Chapter 5

Case studies

This chapter discusses a number of case studies of Meshoil programs run in the concept demonstrator:

• Safety fan (page 128)—demonstrates the process of designing a working user interface by starting from a very simple description of hardware functionality and building up sophisticated user operation in a number of stages

• Radio (page 152)—a user interface for a 3-band radio that demonstrates the mapping of physical controls typically used to operate electronic devices, such as keys, knobs and sliders

• Sports car (page 171)—modelling the performance dynamics of a sports car to demonstrate the programming of dynamic systems

• Sports car with safety fan and radio (page 182)—all three previous examples bundled into one program to demonstrate the simultaneous running of more than one functional program in the Mesh engine

• Calculator (page 184)—a user interface for a calculator that supports all the usual functions such as percentages, square roots, inverse, repeated constant calculations and memory storage

• 3-in-1 calculator (page 202)—demonstrates a layered approach to programming by creating a high-level, multi-notational calculator user interface that is supported by a low-level method library

• Mobile phone/emergency pager (page 217)—a user interface for a communications device that works either as a Nokia 6210 mobile phone or as an emergency pager, demonstrating Mesh concepts of hardware reuse, virtual logic and futures programming.
5.1 Safety fan

5.1.1 Overview

This example demonstrates how Meshoil makes it possible to build a sophisticated user interface from a very simple description of hardware functionality. It also shows how Meshoil can be used to support the process of design, starting from a simple list of desired user features and progressively adding to it until eventually a fully working design is reached—a process characterised by a natural transition from declarative to imperative programming.

The example takes the same fan device that was discussed in section 2.2 on page 26 and creates a user interface that makes the most of the device’s hardware capability (Motor and Pan controls) and limited means of physical control (Display, Black key and White key controls).

The aim is to make the fan as safe as possible to operate. The user interface incorporates such features as a Sleep mode to make the fan switch off automatically after a set time, and a means of locking the fan for child safety.

The final working design is built up in five stages.

5.1.2 Running this simulation

Some user interface operations are timed, such as the fan automatically switching off after 60 minutes in Sleep mode. In the design, these times are quoted in either seconds, minutes or hours. For the purpose of this simulation in demonstrating how the user interface works, all numerical measurements of time are interpreted as seconds. For example, you only have to run the simulation in Sleep mode for 60 seconds to observe the fan stopping automatically.

Design stage 5, the final stage, is the version that runs when the SafetyFan.xml file containing this example is loaded in the Mesh engine. The earlier design stages are ignored because the bricks containing their code in the file are given inactive spin—a spin that the Mesh engine does not recognise.

5.1.3 Design stage 1—mission statement

The starting point in the design process is a wish list of features that the user interface should support.

A modal approach is chosen as the best way to support many user features when the means for their control is known to be so limited. For example, a Lock mode is imagined in which the two keys are disabled from operating the fan in the normal way. Instead, the keys in this mode are used to input a PIN to release the lock.
Other modes are planned, as shown in Figure 5.1 that displays the program at the end of this design stage.

<table>
<thead>
<tr>
<th>Safety Fan (design stage 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td><strong>Normal mode</strong></td>
</tr>
<tr>
<td>Allow the user to control fan speed and panning independently</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td><strong>Sleep mode</strong></td>
</tr>
<tr>
<td>Same as Normal mode except that the fan automatically switches off after a set time.</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td><strong>Overuse mode</strong></td>
</tr>
<tr>
<td>Protect the fan from damage resulting from excessive use.</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td><strong>Unstable mode</strong></td>
</tr>
<tr>
<td>If the fan is running erratically, switch it off and disable it from use.</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td><strong>Locked mode</strong></td>
</tr>
<tr>
<td>Allow the user to lock the fan as a child safety feature.</td>
</tr>
</tbody>
</table>

Figure 5.1: The user interface program at the end of design stage 1

At this stage, the program is no more than a description of intent. It does not do anything if it is run, since it only consists of brick (<b>) and annotation (<a>) elements. Although the program contains no programming logic, it has created a logical framework ready for expansion in subsequent design stages.

This framework is needed as a place to hang future code. It is not the only one that could have been chosen, but it is convenient since it allows us to focus on what we want the user interface to do, rather than on how we are going to achieve it. In fact, this framework proves not to be the best way to structure the code from the point of view of executional efficiency, but this something that is rectified in the final, fifth design stage.

5.1.4 Design stage 2—adding flesh to the bones

Having decided on the main features of the user interface, the next task is to consider how these features are going to be accessed and controlled. This is a critical part of the design process since the fan only has two keys.

To help overcome this limitation, double as well as single key presses are used in the design. In a double key press, the White key is used as a shift key with the Black key. Since the keys are mono-stable, they need to be used to toggle settings rather than select specific settings.

This design stage adds some top-level logic about which keys are used for which features and in what modes. Keys work differently depending on whether the fan is running or not. Underneath this layer more purely descriptive information is added to expand on what will happen in each mode. For example, Sleep mode goes through various wind-down phases such as slowing down and discontinuing panning before the fan finally switches off altogether. The concept of an ‘awake button’ is created that will allow the user to prolong Sleep mode in rather the same manner that a snooze button is used on an alarm clock. The PIN for Lock mode is set as a double key press followed within 10 seconds by a White key press.

For safety, it is realised that pressing the Black key must immediately switch the fan off regardless of what mode the fan is in. A control called Mode is created to represent the current mode.
that the fan is in. This control acts as a software variable to guide the running of the program.

Default settings are taken advantage of in writing the code. For example, the default for the Motor control is 0.0, which equates to the fan not running. The default for the Mode control is a blank string which is used to represent the fan being off and not in any mode.

Figure 5.2 shows the program at the end of this design stage.

---

**Figure 5.2: The user interface program at the end of design stage 2 (1 of 2)**
5.1.5 Design stage 3—completing the logic

The design work is becoming less declarative and more imperative in nature.

This stage concentrates on providing abstract logic for implementing the descriptions in the previous design stage of what should happen in each mode. For example, the description that The White key toggles a choice of slow, medium and fast speeds is transcribed into the equivalent logic based on the settings of a control called Speed. The timing of the various wind-down phases in Sleep mode are detailed using a newly created control called Timer.

