Maemo Diablo Runtime View of maemo Training Material

February 9, 2009

Contents

Chapter 1

Runtime View of maemo

This chapter gives an overview of the application life-cycle on the maemo platform, introduces services used in the application state-management and resource-saving.

Components involved in the application life cycle management and switching are following:

- Task Navigator (TN)
  - Lists the applications in menu and launches them by user request
  - Performs a background killing of applications in case of low memory.
  - Lists applications, both running and background killed applications (user sees them as running applications).
  - Switches between running applications (or to background killed application) by:
    - Requesting the window manager to top the application window.
    - Sending the application a message requesting it to top a window.
    - Restarting the application if it was background killed.
- D-Bus session bus
  - Executes or activates applications by sending activation messages.
  - Takes care that only one instance of the application is running at a time.
  - Watches the application to register itself within given time-out. If application does not register, D-Bus assumes the application startup failed and will kill the process.
- Maemo launcher
  - Provides a way to speed up some applications startup time by providing a way to execute applications that has been compiled as a shared library.
- Window manager
  - Takes care of handling the window switching and window stacking.

1.1 Platform startup

The first phase of the platform startup is the bootloader, which will load the Linux kernel into memory. Linux kernel then loads the initial filesystem (InitFS) from its' own filesystem partition. InitFS, used as a root filesystem during the startup, contains necessary binaries and scripts to initialise the system and to access the actual root filesystem, which will be mounted after the InitFS scripts finish. Root filesystem contains several `init`-scripts, which will handle the startup of necessary daemons, services and the application framework itself, everything that makes the maemo platform. See chapter "maemo Platform Development" for more information.

1.2 Platform state management

Platform state is managed through several system software components:

Device State Management Entity (dsme) A daemon responsible for managing the states of the device, including shutdown and startup. It monitors the status of critical processes (such as D-Bus, X11 and Window Manager), initiates power saving operations based on inactivity etc.

Mode Control (mce) Provides interfaces for controlling device modes, such as offline mode (disabling of Bluetooth and WLAN), and various system level user interfaces, such as device lock, touch screen and keypad lock, LEDs, etc.

Battery Management (bme) Responsible for battery voltage monitoring and recognition, battery charging, and charger recognition.

One of the important (and interesting from developers point of view) components is D-Bus. The D-Bus message bus is a central part of the platform architecture. Applications should listen to D-Bus messages indicating the state of the device, such as "battery low" and "shutdown". When receiving these messages, the application may, for instance, ask the user to save any files that are open, or perform other similar actions to save the state of the application. There is also a specific system message, emitted when applications are required to close themselves.

1.3 Application startup

In maemo platform, user starts the applications primarily from the Task Navigator. There are also few other ways to start the application, either from the Status Bar (e.g. connection manager), from File Manager to view a file, or from other applications (e.g. "Send as E-mail").

User normally start applications from the Task Navigator. Task Navigator starts the application by sending a D-Bus message to the application service with the D-Bus `auto-activation flag` set.

Other applications can also start applications implicitly by sending a D-Bus message, for example to open a file of certain MIME-type that the application has registered to the MIME database.

Every application in the Internet Tablet has a well-known name uniquely identifying the application, e.g. "Browser" or "Email". D-Bus has a service name for each application, derived from the application name.

Where applications get the D-Bus service name:

Task Navigator The service name is specified in the applications' `.desktop` file

File Manager and Browser The service name is retrieved from the GnomeVFS MIME-type-handler registry via LibOSSO-MIME library

Other applications use the service application API libraries. The libraries know what services the application registers to D-Bus.

D-Bus daemon looks into application `.service` file to see how to execute the application before delivering the message. Applications launched by the D-Bus daemon will only have one instance of them running, because D-Bus does not allow the same service name to be registered by more than one process.

The application launched by the D-Bus daemon will inherit environment variables that were defined at the time when the D-Bus session bus was started. D-Bus does not provide a way to change the environment variables passed to an application. As the Task Navigator uses D-Bus to launch the application, it's same as directly launching the application using D-Bus.

Environment variable should not be used for dynamic configuration changes, since they require program restart and D-Bus does not support that. The language change is also communicated through environment variables. As applications and their libraries also cache locale state, changing the language in Internet Tablet requires restarting all applications and processes.

When the application is started, the window manager takes care of drawing the title bars, dialog borders and windows. Application windows are in a stack. When you top an application, it comes on top of the stack. The window manager also keeps the application dialogs stacked together with the application in the window stack, thus when the application is topped, it's dialogs are topped too.

The application can top itself, or the Task Navigator can top the application by sending a standard X message.

To conserve memory and resources, only single instance of an application can be running at the same time. If application is already running, it will only receive a message about the new invocation (for example "open file") and top itself (bring the window to foreground). User can switch to another, already running application either by using the Task Navigator UI, or by closing the topmost application.

1.4 Application state management

Application framework has a mechanism for shutting down GUI applications on the background to save memory so that other application can be run. This is called background killing.

Background killing is implemented by Task Navigator to transparently close an application when the user does not see it and to restart the application when user needs it again. This is possible because applications are required to be able to save their user interface (UI) states and Task Navigator has a list of all

running UI applications. The application is required to save its UI state when it moves to background.

If in some cases Internet Tablets don't have enough memory to run all the applications at the same time, the system may kill an application running in the background that has indicated to be killable.

Saving the UI state may not always be feasible for the application (e.g. during a download in progress), that is why the application must notify the Task Navigator when it has saved the state and can be killed. Task Navigator will kill all the killable applications when system notifies that it is low of memory. When the application is started again, it is required to rebuild the UI according to the saved state. Task Navigator won't restart the application if there is not enough memory in the system for that.

1.4.1 UI State Saving

The application UI state saving is assisted by LibOSSO. The LibOSSO library creates the state file and provides the file descriptor to the application. Libosso will make sure that the real state file will not be updated until the state file write has completed and file closed. The application uses the standard POSIX filesystem API for writing and reading to the file descriptor.

If device is restarted, the UI states will be discarded, so applications start from their default state. Each application with different version number will have it's own UI state file, meaning that if the application is updated (version number changes), it will start from the default state.

1.4.2 Autosaving User data

Some applications are required to save the unsaved user data periodically when on the foreground (top), so that as little as possible is lost on battery failure. Applications should register a LibOSSO callback function for this operation and tell when their user data has changed (the application is in "dirty state"). LibOSSO will then tell the application when they should do the saving.

Note that applications should call "forced autosave" LibOSSO function when they go to background (LibOSSO does not know when this happens).

1.5 Application termination

Applications exit when user closes them from the application UI, or when system requests that. System will request (and force) the applications to exit when:

- The Internet Tablet battery charge drops low
- The Internet Tablet is switched off
- User changes the Internet Tablet language settings
- User resets the Internet Tablet to factory settings
- User tries close or to switch to an application that doesn't respond

- User must accept to dialog whether to close a non-responsive application

If the Internet Tablet runs low on memory, the system can request application to be background killed. If there is not enough free memory to satisfy the applications' request for more memory, the application will be killed by the kernel.