An Event Optimization Scheme for the Multi-window System on Mobile Phones

Jianming Wu    Toshiaki Uemukai    Kazunori Matsumoto    Yasuhiro Takishima    Fumiaki Sugaya
KDDI R&D Laboratories Inc.
2-1-15 Ohara, Fujimino, Saitama, Japan
Tel: +81-49-278-7422
{ji-wu, to-uemukai, matsu, takisima, fsugaya}@kddilabs.jp

ABSTRACT
In this paper, we propose an event optimization scheme for the multi-window system on mobile phones. A four-level event optimization scheme is adopted to reduce the response time. It includes (a) spatial filter, (b) temporal filter, (c) same-kind event chunk, (d) different-kind event chunk. An event optimization agent checks the CPU load constantly, and it applies the four-level optimization process dynamically according to the CPU load. Therefore, it is possible to provide a flexible and efficient mechanism for constructing a rich GUI interface and collaborating with new forms of user input methods on mobile phones. The result of experiment shows that about 60% of the runtime consumption is reduced by the proposed system.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. – Graphical user interfaces, Interaction styles.

General terms: Performance

Keywords: Event Optimization, Bluetooth, Filter, Chunk

INTRODUCTION
As the 3G mobile networks and the mobile terminals evolve, many advanced applications are required to work on the handsets simultaneously in terms of user operability. Therefore, we have developed a multi-window GUI system for mobile phones with BREW, called “K-tai win” [1]. On the other hand, current researches on external input devices via Bluetooth shows a strong focus on entertainment fields for providing straightforward interaction mechanism. Luinge et al. have developed a movement detecting sensor, which is able to detect a movement of the mobile device [2]. Therefore, we proposed several K-tai win application prototypes utilizing external input devices as follows:

1. Figure 1 shows that a user switches the desktop and the application window by mouse. A user can also change or customize his/her favorite actions. For example, a user can associate the right double-click event with switching a skin.
2. Figure 2 shows a web map service application by an acceleration sensor. A user can navigate the map by rolling the hand. When a user rotates the mobile phone it can also switch between the vertical or horizontal display-mode.

Figure 1: Switch Desktop/Window   Figure 2: Map Service

Unfortunately, such a straightforward interaction mechanism causes massive events from external devices. While due to the limitation of CPU power and especially the poor performance of graphics system, the response of the mobile phone would slow down drastically when many events are stored in the event queue to be processed. QT, a cross-platform application development framework, provides an event filter mechanism for applications [3], but it requires programmers to implement the filter logic in each application and is not suitable for applying the optimization process according to the system status.

PROPOSED SCHEME
In order to simplify the event processing, we consider an approach to decrease redundant events in the event queue. However, as optimization process may result in sacrificing the accuracy of user’s operations, we propose a flexible scheme for applying dynamical optimization process. An event optimization agent is applied to check the CPU load constantly. When it finds the system is in a busy status, the optimization process will be executed in sequence according to the CPU load to avoid unnecessary accuracy loss of user’s operations. Four-level optimization process is proposed as below, which will abandon the events according to the threshold and chunk several predefined event sequences. Generally, as a filter process being executed in advance could simplify a chuck process at the very beginning of optimization, a filter process should be applied before a chunk process.

(a) Spatial filter
An event will be abandoned if the manipulation amount (such as moving a mouse) is less than a certain threshold.

(b) Temporal filter
An event will be abandoned if the runtime interval amount is less than a certain threshold.

(c) Same-kind event chunk
The same-kind event sequence will be simplified if it meets a certain predefined condition. For example, a certain count of continuous mouse moving events will be combined into a single event, whereas the moving distance is the same as without chunk process.

(d) Different-kind event chunk
The different-kind event sequence will be simplified if it meets a certain predefined condition. For example, the continuous mouse moving events and direction key pressing events will be combined into a single event, whereas the moving distance is the same as without chunk process.

When the CPU load is moderate, only (a) will be applied. However, when the CPU load increases, (a) and (b), or (a) and (b) and (c) will be applied. When the CPU load is higher than the maximum threshold, all optimization processes will be applied.

Figure 3 shows the hierarchical architecture of K-tai win to apply the proposed scheme. Bluetooth mouse and Bluetooth acceleration sensor are proposed as the extensible external input devices in the device layer. Event optimization layer interacts with the device layer, to filter and chunk the redundant events. Then, the optimized events are passed to the event driven layer, which keeps an action recognition pattern DB of a set of estimated actions for instance “Left”, “Right”, “Up”, “Down”, “Drag”, “Drop”, “Roll”, “Shake”. Finally, GUI Layer provides graphic user interface functions for user applications according to the dispatched event from the event driven layer.

ExPERIMENT
We conducted an initial experiment to compare the performances between the proposed system and the traditional system. In this experiment, we designed an auto-run application based on “K-tai win” to emulate the events from the device layer. After recording users’ actions during operating “K-tai win”, we defined overall 270 actions to be executed, such as WINDOW_SWITCH, MOUSE_MOVE, DRAG&DROP, DESKTOP_SWITCH, BUTTON_CLICK, TEXT_INPUT, and etc. Figure 4 shows the variation of the runtime consumption by each 10 events. It showed that during the event sequence 200–240, when the system is in a heavy CPU load status, the runtime consumption was reduced drastically by applying the 4-level optimization process. The red text showed the approximate level of applied optimization process in that area, and from the system log we found that process (c) reduced most of the response time. Furthermore, on average, all the events were conducted in 19.4 seconds with the traditional system, whereas they needed only 7.8 seconds with the proposed system. Though in this auto-run experiment, there was no obvious accuracy loss of user’s operations for the sake of simplicity, we need to refine the threshold of each optimization process by observing more users’ operations.

CONCLUSION & FUTURE WORK
We have developed a multi-window mobile phone platform. It applied a four-level optimization process dynamically according to the CPU load. By utilizing the proposed scheme, it is possible to realize a simple and natural interaction mechanism on mobile phones. The experiment showed that about 60% of the runtime consumption was reduced. In the next phase, we plan to add more predefined optimization conditions by observing the users’ operations and to evaluate the effectiveness of this system.

REFERENCES