Maemo Diablo The GUI Components of maemo

Training Material

February 9, 2009

Contents

Chapter 1

The GUI Components of maemo

This chapter introduces the basics of the graphical user interface and GUI programming components used in the maemo platform, different views of Hildon desktop, `event-loop` -based GUI model and signals.

1.1 Decomposition of a simple GUI-program

Comparing the simple command-line-program (in previous chapter) and simple GUI-program running on maemo platform reveals the extensive usage of different libraries for GUI programs. Maemo platform provides many APIs to handle the GUI generation, resource management and application integration to the application framework. These APIs also hide the complexity of the `X library`, which normal maemo GUI programs don't have to care about, although it is used internally by the GUI libraries provided. A decomposition view of a simple GUI program running on the maemo platform is seen on Figure 1.1.

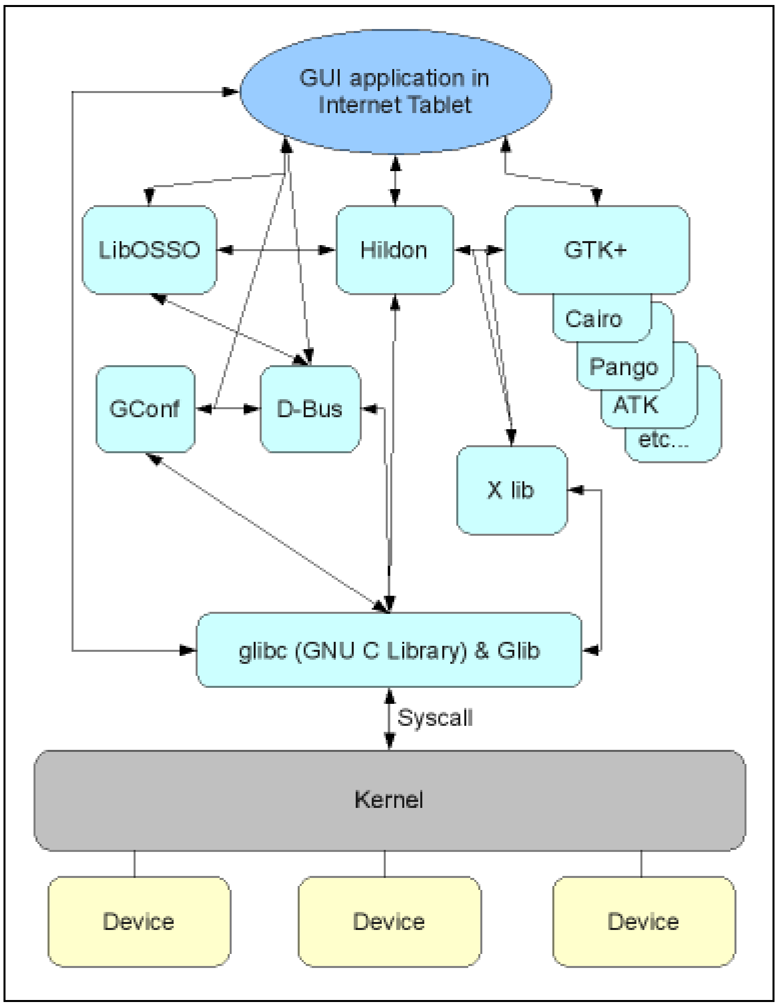

Figure 1.1: Decomposition of a simple GUI-program

1.2 The GUI components

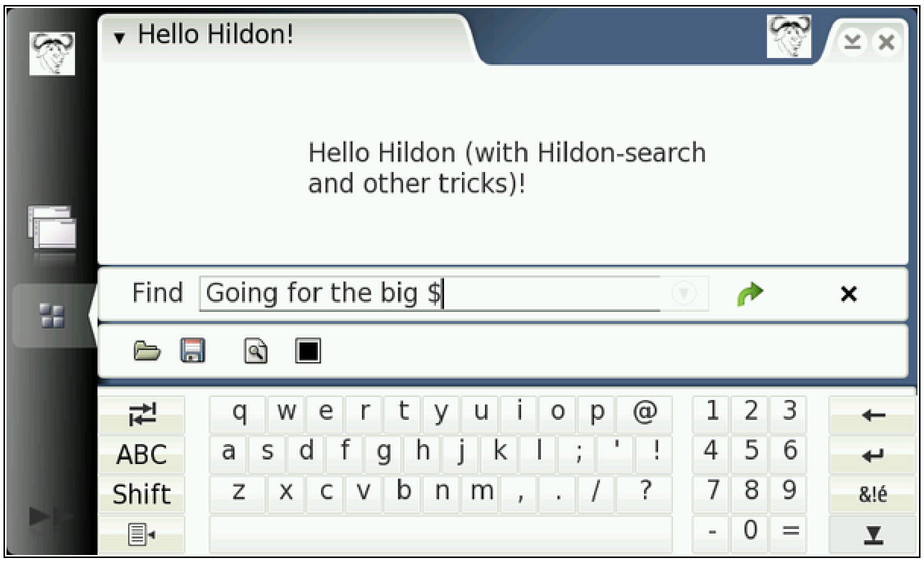

Figure 1.2: Internet Tablet graphical user interface

The graphical user interface application framework of maemo is called Hildon. It is based on the technologies that GNOME framework (used on many desktop Linuxes) is built on, most importantly the GTK+. Hildon contains several enhancements to GTK+ making it more suitable to use on the Internet Tablets: Hildon widgets, speed-improved Sapwood theme engine, image server, task navigator, control panel, status bar, touch screen input method, stylus support and a window management on a high-pixels-per-inch screen.

The GUI programming APIs for maemo are based on GTK+ widgets and Hildon extensions on top of it. Most GTK+ widgets and methods work in the Hildon environment without modification, the most important exception being the application main window widget which is replaced by Hildon window.

