TouchUI: Event from hardware to software

25 Jan 2009

Introduction

Touchscreen technology is widely used in PDA, smartphone, ATM, information kiosk, and a multitude of other devices. Though the concept is not new, the technology was shaken up with the introduction of the Apple iPhone, and the reverberations were felt with touch devices manufacturers and software vendors. In this article we briefly focus on touch UI technology and how the information moves from hardware to our application via the operating system. Touch UI technology has increased the usability of applications and has proved to be more natural to use than its keyboard counterpart. We'll demonstrate how a simple application can receive the pointer event and how it uses it. At the end, we've included an example of a naught and crosses game (provided by Nokia Developer) that has been touch-enabled.

Touch hardware

Touch input is detected by a touch sensor. There are two major types of sensors: resistive and capacitive. (Other technologies are also available, but are beyond the scope of this document.)

Resistive technology

Resistive touch is the mainstream touch technology used in many monitors and displays. This technology is cost effective but has some limitations. Resistive touch technology is based on measuring the changes in resistance as different parts of the touchscreen are pressed. When a stylus or finger is used, it causes a conductive top layer to bend downwards, making electrical contact with another conductive layer. A voltage is applied to the top layer, which varies according to position; from this voltage variation, the touch position can be measured. The Nokia 5800 XpressMusic device uses touch technology.

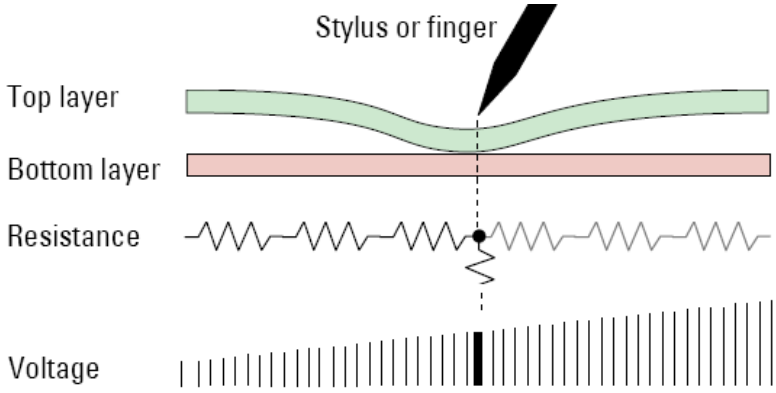

Figure 1: Resistive touch technology

Benefits of resistive technology

- Any probe can be used for touching the UI
- Low cost
- High resolution
- Pressure sensitivity

Capacitive touch technology

Control electronics for capacitive touch are more complex than resistive technology and hence more expensive. A very simple touch UI such as a single-touch switch can be built with a single sense electrode, as illustrated in Figure 2. A measurement circuit measures the capacitances of the electrode by charging and discharging the capacitor formed by the sense with surroundings. iPhones use capacitive touch technology.

(C) Copyright Nokia 2013. All rights reserved.

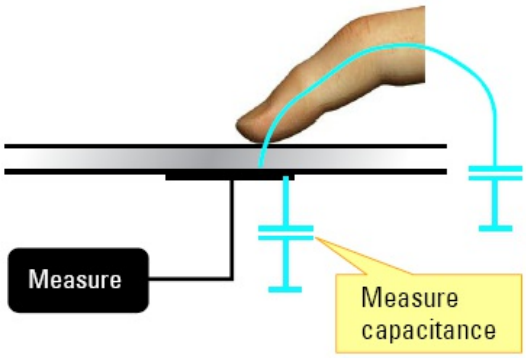

Figure 2: Capacitive touch technology

Benefits of capacitive technology

- Easy scrolling (no resistance to finger movement)
- Design flexibility; touch panel can be implemented on curved surface, too
- Multi-touch can be implemented easily

Touch information flow

A touch controller filters out erroneous touches as well as determines which part of the UI was touched by calculating the coordinates. A window server event is generated by the touch event that is routed to the UI framework, which is further processed by the application. Figure 3 shows the information flow from hardware to application control.