The logic is still abstract at this stage, because it does not relate to the hardware. Interpretation of exactly what Slow, Medium and Fast settings mean to the device is left for a future design stage. Similarly, numerical values are used in the code to represent the times when automatic operations are triggered without giving any consideration to how timers will work and how these values will be reached in practice.
New user features are added as they spring to mind during the design process. For example, in clarifying how PIN entry will work, the idea of tamper detection and reporting on the entry of a wrong PIN is thought up and added to the program.

Figure 5.3 shows the program at the end of this design stage.

Figure 5.3: The user interface program at the end of design stage 3 (1 of 3)
5.1. SAFETY FAN

Figures 5.3: The user interface program at the end of design stage 3 (2 of 3)
CHAPTER 5. CASE STUDIES

Figure 5.3: The user interface program at the end of design stage 3 (3 of 3)
5.1.6 Design stage 4—first working version

This design stage completes the task of creating a working version of the user interface.

Abstract logic is converted into its physical counterpart. Design concepts such as Slow, Medium and Fast are transcribed into their hardware-equivalent values, chosen to be the 0.3, 0.7 and 1.0 settings of the Motor control. Further operational details are added as required. For example, when the user switches on the fan in Normal or Sleep mode, the speed that the fan starts running needs to be specified.

Key timing issues need to be addressed to correctly distinguish between single and double key presses. For example, an operation that requires a White key press cannot be triggered the moment the White key is pressed, since the user might be using the White key as the shift key for a double key press instead. The condition for detecting the user’s intention of a White key press is the point that the White key is released while the Black key is not also being pressed.

Figure 5.4 on page 136 shows the program at the end of this design stage. The main operating features of the user interface are labelled from 1 to 21.
CHAPTER 5. CASE STUDIES

Figure 5.4: The user interface program at the end of design stage 4 (1 of 5)
5.1. SAFETY FAN

Figure 5.4: The user interface program at the end of design stage 4 (2 of 5)
Figure 5.4: The user interface program at the end of design stage 4 (3 of 5)
5.1. SAFETY FAN

3

Overuse mode
Protect the fan from damage resulting from excessive use.

1

After continuous running for 800 hours (25 days), switch the fan to the slowest speed and stop any panning. Disable use of the keys for changing speed and pan settings.

[14] Auto start Overuse mode

New!

if

Timer
000

1 2 3

Mode Motor Pan
Status 0.3 Default

4

Unstable mode
If the fan is running erratically, switch it off and disable it from use.

1

Detect erratic running by noting when the actual speed of the motor in the device differs from any of the values specified in the user interface. Allow 30 minutes for the fan to cool before cancelling Unstable mode and allowing the fan to be used again.

[15] Auto start Unstable mode

[16] Auto cancel Unstable mode

Figure 5.4: The user interface program at the end of design stage 4 (4 of 5)
5.1.7 Design stage 5—tuning for efficiency

While the user interface from the previous design stage runs perfectly well, it is possible to tune it to make it execute more efficiently. This is instructive, but it isn’t really necessary when the program is a user interface, since most of the time such a program is idling and waiting for the user to do something.

In this example, the program was built around the initial design concept of using modes to get around the problem of only having two keys for operating the fan. Since so much of the code is dependent on the pressing of keys, we find that program size can be reduced by restructuring the code around key presses instead of modes.

In this design stage, the code of the design stage 4 is rearranged under four top-level bricks: Black key operations, White key operations, Black and White shift key operations, and Automatic operations triggered by elapsed time. This reduces program size from 103 bricks to 80, a saving of 22%. Although the resultant effect of the logic is unchanged, the average number of steps involved...
in executing each user operation is reduced because more of the logical pathways through the code are shared. This reduces the time taken for code to be parsed and increases the speed of the program execution cycle.

Reducing program size by combining areas of identical logic is, itself, a purely logical problem. The task is made that much easier because of Meshoil’s simple, modular syntax. Although program tuning was achieved in this example by manual inspection, it is not hard to imagine the creation of a special optimising tool for doing the job automatically.

5.1.8 Multithreading
As a further refinement in executional efficiency, the user interface has been broken up into concurrently running threads for each of its four main tasks by placing the Black key operations, White key operations, Black and White shift key operations, and Automatic operations code in individual do bricks.

Figure 5.5 on page 142 shows the program at the end of this final design stage. The process of tuning the program is evident from the way that the 21 main operating features of the user interface have become shuffled.

5.1.9 Hardware simulation
The program includes a fifth do brick called Timing and key management that runs an additional thread for managing timing issues and simulating the physical operation of the keys. This brick consists of the two bricks Timing management and Key management.

Timing management runs the various timers needed for triggering the automatic user interface operations. The basic timer is Beats. Each time the Mesh engine parses the code, Beats is incremented by one. Other timers such as Timer, Seconds, and Minutes are derived from Beats. Timer acts as a stopwatch, counting in seconds for triggering automatic operations. Seconds and Minutes are used to build the display of time in Clock that is mapped to the fan’s Display control, along with feedback on what the fan is currently doing. The number of Beats equivalent to a second is an empirical value that depends on how quickly the Mesh engine parses the code.

Key management simulates the return action of the two mono-stable keys to their Up position after being ‘pressed’. So that shift key operations can be simulated when using the drop-down menus of the Mesh engine application, keys remain Down for a short time after being ‘pressed’ before being reset to Up.
Figure 5.5: The user interface program at the end of design stage 5 (1 of 7)
Figure 5.5: The user interface program at the end of design stage 5 (2 of 7)
Figure 5.5: The user interface program at the end of design stage 5 (3 of 7)
Figure 5.5: The user interface program at the end of design stage 5 (4 of 7)
Figure 5.5: The user interface program at the end of design stage 5 (5 of 7)
5.1. SAFETY FAN

Figure 5.5: The user interface program at the end of design stage 5 (6 of 7)
CHAPTER 5. CASE STUDIES

Figure 5.5: The user interface program at the end of design stage 5 (7 of 7)
5.1. SAFETY FAN

5.1.10 Running the safety fan user interface

Figure 5.6 shows a sequenced example of running the safety fan user interface.