Only one application is visible at the time, as the application's window fills the whole `application area`. Switching between running applications is done from `task navigator`. Task navigator also includes menus for launching new applications. Statusbar (titlebar) area includes application menu and also buttons to close and minimise the running application. The application has only one main menu, with submenus spawning horizontally to the right, so the application developer must plan menus carefully as the space for menus is limited. The on-screen virtual keyboard is launched automatically when the user of the Internet Tablet with stylus-input activates a text-input UI element. The running application is resized as the on-screen-keyboard reserves space from the application area.

Statusbar/titlebar can be expanded by user defined plug-in applications, providing different status information of the applications. Also the main window (visible when no applications are running, or all applications are minimised) allows running plug-ins, called `home applets`, usually being some kind of small informational applications, e.g. news ticker, weather information or clock.

1.3 Hildon user interface views

The Hildon user interface has several layout modes which applications can use, and even switch between views dynamically.

1.3.1 Normal view

- Application area of 696x396 pixels
- Task navigator, statusbar/titlebar visible

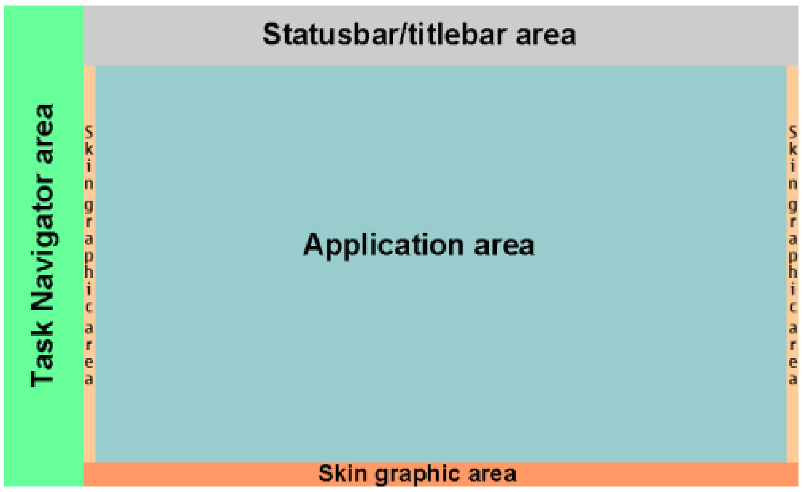

Figure 1.3: Normal view

1.3.2 Normal view with toolbar

- Application area of 696x360 pixels with a single toolbar
- Application area of 696x310 pixels with two toolbars (e.g. Application and Find toolbars)
- Task navigator, statusbar/titlebar visible
- Skin graphics area on the bottom of the screen replaced by toolbar