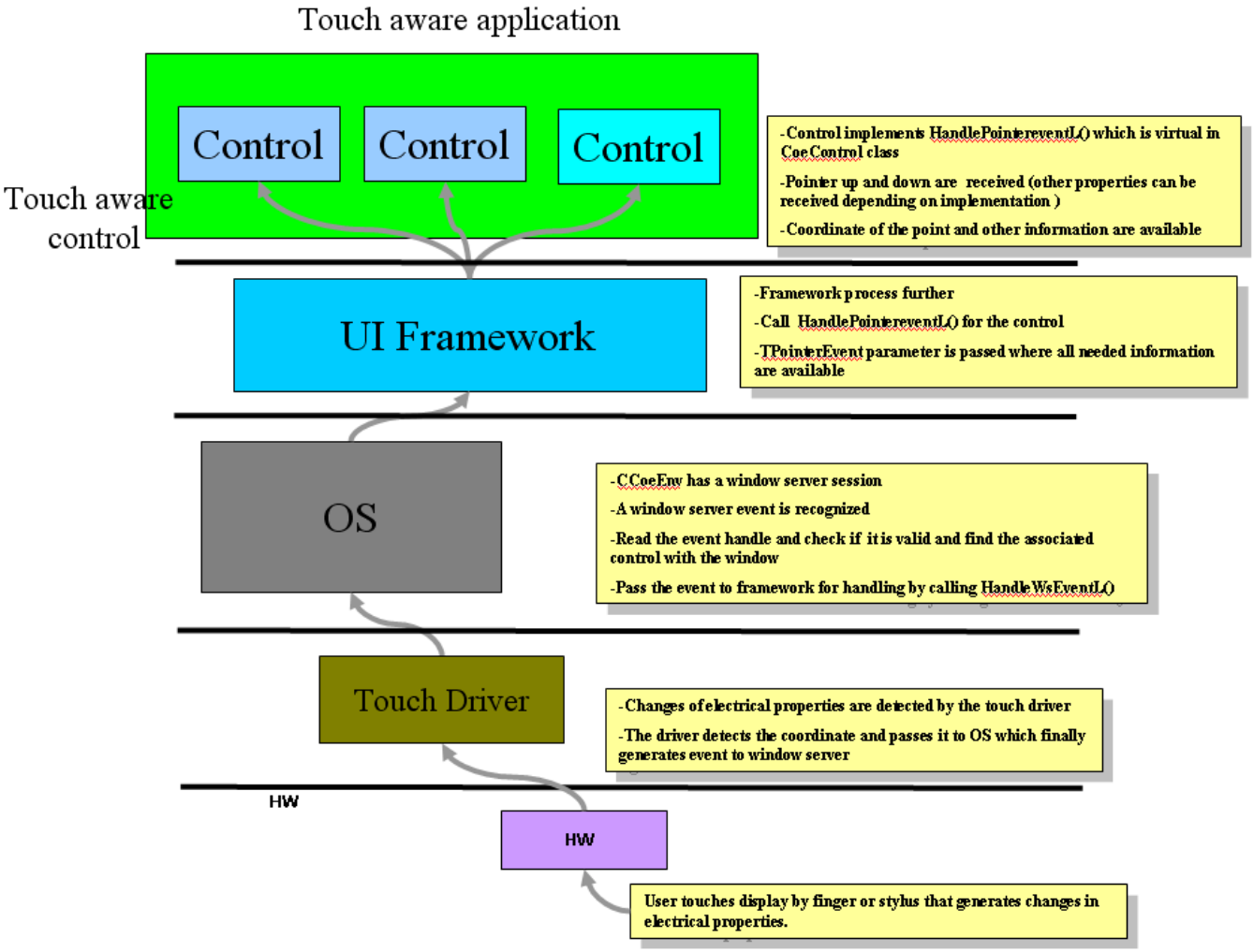

Figure 3: Information from hardware to `CCoeControl`

Pointer event received by `CCoeControl`

When the touch UI is pressed by a stylus or finger it generates a completion of a wait loop, which in turn calls the `RunL` of `CCoeEnv`. The window that generated the event is identified by the window server, and the event is passed to the control by calling function `HandlePointerEventL`. `HandlePointerEventL` is a virtual function that must be reimplemented by the control if we want to receive the pointer event. `TPointerEvent` is the only parameter that is passed to the control. In our naught and cross game we are interested only in the coordinate when the user presses on the screen. We

(C) Copyright Nokia 2013. All rights reserved.

need to draw only when we receive a button down (in our example), and hence we ignore the button up events. Here is the pointer event implementation for the nought and cross game.

In header file

```
class CNoughtsAndCrossesContainer : public CCoeControl,
    public CNoughtsAndCrossesEngine::MObserver
    {
// some other stuff
   void HandlePointerEventL(const TPointerEvent& aPointerEvent);
// some other stuff
};
```

In implementation file

```
Void CNoughtsAndCrossesContainer::HandlePointerEventL(const TPointerEvent& aPointerEvent)
    {
    if(aPointerEvent.iType != TPointerEvent::EButton1Down)
        {
        return;
        }
    TRect rect2 = iView.ClientRect();
    TPoint pt1(0, (aPointerEvent.iParentPosition.iY-aPointerEvent.iPosition.iY));
    TPoint pt2 = rect2.iBr;
    TRect drawablerect(pt1, pt2);
    TInt xwidth = (TInt)(rect2.Width()/BOARD_SIZE);
    TInt yheight = (TInt)(rect2.Height()/BOARD_SIZE);
    TPoint temp = pt1;
        if(drawablerect.Contains(aPointerEvent.iParentPosition))
        {
        TRect allboard[BOARD_SIZE][BOARD_SIZE];
        for(TInt i = 0; i < BOARD_SIZE; i++)
         {
         for(TInt j = 0; j < BOARD_SIZE; j++)
          {
          TSize siz(xwidth,yheight);
          allboard[i][j].SetRect(temp,siz);
              if(allboard[i][j].Contains(aPointerEvent.iParentPosition))
           {
           iCursorRow = i;
           iCursorColumn = j;
           TKeyEvent aKeyEvent;
           aKeyEvent.iCode = EKeyEnter;
           TEventCode aType = EEventKey;
           OfferKeyEventL(aKeyEvent,aType);
           return;
           }
          temp.SetXY(temp.iX+xwidth, temp.iY);
          }
         temp.SetXY(pt1.iX , pt1.iY+(i+1)*yheight);
         }
        }
    return;
    }
```

Example source code

The modified application can be found at notandcross.zip.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.