The Hardware controls/Software controls window displays all the controls that have been defined in the program, set up in the two with bricks called Hardware controls and Software controls. The Timing and key management window displays all controls used in the fifth thread for handling timing and key management. These controls display here because, in being used exclusively in this thread, there was no need to define them in any with brick in order to increase their scope and make them accessible to other threads.

No windows display for the four threads running the user interface (Black key operations, White key operations, Black and White shift key operations and Automatic operations), since all the controls used by these threads are already defined in the two with bricks.

Frame 1  The fan is off. This is indicated by the Display showing nothing other than the running digital clock.

Frame 2  The user presses the Black key. The Display indicates that the fan is on at the slowest speed. This speed is defined by the user interface as being 30% of the maximum motor speed of 1.0 (ie, the Motor setting of 0.3).

Figure 5.6: Running the safety fan user interface (1 of 3)
Frame 3  The user presses the White key twice in succession to toggle the speed to medium then fast. The Motor setting is now at its maximum of 1.0.

Frame 4  To simulate unstable running caused by a drop in the voltage, the setting of the Motor control is perturbed by directly attempting to change its value. This represents an action taken outside the realm of the user interface, since in the real world the only means available to the user for interacting with the running of the fan would be the two keys.

Frame 5  On detecting that the Motor is no longer running at any of the three standard speeds, the user interface immediately shuts the fan down in Unstable mode. The fan is now disabled until this mode automatically times out.

Figure 5.6: Running the safety fan user interface (2 of 3)
5.1. SAFETY FAN

Frame 6  After a further 30 seconds has elapsed, Unstable mode automatically ends. The fan can now be used again.

Frame 7  With the fan switched off but usable, the user presses both keys to put the fan in Lock mode to make the fan safe to be left unattended while young children are around. The fan is now disabled until the PIN is entered.

Frame 8  Someone makes an illegal attempt to switch the fan on by pressing the Black key. Since the fan is locked, it does not respond other than to indicate on the Display that the Black key has been tampered with. Until the entry of the correct PIN—a double key press followed within 10 seconds by a White key press—the fan cannot be used.

Figure 5.6: Running the safety fan user interface (3 of 3)
CHAPTER 5. CASE STUDIES

5.2 Radio

5.2.1 Overview

This example is a user interface for a conventional 3-band radio. It demonstrates how Meshoil can be used to map the running of a user interface to the full range of physical controls such as keys, knobs and sliders that are typically used to operate electronic devices.

5.2.2 Hardware controls

The radio’s physical controls consist of a display, seven push-button keys, two knobs and a slider. The keys are labelled as the Power key, the three radio button keys 1 key, 2 key and 3 key, the Up key and Down key for manually scrolling the frequency, and the Save key for saving the current frequency as a preset. The knobs have 180-degree travel. The discrete slider has three positions.

The hardware controls consist of circuits for power, volume, tone (frequency attenuation), tuning and frequency band selection. The tuning circuits cover the standard frequency range for Australia.

5.2.3 Software

The main role of the user interface is to make the hardware controls accessible to the user by linking them to the physical controls. The keys are used to control the power and tuning circuits, the knobs for the volume and tone circuits, and the slider for frequency band selection. Figure 5.8 on page 154 shows the program for running the radio user interface, structured around the use of the physical means of controlling the radio (code for simulating physical key operation is not shown).

The Power key is used to toggle the DC supply circuit On and Off. The 1 key, 2 key and 3 key radio buttons are used to select preset frequencies for the currently selected frequency band.

The Up key and Down key are used for manually scrolling the frequency. Each press increments or decrements the frequency by the appropriate step size: 0.05MHz for FM, 9kHz for AM and 0.005MHz for SW. These keys act as infinite-travel, continuous controls. Holding down a key causes the frequency to scroll and wrap at the end of the frequency range endlessly.

The Volume knob is a limited-travel, continuous control that maps to the Volume circuit. Its 180-degree travel is made to correspond to the minimum and maximum volume settings.

The Tone knob has the same 180-degree travel as the Volume knob but its zero position is at the centre of its range rather than at the start. This knob maps to three separate circuits: Bass boost circuit, Attenuator circuit and HF filter circuit. These circuits cut in and out at different knob settings. The Attenuator circuit maps to the inverse of the Tone knob’s full range. The HF filter circuit starts to operate when the knob is turned more than 45 degrees in the treble (anti-clockwise) direction, reaching maximum power at the maximum treble position. The Bass boost circuit is simply on whenever the knob is turned more than 60 degrees in the bass (clockwise) direction.
Figure 5.7 shows these relationships graphically.

![Radio Diagram](image)

**Figure 5.7: Mapping the knobs and keys to radio functions**

The **Band selector** is a multi-state, discrete slider that is similar in function to a stepped rotary switch. Each state is made to correspond to one of the three available frequency bands.

The **Save key** allows any of the three radio buttons to be reprogrammed with the current frequency. Pressing the **Save key** selects **Save** mode, indicated by an ‘s’ on the display. In this mode, pressing a radio button does not change the current frequency but reprograms the radio button to the frequency currently displaying. If the **Save key** has been pressed in error, **Save** mode can be cancelled by pressing the **Save key** again.

The **Display** is divided into three fields: the ‘s’ **Save** mode indicator, the frequency band and the currently selected frequency.

### 5.2.4 Running this simulation

To make it easy to simulate the pressing of keys, controls representing keys are declared as having the integer value 0 (key up) or 1 (key down), rather than the string **Up** or **Down**. This causes these controls to be represented on the screen by GUI spinners instead of drop-down menus, and allows the ‘pressing’ of a key to be conveniently simulated by a single mouse click.
### Figure 5.8: Radio user interface (1 of 8)

**Hardware controls**

<table>
<thead>
<tr>
<th>Display</th>
<th>Small alphanumeric display screen.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power key</td>
<td>1 key</td>
</tr>
<tr>
<td>2 key</td>
<td></td>
</tr>
<tr>
<td>Up key</td>
<td></td>
</tr>
<tr>
<td>Down key</td>
<td></td>
</tr>
<tr>
<td>Save key</td>
<td></td>
</tr>
<tr>
<td>DC supply</td>
<td>Controls the DC power supply to the radio.</td>
</tr>
<tr>
<td>Band selector</td>
<td>Sliders, multi-state.</td>
</tr>
<tr>
<td>Tone knob</td>
<td>180-degree travel, stepping 0 to 180.</td>
</tr>
<tr>
<td>Volume knob</td>
<td>180-degree travel, stepping 0 to 90.</td>
</tr>
<tr>
<td>Volume-tone circuits</td>
<td>Circuits have value in the continuous range of 0.0 to 1.0.</td>
</tr>
<tr>
<td>Bass boost circuit</td>
<td>Controls the bass frequency boost circuit (circuit either off or on).</td>
</tr>
</tbody>
</table>

**Radio**

1. **Display**
   - Small alphanumeric display screen.