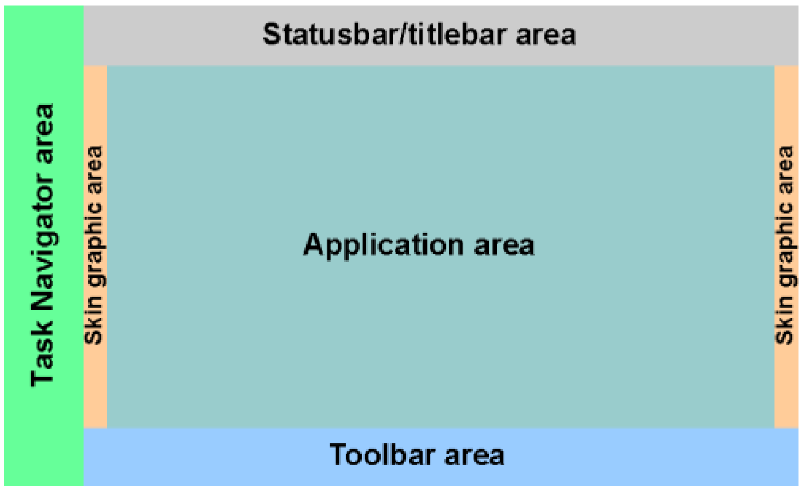

Figure 1.4: Normal view with Toolbar

1.3.3 Full screen view

- Application area of 800x480 pixels fully available
- Task navigator, statusbar/titlebar and skin graphic area not visible
- Mode can be activated and deactivated by a hardware button or by the application code

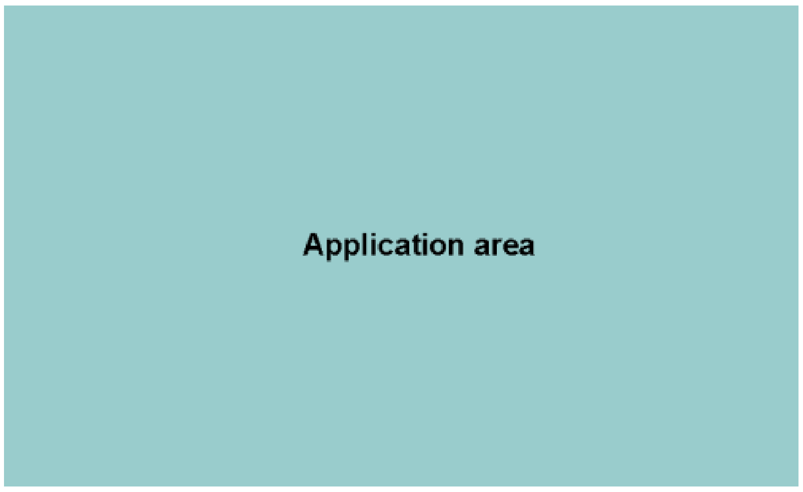

Figure 1.5: Full screen view

1.3.4 Full screen view with toolbar

- Variation of the full screen view

- Application area of 800x422 pixels with a single toolbar
- Application area of 800x370 pixels with two toolbars (e.g. Application and Find toolbars)
- Task navigator, statusbar/titlebar and skin graphic area not visible
- Toolbar should be scalable as it can be visible in both normal and full screen modes

