Since the Greenphone used the keypad as its primary method of input and the Neo1973 primarily used a touch screen, some way of representing the how text was entered was needed. The solution was to introduce a new operator, *I*, which would indicate an event from an input method. It would represent both the composing of the text for entry and the final text that is committed. It would have been possible to combine multiple *I* operators into one jumbo operator, but that would have hidden information about what keys were pressed or how many steps were executed before the input was confirmed, therefore each “composing” operator is included.

- Remove the keypad lock.
- Add a contact, “Marius” with a mobile number.
- Send an SMS with text “Hi” to a new number.
- Answer an incoming call.
- End a telephone call.
- Add number from the previous call to a new contact, “Jo.”
- Call the “Marius” contact from the address book.
- Change the name of the “Jo” contact to include a last name.
- Make the telephone silent.
- Answer the SMS sent to the phone.
- Set an alarm for tomorrow at 10:00
- Add a picture to a contact from the library of pictures included in the phone.
- Dial a number
- Call the most recently called contact.
- Add a contact as a speed dial.
- Remove the picture from a contact.
- Add a “Meeting” at 10:00 the next day in the calendar.
- Check missed calls.
- Delete a contact.
- Activate the keypad lock.

Figure 2: The tasks performed for the second evaluation of the Greenphone, iPhone, and Neo1973.

Results

Testing the telephone functionality was revealing. It seemed that the majority of the telephone functions worked as expected on all the phones. Dialing a contact, dialing a phone number, or reading an SMS worked similarly to how it works on other phones. Writing SMS messages was a little different as it involved dealing with the messages program that appeared to have been designed for more of an email-based way of working. It also took extra steps to confirm the person to that was supposed to be receiving the SMS if you were starting from the messages program.

Though Neo1973 and the Greenphone used the same software and appeared that time had been spent making Qtopia more tuned for the Neo1973 with a special input method and the Neo1973 gave an experience that made it seem to run faster, the times for doing things were longer than on the Greenphone. Things like adding contacts, writing SMS messages, or adding a calendar event. This sometimes took 10 seconds longer than doing it on the Greenphone. Even doing such things as selecting a contact and calling it took about 40% longer. Part of this was due to the fact that inputting text was more cumbersome than on the Greenphone Details of times to complete a task and the number of keystrokes is shown in table II. Though there is more to the models than just counting keystrokes, presenting both values here can be useful summary for comparing among the phones. Keep in mind that the iPhone models are generated by hand, so the times given are not exact.

Tasks	Greenphone	iPhone	Neo1973
Add a contact	21.022 sec/22 keystrokes	20.150 sec/20 keystrokes	40.018 sec/26 keystrokes
Send an SMS	14.912 sec/14 keystrokes	16.100 sec/10 keystrokes	18.025 sec/11 keystrokes
Add number from a call	22.240 sec/12 keystrokes	16.600 sec/8 keystrokes	18.868 sec/13 keystrokes
Call a contact	4.921 sec/4 keystrokes	10.600 sec/4 keystrokes	7.988 sec/5 keystrokes
Change contact name	22.326 sec/28 keystrokes	24.100 sec/16 keystrokes	41.688 sec/16 keystrokes
Answer SMS	23.379 sec/18 keystrokes	11.650 sec/8 keystrokes	17.275 sec/11 keystrokes
Set an alarm	5.677 sec/6 keystrokes	15.850 sec/5 keystrokes	1.000 sec/3 keystrokes
Add picture to contact	20.326 sec/14 keystrokes	21.600 sec/10 keystrokes	43.364 sec/17 keystrokes
Dial a number	6.915 sec/9 keystrokes	6.200/10 keystrokes	8.556 sec/10 keystrokes
Call most recent	4.958 sec/2 keystrokes	5.300 sec/2 keystrokes	11.483 sec/5 keystrokes
Add contact to speed dial	9.489 sec/13 keystrokes	13.450 sec/5 keystrokes	10.403 sec/5 keystrokes
Remove contact picture	25.290 sec/20 keystrokes	13.400 sec/7 keystrokes	17.572 sec/8 keystrokes
Add a meeting	20.980 sec/23 keystrokes	27.500 sec/16 keystrokes	37.419 sec/21 keystrokes
Check missed calls	3.609 sec/2 keystrokes	5.300 sec/2 keystrokes	5.576 sec/3 keystrokes
Delete a contact	9.832 sec/7 keystrokes	17.400 sec/7 keystrokes	9.003 sec/5 keystrokes

Table II: Times and number of keystrokes for select tasks for the phones under evaluation.