2. **Power key**
   - 1 key
   - 2 key

3. **Up key**

4. **Down key**

5. **Save key**

6. **DC supply**
   - Controls the DC power supply to the radio.

7. **Band selector**
   - Sliders, multi-state.
   - FM
   - AM
   - SW

8. **Tone knob**
   - 180-degree travel, stepping 0 to 180.

9. **Volume knob**
   - 180-degree travel, stepping 0 to 90.

10. **Volume-tone circuits**
    - Circuits have value in the continuous range of 0.0 to 1.0.
    - Volume circuit: controls the volume (0 mute, 1 max).
    - Attenuator circuit: controls high frequency attenuation (0 none, 1 max).
    - HF filter circuit: controls low frequency attenuation, truncating frequencies that drop below a variable threshold (0 none, 1 max).

11. **Bass boost circuit**
    - Controls the bass frequency boost circuit (circuit either off or on).

**Table:**

| in | integer |
| 1  | min 2 max 0 1 |

| out | string |
| 1  | list |

| 1 | list |

| off | On |

| set | decimal |
| 1  | min 2 max 0.0 1.0 |
5.2. RADIO

Figure 5.8: Radio user interface (2 of 8)
CHAPTER 5. CASE STUDIES

Figure 5.8: Radio user interface (3 of 8)
Figure 5.8: Radio user interface (4 of 8)
Figure 5.8: Radio user interface (5 of 8)
Figure 5.8: Radio user interface (6 of 8)
Figure 5.8: Radio user interface (7 of 8)
Figure 5.8: Radio user interface (8 of 8)
5.2.5 Running the radio user interface

Figure 5.9 shows a sequenced example of running the radio user interface.

**Frame 1** The radio is off. This is indicated by the blank Display.

Figure 5.9: Running the radio user interface (1 of 9)
Frame 2  The user presses the Power key to switch the radio on. FM is the currently selected band but the Volume knob is at the zero setting.

Figure 5.9: Running the radio user interface (2 of 9)
The user rotates the Volume knob 45 degrees clockwise, turning up the Volume circuit to 25% of the maximum value. The Tone knob is at the middle, neutral position (setting 0). In this position, the Attenuator circuit is working at half strength (setting 0.5), there is no HF filtering (HF filter circuit at setting 0) and the Bass boost circuit is Off.

Figure 5.9: Running the radio user interface (3 of 9)
The user rotates the Tone knob 70 degrees anticlockwise to boost the bass. This has the effect of increasing the strength of the Attenuator circuit that attenuates high frequencies. The Bass boost circuit is also now on, having come on at the point that the knob reached the 60 degree position.

Figure 5.9: Running the radio user interface (4 of 9)
Frame 5  The user moves the Band selector to the AM setting. The change is indicated on the Display as the radio switches to the 729kHz frequency that the AM tuner is currently tuned to.

Figure 5.9: Running the radio user interface (5 of 9)
Frame 6  The user holds down the Up key, causing the frequency to scroll up repeatedly in steps of 9kHz.

Figure 5.9: Running the radio user interface (6 of 9)
Frame 7 When the station broadcasting at 972kHz is reached, the user releases the Up key and presses the Save key to save this station as a preset.

The ‘s’ symbol appears on the left of the Display indicating that the user must indicate which radio button to reprogram.

Figure 5.9: Running the radio user interface (7 of 9)
5.2. RADIO

Figure 5.9: Running the radio user interface (8 of 9)

Frame 8  The user presses the 2 key radio button. The AM frequency preset for the 2 key changes from 891kHz to 972kHz. This completes the save operation and the ‘s’ symbol disappears from the Display.
The user presses the Power key to switch the radio off. The DC circuit switches off and the Display goes blank.

Figure 5.9: Running the radio user interface (9 of 9)
5.3 Sports car

5.3.1 Overview

This example models the performance dynamics of a sports car. It demonstrates how Meshoil can be used for programming dynamic systems.

5.3.2 Hardware controls

Figure 5.10 on page 172 shows the hardware controls used in the program that represent the list of dashboard and floor controls available for ‘driving’ the car.

On the dashboard is the Warning display, the RPM dial showing the engine revs, the Speed (km/h) dial displaying the car’s speed in kilometers per hour, the Odometer (0.1km) readout displaying the distance travelled in tenths of a kilometer, and the Turbo indicator showing whether the turbo fan is on or not.

Drive controls consist of the Ignition key, the Trip reset button for resetting the odometer, the Cigarette lighter, the Drive selector for choosing manual (clutchless) or automatic drive, the Power selector for switching between Economy and Sports mode driving, the Gear shift for selecting the four gears, the Brake pedal and the Gas pedal.

Automatic drive has four settings: Automatic 1, Automatic 1-2, Automatic 1-3 and Automatic 1-4 for selecting the range of gears that can be used.

5.3.3 Driving the sports car

The engine is started by selecting Start on the Ignition key. The Gear shift must be in Neutral.

The engine ticks over at 850 RPM. It will stall if revs are allowed to drop below 500 RPM for any reason.

Revving the engine past 7000 RPM generates a red line warning. The engine seizes if revs exceed 8000 RPM, caused either by not changing up in time when accelerating or by changing down when the speed is too fast. Forcing a change to a lower gear by selecting the wrong automatic drive option at high speed can also lead to over-revving problems.

Torque for maximum acceleration peaks around 5000 RPM. To make use of this power band, gear changes in automatic drive occur at higher revs when Sports mode is selected.
Figure 5.10: Dashboard controls and floor pedals
5.3.4 Vehicle dynamics

The model is a simple one, but aims to give the feel of an average sports car. It takes into account the performance of the engine, rolling dynamics of the vehicle and the efficiency of the braking system.