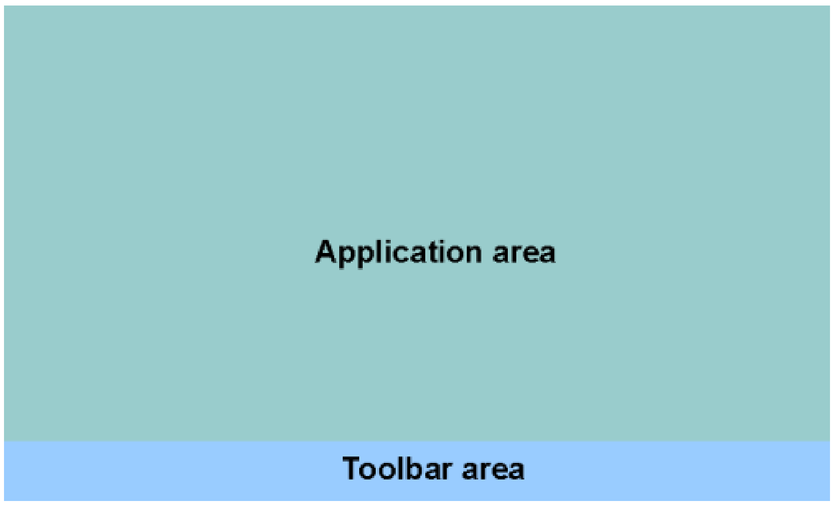

Figure 1.6: Full screen view with toolbar

1.4 Event-loop model

GTK+ is a event-loop based (or event-driven), cross-platform GUI library. In event-loop programming model when the user is doing nothing, GTK+ waits in its `main loop`, waiting for input. When user performs an action - for example a click of a button - the main loop wakes up and sends an event to one or more widgets. When widget receives an event, they usually emit one or more `signals`. These signals can be connected in the application code to functions performing certain action based on the signal emitted and the widget emitting the signal. Functions connected to a signal are referred as `callback` functions. After a callback finishes its task, GTK+ will return to the main loop and wait for more input. Events can also be sent by the application engine itself.

There are graphical GUI-builder applications that assist in creation of the UI-schema and even binding the signals from UI elements to the callback functions, speeding up the development process significantly. Most commonly used GTK+ GUI-builders are Gazpacho and Glade.

1.5 Asynchronous programming model

Sometimes it is necessary to perform actions in the application at the same time while the `main loop` sits and waits for events. This approach is called

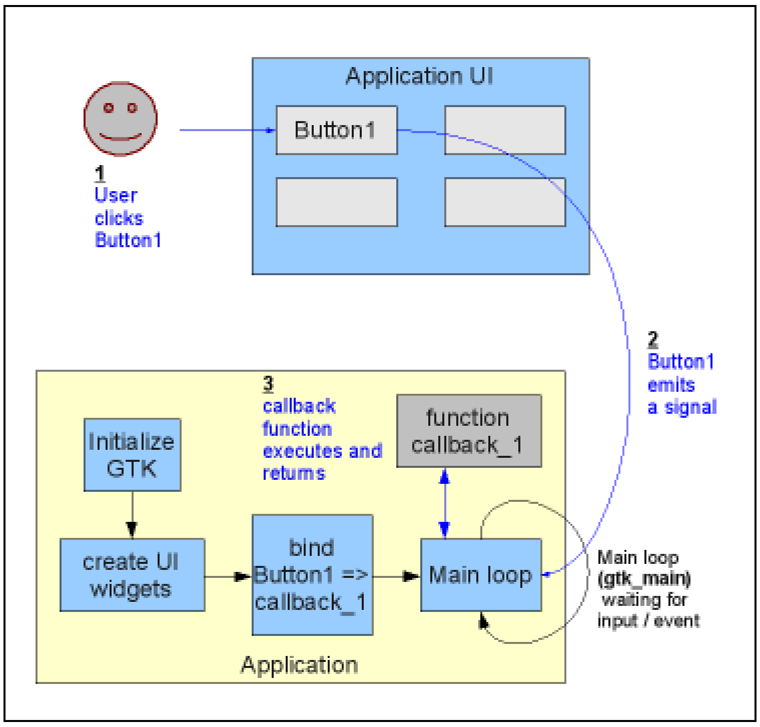

Figure 1.7: Event-loop model, signal and a callback

asynchronicity. Asynchronous actions are executed as non-blocking, meaning the control return to the caller immediately without interrupting the main program flow. The caller will have to specify a callback function that will be called when the operation is completed. Using asynchronicity makes the application UI more responsive and prevents the application "locking up" while, for example, reading data from over the network connection. For example GnomeVFS API has asynchronous counterparts to all functions. Callbacks for asynchronous operations are triggered in the normal event-loop, meaning that the application will be able to handle both GUI events and GnomeVFS events simultaneously. Other example of the API supporting asynchronicity is D-Bus, more information about using D-Bus asynchronously can be found from the maemo platform development course material.

Other option to achieve asynchronicity and improve UI responsiveness is to use threads, but this approach is not recommended unless you have experience on thread based programming. Threaded applications and other software components are hard to debug and may easily cause synchronisation problems.