How to work with views and view architecture

View Architecture

The View architecture is part of the overall UI Control Framework for Symbian OS, and it allows applications to make and receive requests to show a particular view of their data. It also provides finer level of integration between the user interfaces of different applications, and enables users to navigate through the overall UI on the basis of the task they are performing, instead of having to load different applications.

The View architecture allows you, for example, to create links directly from the email addresses in the Detail view of a Contacts application to a 'New email' view for that particular contact in your Messaging application. This enables the user to quickly move from one task to another when using the phone.

Defining View

A view is essentially defined as a class that enables you to display application data. Views are owned by the application's main user interface class (app UI). This is derived from CEikAppUi (the framework provided by the Uikon Core API) and CCoeAppUi (the base class in the UI Control Framework):

- CEikAppUi – this class handles application-wide aspects of the application's

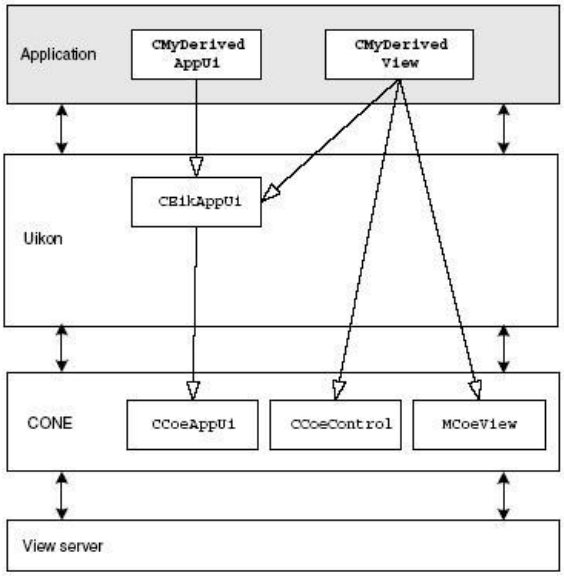
user interface such as toolbar pop-up menus, opening and closing files and exiting the application cleanly. Every Uikon application should derive its own app UI class from CEikAppUi to handle application-wide user interface details.

- CCoeAppUi – this is a general-purpose application user interface

class which handles application-wide aspects of the user interface, such as key-press events. When the view is to be used as a control it also needs to be derived from CCoeControl, the control base class from which all controls are derived. The client/server process of sending and receiving requests to display a particular view is handled by the View server, which is the main part of the View architecture. This client-server interface (API) is not used directly by your application, but through wrapper functions within CEikAppUi (derived from CCoeAppUi), in the app UI framework. So, for a class to be called a view, it must, therefore, also implement specific virtual functions available from the abstract view interface, MCoeView, aswell as declare a View ID, so the View server can recognize that particular view:

- MCoeView – this class specifies an interface for views and should be

used by all application views.



Implementing View

When creating a view for your application, the overall process is:

1. **Create the View**-this includes creating the view class and implementing the functions that enable view activation and deactivation

2. **Register the View**-this means calling a specific function implemented in CEikAppUi, to make the view known by the view server.
3. **Enable Switching between the views**-this includes activating and deactivating views (through buttons, links or menus), as well as packaging data to be sent from one view to another, as either a UID or a descriptor message.
4. **Deregister the View**-this means calling the deregister function implemented in CEikAppUi, so the View server does not try to load the view when the application is closed.

Advantages

- **Application/view switching capability** – the user can switch from one view to another, either within the same application or within another application.
- **Support for saving data** – by registering a view with the View server, the data for that view is always saved before the view is deactivated.
- **Support for sending data** – by packaging messages as descriptors identified by the UID, data can be sent from one view to another (within the same or different applications), via the View server.

When to use View Architecture

When your application

- Has multiple screens that form complex navigational paths
- Has to save data on every view switching, to update the model with the newly entered or updated data
- Has to send data among screens or to external applications