Figure 5.11 on page 174 shows the part of the program that models vehicle dynamics. Modelled parameters include RPM, Power, Engine inertia, Drag, Braking force, Acceleration and Speed. Parameters are updated every half second during a simulated run, with the control called Valves acting as a basic timer for the calculations.

Power is based on engine revs and the extent to which the Gas pedal is depressed. When Sports mode is selected and the revs exceed 3000 RPM, the Turbo kicks in to increase the engine’s Power output. Engine inertia represents the slowing effect created by the engine when the car is moving in gear with the foot off the Gas pedal. It increases with increasing RPM.

Drag, representing rolling resistance of the vehicle, increases with Speed. Braking force is proportional to the square of the Brake pedal depression.

Acceleration is the increase or decrease in Speed calculated in each half-second cycle. Power works to increase the Acceleration while Engine inertia, Drag and Braking force all work to decrease it. If the car is free wheeling in neutral, the only factors affecting Acceleration are Drag and Braking force.

When the car is in gear, the RPM is calculated from the Speed and knowing the gear ratio implied by the currently selected gear. When the car is in neutral, the RPM is calculated according to how far the Gas pedal is depressed.
Figure 5.11: Model of vehicle dynamics (1 of 2)
Figure 5.11: Model of vehicle dynamics (2 of 2)
5.3.5 Running the sports car simulator

Figure 5.12 shows a sequenced example of driving the sports car.

Frame 1  The simulation starts.
The engine is not running. All engine diagnostics in the Simulation of car operation window are inactive.

Frame 2  The driver ‘turns’ the Ignition key to the Start position to start the car.
The key automatically returns to the On position and the engine comes to life. The car ticks over at 850 RPM.

Figure 5.12: Running the sports car simulator (1 of 6)
Frame 3  The driver puts the car into first gear. The car starts moving, but with no foot on the Gas pedal yet, RPM and Speed start dropping.

Frame 4  The driver is too slow to hit the Gas pedal. Revs drop below 500 RPM and the car stalls. The Simulation of car operation window shows the state that the car was in at the point that the engine stalled.

Figure 5.12: Running the sports car simulator (2 of 6)
Frame 5  The driver puts the car back into neutral and restarts the engine.

Frame 6  Determined not to stall the car again, the driver puts the car into first gear and floors the Gas pedal. The car rapidly accelerates, with Acceleration peaking around 5000 RPM.

Figure 5.12: Running the sports car simulator (3 of 6)
Frame 7  The car continues to accelerate, but at a slower rate as revs pass the 7000 RPM red line.

Frame 8  Still the driver does not ease up. Revs exceed 8000 RPM causing the engine to seize.

Figure 5.12: Running the sports car simulator (4 of 6)
Frame 9  Chastened by the experience, the driver decides to try automatic drive. After putting the car back into neutral and restarting the engine, the driver changes the Drive selector to Automatic 1-4 and again floors the Gas pedal.

The car accelerates and starts to go through the gears automatically.

Frame 10  The car continues to accelerate, eventually reaching a top speed of about 170 km/h in top gear.

Acceleration drops to zero caused by the Power output of the engine just balancing the effects of Drag and Engine inertia.

Figure 5.12: Running the sports car simulator (5 of 6)
5.3. SPORTS CAR

Frame 11  Still hungry for speed, the driver moves the Power selector to the Sports mode position. The Turbo immediately kicks in because the revs are already above the threshold required for the turbo fan to switch on in Sports mode. The Power output of the engine increases. With Power exceeding the effects of Drag and Engine inertia, the car starts to accelerate again.

Frame 12  Eventually, a new top speed over 200 km/h is reached when Drag and Engine inertia balance out the increased Power output of the engine.

Figure 5.12: Running the sports car simulator (6 of 6)
5.4 Sports car with safety fan and radio

5.4.1 Running multiple programs simultaneously

This example demonstrates how it is possible to run more than one functional program simultaneously in the Mesh engine. In the same way that a program written for a single purpose in mind can be split up to run as a number of independent threads, functionally distinct programs can be run as independent threads to provide the user with what seems like several programs in one.

Here, the program files of the three previous examples of car, safety fan and radio are loaded one after the other in the Mesh engine. When the Run button on the Mesh engine is pressed, all three programs run autonomously, just as they did when loaded and run individually. The user is able to operate the car, fan and radio all at the same time.

Figure 5.13 shows the three programs running and being controlled from a single window that displays all their hardware controls.

![Figure 5.13: Sports car with safety fan and radio](image-url)
Namespacing has been used to logically isolate the respective programming space for each of the three programs: car for the car simulator, fan for the safety fan user interface and radio for the radio user interface (for details on namespacing, see appendix A.6 on page 315). This was necessary to avoid the programs interfering with each other. For example, the user interfaces for the safety fan and radio both refer to a control called Display. By giving each user interface a different namespace, references to these two quite different controls that just happen to have the same name are kept distinct.
5.5 Calculator

5.5.1 Overview

This example is a user interface for a standard calculator. It supports all the usual functions such as percentages, square roots, inverse, repeated constant calculations and memory storage.

The program is a good example of using brick insertion to achieve code reuse.

5.5.2 Hardware controls

Device hardware is very simple, consisting of a single-line display and a bank of push-button keys. The keys consist of a numeric keypad and the standard set of function keys found on a calculator. The manufacturer decided to add an extra, unlabelled key in case it might be needed for some unplanned function.

5.5.3 Software

The program runs as a typical calculator user interface:

- ‘M’ at the left of the Display indicates when the memory is in use
- the (C) key resets the calculator, and is the only key that still works after an illegal operation such as the square root of a negative number or division by zero
- the (CE) key clears the number that has just been entered
- the decimal point is not allowed to be entered more than once when entering a number
- the (+/-) key toggles the sign of the currently displaying number
- the (%) (sqrt) and (1/x) keys perform percentage, square root and invert operations on the currently displaying number
- the (MC) key cancels the number held in memory
- the (MR) key recalls the number held in memory
- the (M+) key adds the currently displaying number to the number held in memory
- the (M-) key subtracts the currently displaying number from the number held in memory
- pressing the (=) key again repeats the last +, -, * or / operation on the currently displaying number.