Comparing the Keystroke-Level Models that were generated from the iPhone to the Greenphone or Neo1973 shows a different philosophy on how tasks were approached. As can be seen in table II, the iPhone won in number of keystrokes for almost all of the tasks. However, there were some things that worked better on the Greenphone. For example, the Greenphone could assign a phone number to speed dial. This meant that hitting a single button, pressing the button would dial that number. The iPhone and the Neo1973 maintained a list of favorites. After setting the speed dial, it was very fast to reach this contact on the Greenphone, and a bit longer on the others, as it meant scanning a list. However, setting a number as a favorite required fewer keystrokes and could be completed faster on the iPhone than setting up the speed dial on the Greenphone. Another task where the Greenphone came out ahead of the iPhone was in setting times. On the Greenphone, it is possible to just type in the time; on the iPhone, setting a time involves sliding rollers to get a correct time. In this case, setting an alarm or creating a meeting was a bit faster on the Greenphone. However, setting times on the iPhone is less error-prone than the Neo1973.

Certain tasks were simple on all the phones. For example, answering a call, hanging up on a call, or locking the phone, all had the same amount of complexity—one button press. Unlocking the phones was similarly straightforward, either a slide or two key presses. The iPhone had a major departure from the other two phones with regard to making it silent. The Greenphone and Neo1973 required going into the settings to activate the "silent" profile, while the iPhone had a switch on the side to flip. Obviously it was quicker to switch the iPhone into and out of silent than navigate through to the profile area and activate a profile. The Greenphone had a slight advantage on pure phone functionality, owing to the fact that it possessed a keypad for dedicated phone tasks. This made it easy to dial a number, check missed calls, or call recent contacts, while on the other phones, required navigation into the specific "phone" area of the software.

Discussion

The Keystroke-Level Model and Mobile Devices

Even though the Keystroke-Level Model was designed for desktop systems, it appears to produce models that are useful in evaluating mobile devices and comparing them. The models help to capture the interaction that goes on with using a mobile device and can reveal how well or poorly a task is supported on a device. Having the models also provides a way of comparing the interaction between two devices and seeing which one allows the quickest way of doing things. In this situation, where two devices used the same software, it was also possible to see which design interactions worked the best. This information could be used to guide choices in what hardware to select.

While the Keystroke-Level Model can transition over to mobile devices in the abstract sense that the operators map over, it is a bit more difficult to say if that is the case with the actual values that were originally determined by Card et al. (1983). Pressing buttons seems to transfer well to the *K* operator, even when they are software buttons on the screen pressed by a stylus. The original timing from

Card et al. (1983) of 0.200 for a person typing 50 words a minute seems to be the average time for pressing buttons on the phones as well. However, some operators do not make this clear transfer. For example, the *P* operator is clearly not necessary on phones that do not have a touch screen, but those that do make it a bit difficult to apply consistently. On a desktop machine, the mouse dictates the *P* operator, since it tracks the movement of the cursor. This cursor movement is not straightforward on a mobile device. The hand does very little movement and it is possible to access different points on the screen with different fingers, which makes the *P* much faster in those occasions. Perhaps using Fitts' Law tailored to each screen could provide better values.

Another example is the *H* operator. It might mean completely different things on different devices. On the iPhone, for example, it could be argued that there is an *H* operator that happens when clicking on a field that requires text input and the on-screen keyboard comes up. At these points, users may change to using their thumbs for typing instead of the pointer finger. This is much different on the Neo1973 because the user is always using the same pointing device and there is no perceived difference. The *H* does not seem to exist on devices like the Greenphone where the primary mode of interaction is through a keypad, and text input and navigation are simply a matter of pressing different buttons.

The *M* operator can be another issue that may need to be addressed. Myung (2004) claims that the *M* operator for a mobile phone should be around 570 milliseconds, which is much shorter than the traditional value of 1350 milliseconds. No data was collected to see if this was in fact true. Holleis (2007) contends that this value is for a very specialized area of use and asserts that the normal value should be used. Besides trying to find a new time for an *M* operator, developing a new set of heuristics that focus more on mobile devices (e.g., taking into account confirming input) could also be useful.

Even though the *K* operator works well in most cases of pressing buttons, it does not transfer well in all cases. In particular when using predictive input methods, it can require the person to confirm that the text that is entered is correct. Myung (2004) found that this was the case for Korean input methods. More work needs to be invested in created a better operator to encapsulate typing using an input method. The *I* operator creating introduced for using with KLM-Qt is not good enough to represent different input methods.

When the evaluation was being run, the decision was made to have the evaluators sitting at the table and having the phone in their hand. This does not cover all possible situations of how and when someone uses a mobile phone, but it covers an adequate subset. Holleis et al., (2007) proposed many new types of operators that could be used, such as a macro and micro attention shifts (S_{macro} and S_{micro}), distraction (*X*), an arbitrary action (*A(t)*), broad gestures (*G*), finger movement (*F*), and initial act (*I*)[1]. These suggested operators were not used in our models, partly because of point in time when the information was discovered and partly because of the fact that it adds complexity to the models. We were still able to create models that provide useful information about the number of operators it requires for completing a task and how long it takes. Adding in the operators from Holleis et al. would not invalidate the data, but instead provide a deeper understanding of what is going on.