To put the blank key to some practical use and for want of a better idea, the designer of this user interface decided to use this key for calculating numbers in the Fibonacci series. The number currently displaying when the blank key is pressed determines how many Fibonacci numbers to calculate. If this number is less than one, none are calculated.
5.5. Code reuse by using brick insertion

Copying parts of the program by using brick insertion reduces the size of the program by more than a half. Figure 5.14 on page 186 shows the do brick that holds the executable code of the user interface. The extent of brick reuse is apparent by the number of bricks that display faint in the figure as inserted bricks.

Various bricks are reused such as Save old number, Start new number, Clear constant, Clear display and Do pending calculation. The Clear display brick saves the previously entered number before clearing it from the Display and entering the first digit of a new number. Instead of defining this brick within each numeric key, it is defined just once for the (.) key and copied by the other numeric keys. The Do pending calculation brick is defined for the (+) key and copied by the other (-), (*) and (/) operator keys. When the user selects an operator, this brick handles any pending calculation for a previously entered operator that needs to be completed first.
CHAPTER 5. CASE STUDIES

Figure 5.14: Calculator user interface (1 of 13)
Figure 5.14: Calculator user interface (2 of 13)
CHAPTER 5. CASE STUDIES

Figure 5.14: Calculator user interface (3 of 13)
Figure 5.14: Calculator user interface (4 of 13)
Figure 5.14: Calculator user interface (5 of 13)
Figure 5.14: Calculator user interface (6 of 13)
Figure 5.14: Calculator user interface (7 of 13)
Figure 5.14: Calculator user interface (8 of 13)
Figure 5.14: Calculator user interface (9 of 13)
Figure 5.14: Calculator user interface (10 of 13)
Figure 5.14: Calculator user interface (11 of 13)
5.5. CALCULATOR

Figure 5.14: Calculator user interface (12 of 13)
Figure 5.14: Calculator user interface (13 of 13)
5.5.5 Running the calculator user interface

Figure 5.15 shows a sequenced example of running the calculator user interface.

**Frame 1**  The calculator is ready to be used.

The user wants to calculate the following expression: \( \sqrt{(31.4 \times 3.5) - \left(\frac{149}{9.64}\right)} \)

**Frame 2**  To do the first part of the calculation, the user enters 31.4, presses the (*) key, enters 3.5 and presses the (=) key.

The result of 109.9 displays. (To save space, only the top of the window is shown from here on.)

Figure 5.15: Running the calculator user interface (1 of 3)
CHAPTER 5. CASE STUDIES

Frame 3  To save the result, the user presses the (M+) key. This saves 109.9 in the calculator’s memory. The ‘M’ on the Display indicates that the memory now holds a value.

Frame 4  To do the second part of the calculation, the user enters 149, presses the (/) key, enters 9.64 and presses the (=) key. The result of 15.456431 displays.

Frame 5  In attempting to subtract the result of the second part of the calculation from that of the first, the user presses the (-), (MR) and (=) keys. This performs the subtraction the wrong way around, subtracting the number held in memory from the result of the second part of the calculation. The incorrect result of -94.443565 displays.

Frame 6  Unaware of the mistake, the user presses the (sqrt) key. The calculator recognises that the user is attempting an illegal operation in calculating the square root of a negative number and locks up.

Frame 7  The user has no choice but to press the (C) key to clear everything and start again. Just as before, the user calculates the first part of the calculation, stores the result in memory and calculates the second part of the calculation.

Figure 5.15: Running the calculator user interface (2 of 3)
5.5. CALCULATOR

Frame 8  This time, the user presses the (+/-) key to change the sign of the result of the second part of the calculation.

Frame 9  To subtract the result of the second part of the calculation from that of the first, the user presses the (+), (MR) and (=) keys. The result of 94.443565 displays.

Frame 10  The user presses the (sqrt) key to obtain the final result of 9.718208.

Frame 11  For interest, the user presses the blank key to see what the first nine numbers in the Fibonacci series look like. The calculator rounds the displayed number of 9.718208 down to the nearest integer and outputs nine Fibonacci numbers on the Display. To continue to use the calculator normally, the user will now need to press the (C) key.

Figure 5.15: Running the calculator user interface (3 of 3)
CHAPTER 5. CASE STUDIES

5.6 3-in-1 calculator

5.6.1 Overview

This example demonstrates how Meshoil can support a layered approach to programming. It extends the calculator user interface described in the previous section to provide the user with a choice of three notations: standard, algebraic (DMAS: priority of Divide/Multiply over Add/Subtract) and Reverse Polish.

The program is split into two levels. The upper level is the user interface that describes how the calculator works in the various notations. The lower-level supports the running of the user interface by providing a library of useful methods. The double-layered approach is implemented by one module of Meshoil forming the user interface and another forming the library. The two modules run as separate threads.

The user interface remains a high-level, abstract description of calculator function that doesn’t perform any arithmetic operations itself since it relies completely on the method library. The method library supports four arithmetic methods (Add/Divide/Multiply/Subtract) and three methods for managing the stack needed in Reverse Polish notation (clear/pop/push).

5.6.2 Hardware controls

Device hardware consists of a single-line display, a bank of push-button keys and a slider switch. The keys consist of a numeric keypad and a few function keys. The slider switch is provided for selecting the notation.

5.6.3 User interface

To simplify the example, the calculator only performs addition, subtraction, multiplication and division:

- the (C) key resets the calculator, and is the only key that still works after an illegal operation such as division by zero
- the (CE) key clears the number that has just been entered
- pressing the (CE) key again in Reverse Polish notation clears the number at the top of stack
- the decimal point is not allowed to be entered more than once when entering a number
- the (+/-) key toggles the sign of the currently displaying number
- pressing the (=) key in Reverse Polish notation adds the currently displaying number to the stack (arithmetic operations are performed when an arithmetic operator key is pressed)
- pressing the (=) key again in standard or algebraic notation repeats the last +, -, * or / operation on the currently displaying number.

Changing notation at any time resets the calculator.

Pressing a key may trigger a number of method calls. For example, pressing the (+) key in Reverse Polish notation to add two numbers involves four method calls: popping the stack twice to obtain two numbers from the stack, calculating their sum, and pushing the result back to the stack.