However, one thing to keep in mind is that the models were generated differently between the iPhone and the other phones and it could be that the default times that used for the *P* and *M* on the iPhone add more time to the task than it would really take. Even using these values that are most likely larger than their actual values, the iPhone is able to accomplish most of its tasks faster than on the Greenphone or the Neo1974 and with less keystrokes. Of the three phones tested, the iPhone models showed the phone to be the most efficient phone. This matches informal opinion about experiences with the devices. Having a better way of capturing input methods may make it possible to try alternate methods and see how well they work on a device before time is spent implementing it.

Finally, it is worth considering the limitations in the Keystroke-Level Model and GOMS itself. The Keystroke-Level is limited to modeling the error-free execution of expert users. Novice users may have a completely different way of interaction or the interfaces themselves could be very error-prone. KLM-Qt cannot provide any information about this. Therefore, it is a good idea to use other methods, like usability testing, to find out information about these areas, or to combine them to get a better overall picture.

KLM-Qt and Mobile Devices

KLM-Qt works for generating an initial model of a task. During the evaluation, it was possible create connect a Greenphone or the Neo1973 and start recording a model. Once the model was complete, it was just a matter of going back and adding extra notes for the operators that should be documented. It was also easy to add or remove operators if it was necessary. It was much faster using KLM-Qt to generate the model by recording than generating the model by hand. For our final set of models, we were able to start and complete the whole process in around 20 minutes. This was compared to spending around two hours when doing the models by hand. So, the goal of saving time was clearly realized.

When creating the models by hand, going through the task a couple of times was necessary in order to get each step. After initial models were written, it usually required going back over the models a couple of times to add in bits that were missing (like *P* and *M* operators). It would have been very helpful to have

[1] Not to be confused with the *I* operator mentioned in the previous paragraph.

had the ability to have captured events from the iPhone to have generated the models automatically.

It was even useful to have KLM-Qt available for generating the models by hand. It was able to provide some specialization that is not built in to most spreadsheet programs. However, it would have been helpful to be able to copy and paste groups of operators in the model in order to have access to commonly repeated operations (such as sliding a bar on the iPhone). Of course, KLM-Qt was primarily targeted as a recorder and not a full-fledged editor.

Recording the events from the device worked well in general, but it could have been better for capturing the input methods. The new operator, *I*, helps solve the problem of including the information in the model, but it also hides information because it is impossible to know from the event what type of input method generated the event. In some cases, this is problematic when wanting to compare different input methods. Some input methods give a good indication of which one is being used. For example, when looking at the model and seeing that the "composing text" changing radically as each letter is input, it is a good indication that it is some type of predictive method. This change in text does not happen when using an on-screen keyboard.

As mentioned above, the *P* operator is a bit tricky to track on the mobile device. However, KLM-Qt would be in a could position to attempt to solve this issue since it has access to the events. It could look for events where there is "mouse" release event and record the position. It could then record the new position on the next press and generate a *P* event given the time and distance between the two events. This has a side effect of swallowing *M* events that happen between the two events though, but if the time is longer than that for a normal *M* an *MP* could be inserted instead.

The Neo1973 was the most troublesome of the phones. It did generate models that were usable, but the models contained a lot of noise because the touchscreen seemed to detect slight movements, which would show up as *P* operators between a mouse press and release ($K_{press}PK_{release}$). The solution was to substitute these sequences with one *K* that contained all these operators. Another issue seemed to be an implementation detail of Qtopia on the Neo1973, the mouse presses would generate other input events, but both of the events would be reported, so the models would contain the mouse's *K* followed by an *I* or a keyboard *K*. Some work was done to filter out spurious *H*s that would be generated this way, but there were still extra operators. Taking the time of the *K* and the *I* would generate times that are similar for *K* in table I.

The *M* operator is another place where the situation could be improved. Having the ability to look for potential *M* operators in spaces where there is a pause between operators is useful and it turns up many *M* operators at locations where it might be considered this to happen, such as preparing to input text or confirming a selection. However, it is also possible to detect false positives. There are points where the user has to wait for the device to load a dialog and this may be more than a few seconds. This is over the limit for inserting an *M* operator and consequently, an *M* gets inserted into the model. This is part of the reason why finding *M*s this way is not the default. On the other hand, this can be used as a way of discovering many of the points where someone has to wait for the device because it usually results in *M* operators that are three to four times the standard duration. It is possible to replace these with an *M* and *R* or simply an *R*.