Figure 5.16 on page 203 shows the part of the user interface that uses the method library to perform calculations in Reverse Polish notation.
Figure 5.16: Performing calculations in Reverse Polish notation (1 of 3)
Figure 5.16: Performing calculations in Reverse Polish notation (2 of 3)
The user interface triggers a method call in the library by setting the name of the method in the Method control. Arg 1 and Arg 2 are two optional arguments passed to the method. Arg return is any argument returned by the method. Call status indicates whether the method call was successful or not, and is used in synchronising the two threads.
5.6.4 Method library

Figure 5.17 shows the code for the library’s Divide method. As well as checking for a division by zero error, this code sets the Last call and Last status controls. These two controls provide visual feedback during the running of the simulation. They are used to hold a record of the last call that the user interface made to the method library, and whether the call was successful or not.

![Diagram of the Divide method in the method library](image)

Figure 5.17: The Divide method in the method library

5.6.5 Using the platform’s API to provide method functionality

If the API of the platform where the Mesh engine was running was used to support the user interface instead of writing methods directly in Meshoil as shown here, not only would the library module be unnecessary but the user interface code would be simpler too since synchronisation issues would not have to be handled. The user interface would simply declare the use of the API methods in the manner shown in Figure 5.18 on page 207 for a Java platform (for details on API method calling, see appendix A.5.1 on page 307).
5.6.6 Error handling

The way that this program is written to handle user errors is a good example of how a Meshoil program split into a number of threads can be functionally integrated.

The user interface demonstrates three levels of error handling. In first-level error handling, the user interface anticipates that a call to a method is going to fail. It avoids making the call and handles the error itself. It displays an error message and allows the user to continue operating the calculator. An example of this is the way in which the user interface avoids attempting a calculation in Reverse Polish notation when this would lead to a stack underflow error. It is able to do this by keeping its own record of how many numbers are in the stack.

In second-level error handling, the user interface does not recognise the error until the method fails. It recognises what has gone wrong from the returned error code, handles the error, displays a message and allows the user to continue operating the calculator. An example of this is stack overflow (stack size is nine). The user interface has no way of knowing the point when insufficient memory will cause the push method to fail. All the user interface can do is to handle the event...
smoothly when it happens.

In third-level error handling, the user interface does not know what has gone wrong because it does not recognise the returned error code. Unable to take the necessary steps to recover from the error, it displays the error code verbatim and takes the safest option of disabling the keypad. This forces the user to reset the calculator by pressing the (C) key. An example of this is division by zero. The user interface could have been written to handle this as a first-level error, but for the purpose of this demonstration it is treated as an unrecognised error. For the same reason, attempting to delete an entry from an empty stack is handled as a second-level, rather than first-level, error.

Figure 5.19 shows the part of the user interface that handles error codes returned after a method call.

Figure 5.19: Handling error codes returned after a method call
5.6.7 Running the 3-in-1 calculator user interface

Figure 5.20 shows a sequenced example of running the 3-in-1 calculator user interface.

Frame 1  The 3-in-1 calculator is ready to be used. The five controls that manage method calling display under the hardware controls in the first window.

The user wants to test the difference between the three notations by doing the same calculation in each. The simple expression: \(1 + 2 \times 3\) should produce a result of 9 in standard and Reverse Polish notations, and 7 in algebraic notation.

Figure 5.20: Running the 3-in-1 calculator user interface (1 of 8)
Frame 2  
Trying Standard notation first, the user enters 1, presses the (+) key, enters 2 and presses the (*) key. 
As soon as the (*) key is pressed, the calculator performs the addition and displays the interim result of 3. Last call in the library window displays $3 = \text{add}(1,2)$ showing that the most recent call made by the user interface was the library's add method. The two arguments 1 and 2 were passed to add and it responded by returning the argument 3. OK in Last status indicates that the call was successful.

Frame 3  
The user completes the calculation by entering 3 and pressing the (=/Enter) key. 
The user interface calls the multiply method to perform the multiplication. The final result of 9 displays. Last call updates to show $9 = \text{multiply}(3,3)$.

Figure 5.20: Running the 3-in-1 calculator user interface (2 of 8)
5.6. 3-IN-1 CALCULATOR

Frame 4  The user switches the Notation to Algebraic.
Switching notation at any time resets the calculator. As part of this reset process, the
user interface makes a call to the clear method to empty the stack of any remaining
contents. Last status shows that the call to this method was successful.

Frame 5  The user repeats the first part of the calculation, entering 1, pressing the (+) key,
entering 2 and pressing the (*) key.
The user interface saves the numbers that have been entered in the 1st number and
2nd number controls, but cannot perform any calculation yet. The still incomplete
multiplication operation will take priority over the addition. The last method to be
used in the library is still clear that was performed when the user selected Algebraic.

Figure 5.20: Running the 3-in-1 calculator user interface (3 of 8)
Frame 6  The user completes the calculation by entering 3 and pressing the (=/Enter) key. As soon as the (=/Enter) key is pressed, the user interface performs the two calculations needed to produce the final result of 7. Last call shows that the second calculation was the addition of 1 and 6—the value of 6 having been obtained by multiplying 2 and 3.

Frame 7  The user switches the Notation to Reverse Polish. Reverse Polish notation requires the use of a stack. The display format changes. Numbers on the Display are now prefixed by a counter showing how many numbers the stack holds. The display of 0: indicates that the stack is currently empty.

Figure 5.20: Running the 3-in-1 calculator user interface (4 of 8)
The user enters 1 and presses the (=/Enter) key to save the number in the stack. As soon as the key is pressed, the user interface calls the push method to add the number to the stack (the stack consists of the nine controls S-1 to S-9).

In error, the user presses the (+) key. This is an invalid operation since only one number has been entered so far.

The user interface detects the error since it has kept a track of how many numbers are in the stack and knows that the current value is still less than two. In this example of first-level error handling, it catches the error in time to avoid making any abortive method calls. It displays the Too few numbers! error message and waits for the user to continue. No calls to the library are made and push(1) in Last call remains unchanged from the previous step.

Figure 5.20: Running the 3-in-1 calculator user interface (5 of 8)
Frame 10  The user enters 2, presses the (=/Enter) key and enters 3. The calculator now holds three numbers entered by the user, two in the stack and one on the Display.

Frame 11  The user presses the (*) key to multiply 2, the number at the top of the stack, with 3, the number just entered on the Display.