As mentioned above, the Keystroke-Level Models that were generated from these evaluations did not deal with certain situations such as getting the device out of a purse or jacket pocket to answer it. Part of the reason for this was that it was clumsy to do with the way the phones communicated with KLM-Qt. The phones use a TCP/IP connection and the Greenphone and Neo1973 only are able to provide a connection through a USB cable. This cable gets in the way stowing the phone and could potentially become disconnected. In the future, as more devices get the ability to do wireless Internet access, it may very well be possible to use KLM-Qt to model these extra situations as well. It would be interesting to see how KLM-Qt or simply the Keystroke-Level Model would deal with situations that are beyond the sitting at a table.

Conclusion

The Keystroke-Level Model works for evaluating the interfaces on the mobile phones that we tested on. The KLM-Qt tool produced models quickly on the Greenphone and Neo1973 by recording the operating system events. It also was helpfully in creating the models for the iPhone by providing more support than what would be found in a generic spreadsheet application.

There are some issues with both KLM-Qt and the Keystroke-Level Model that need to be addressed to make them provide better models. Some sort of adaptation to handle the various input methods that are available on the mobile phones would be useful. Having a better time for the *P* and *M* operators could also help make more accurate models when doing things by hand. Perhaps taking some of the operators from Holleis et al. (2007) would be useful in handling some of these challenges without making new models overly complex.

KLM-Qt would work great for desktop applications, but needs some adjustments for the events it sees on the mobile phone. It could be more helpful in determining when the UI blocks and translating that into *R(t)* events that could flag problem points a design. KLM-Qt could also try harder to determine when a *P* and an *M* operator occur as well. This might mean giving hints to the model to help it apply the heuristics originally made by Card et al. (1983).

More research needs to be done to address these issues. However, it appears that the Keystroke-Level Model is a valid way of evaluating mobile devices and can be a new tool in building better user interfaces.

References

Alsos, O. A. and Svanæs, D. (2006). “Interaction techniques for using handhelds and PCs together in a clinical setting,” in *NordiCHI ’06: Proceedings of the 4th Nordic conference on Human-computer interaction*, pages 125–134, New York, NY, USA.

Beard, D. V., Smith, D. K., and Denelsbeck, K. M. (1996). “QGOMS: a direct-manipulation tool for simple GOMS models,” in *CHI ’96: Conference companion on Human factors in computing systems*, pages 25–26, New York, NY, USA.

Byrne, M. D., Wood, S. D., Foley, J. D., Kieras, D. E., and Sukaviriya, P. N. (1994). “Automating interface evaluation,” in *CHI ’94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 232–237, New York, NY, USA.

Card, S. K., Newell, A., and Moran, T. P. (1983). *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA

Gray, W. D., John, B. E., and Atwood, M. E. (1993). “Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world task performance,” *Human-Computer Interaction*, 8(3):237–309.

Heim, S. (2007). *The Resonant Interface: HCI Foundations for Interaction Design*. Addison Wesley, Boston, MA, USA

Holleis, P., Otto, F., Hussmann, H., and Schmidt, A. (2007). “Keystroke-level model for advanced mobile phone interaction,” in *CHI ’07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1505–1514, New York, NY, USA.

Hudson, S. E., John, B. E., Knudsen, K., and Byrne, M. D. (1999). “A tool for creating predictive performance models from user interface demonstrations,” in *UIST ’99: Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 93–102, New York, NY, USA.

John, B. E. and Kieras, D. E. (1996). “Using GOMS for user interface design and evaluation: which technique?” *ACM Trans. Comput.-Hum. Interact.*, 3(4):287–319.

John, B. E., Prevas, K., Salvucci, D. D., and Koedinger, K. (2004). “Predictive human performance modeling made easy,” in *CHI ’04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 455–462, New York, NY, USA.

John, B., Vera, A., Matessa, M., Freed, M., and Remington, R. (2002). “Automating CPM-GOMS,” in *CHI ’02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 147–154, New York, NY, USA.

Kieras, D. E. (1988). “Towards a practical GOMS model methodology for user interface design,” in The Handbook of Human-Computer Interaction. North-Holland, Amsterdam, 135–158.

Kieras, D. E., Wood, S. D., Abotel, K., and Hornof, A. (1995). “GLEAN: a computer-based tool for rapid GOMS model usability evaluation of user interface designs,” in *UIST ’95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 91–100, New York, NY, USA.

Myung, R. (2004). “Keystroke-level analysis of Korean text entry methods on mobile phones,” *International Journal of Human-Computer Studies*, 60(5-6):545–563.

O’Hara, K., Black, A., and Lipson, M. (2006). “Everyday practices with mobile video telephony,” in *CHI ’06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 871–880, New York, NY, USA.

Olson, J. R. and Nilsen, E. (1987). “Analysis of the cognition involved in spreadsheet software interaction,” *Human-Computer Interaction*, 3(4):309–349.