The user interface performs three method calls to obtain the interim result of 6. It pops the value 2 at the top of the stack, multiplies it with the currently displaying number of 3, and pushes the result of 6 back to the stack.

Figure 5.20: Running the 3-in-1 calculator user interface (6 of 8)
5.6. 3-IN-1 CALCULATOR

Frame 12  The user presses the (+) key. This time, the user interface needs to make four method calls, since two pops are needed to get two values from the stack before they can be added to produce the final result of 7 that gets pushed back to the stack.

Frame 13  Deciding to clear the stack one number at a time, the user presses the (CE) key twice. The first press simply clears the value of 7 from the Display. With Display now blank, the second press clears the value currently at the top of the stack by calling pop. The stack is now empty.

Figure 5.20: Running the 3-in-1 calculator user interface (7 of 8)
Frame 14 In error, the user presses the (CE) key once more. Attempting to clear numbers from an empty stack is an example of second-level error handling. The call to pop fails, resulting in the return of a null argument and the error code E27: Stack underflow. The user interface recognises this error code and displays the more user friendly message of Nothing to clear! for the user. With the error successfully caught and handled, the user is allowed to continue.

Frame 15 Wanting to see what happens when a number is divided by zero, the user enters 1, presses the (=/Enter) key, enters 0 and presses the (/) key. Divide by zero is an example of third-level error handling by the user interface. When the user interface calls the Divide method with the arguments 1 and 0, the error code E12: Divide by zero is returned. Unable to interpret this code and recover from the error by itself, the user interface displays the error code to the user verbatim and locks the keyboard. This forces user to reset the calculator by pressing the (C) key, risking the loss of any numbers already entered in the stack.

Figure 5.20: Running the 3-in-1 calculator user interface (8 of 8)
5.7 Mobile phone/emergency pager

5.7.1 Overview

This example is a user interface for a communications device that is made to work either as a conventional mobile phone or an emergency pager.

When running as a mobile phone, the user interface emulates the Nokia 6210. The emergency pager is designed to make emergency calling as simple and safe as possible for those with reduced motor and cognitive skills, such as the elderly and disabled.

The example demonstrates the Mesh concept of hardware reuse. The user interface is really two interfaces in one. When running as an emergency pager, the mobile phone handset changes from operating as a Nokia 6210 phone to an easy to operate emergency pager. The existing hardware features of the handset are reused to provide the user with what appears to be quite a different type of device.

The example also demonstrates the Mesh concepts of virtual logic and futures programming, as well as how Meshoil can be used with existing technology to control devices that are not yet Meshable.

5.7.2 Using Meshoil to control non-Meshable devices

Complex electronic devices often support a device-specific language that allows some degree of remote control of the device. For example, manufacturers of GSM mobile phones typically provide their own proprietary interface protocol that allows remote access to basic phone features. Direct use of the protocol is usually restricted to the manufacturer and those granted special licence, but the protocol usually supports a command set available for more general use that allows access to some of these features. This command set is effectively the support of AT commands that offer a fairly standard way of remotely interfacing with any brand of mobile phone.

Such conventional, device-specific languages do not offer the same potential for flexible control that a Meshable device would, but they do provide a means of using Meshoil with existing technology that is not Meshable.

This example takes a Nokia 6210 mobile phone and, as far as possible, rebuilds the user interface in Meshoil to run in a remote handset or equivalent outside the physical Nokia handset. The only part of the user interface that cannot be replaced in this way is the Nokia handset’s fundamental ability to transmit and receive calls, since such hardware features are only accessible to the Meshoil program via AT commands. By modelling the greater part of the user interface in Meshoil, the Nokia handset is effectively reduced to functioning as a simple, low-level communications device that is no longer directly operated by the user.

5.7.3 Running this simulation

This Meshoil program not only runs the user interface but also simulates the operation of a Nokia 6210 handset when working as a communications device under the control of AT commands.

The program runs as two threads. The first thread runs the user interface in a simple handset that allows the user to interact with the communications device. This could either be the shell of another mobile phone handset working as a dumb terminal—used for the convenience of its keypad, earpiece and microphone without any calling capability—or it could be a GUI emulation of a mobile phone. The second thread simulates the Nokia 6210 handset working as the communications device.

All two-way communication between the handset and communications device is by AT command. For example, the handset initiates a voice call by sending the ATD command to the
communications device. It responds to an incoming call by detecting when it has been sent the
CRING command from the communications device. As well as call control, AT commands are used
to find out the current status of the communications device, such as signal quality and battery
strength, to keep the display of such information up-to-date on the user’s handset.

5.7.4 Hardware controls

Hardware controls consist of the screen and keys on a Nokia 6210 handset, controls for audio
hardware such as earpiece, microphone, ringer circuit and volume, and two lines for communication
between the handset and communications device. The screen is split into four addressable areas
to support the keypad’s use of the (\) and (/) selection keys that perform the functions indicated
by the text above them on the screen.

The (End) key has a special function for toggling between Mobile phone and Emergency pager
operation. When the handset is powered up, holding down this key switches the handset off,
toggling the type of device ready for the next time that the handset is powered up.

5.7.5 Phone features

The following features are supported in Mobile phone operation:

Voice and message calling

Voice and message calling works like a Nokia 6210. When entering text, pressing a key works in
the standard way of scrolling the available characters that can be entered by the particular key.

The following send options can be attached to a message: Reply request, Read receipt and
Urgent.

Ring tones, ring volume and vibrate alert

Settings in the menu control which ring tone is used, the volume level and whether the handset
vibrates on an incoming call.

Profiles

There is a choice of three profiles: General, Meeting and Silent. Each contains settings for Ring
tone, Ring volume, Vibrate alert and Call divert that can be customised.

Caller ID

If the number of the caller matches an entry in the phone book, the name of the caller displays
while the phone is ringing.

Caller profile

The mobile phone allows the user to set up profiles for individual callers by linking each phone book
entry to any profile—a feature not supported by the Nokia 6210 but included here to demonstrate
how easy it is to create additional user functionality in Meshoil.

When an incoming call matches an entry in the phone book that has an associated profile, the
Ring tone, Ring volume and Vibrate alert settings for that profile are used instead of the equivalent,
generic settings currently set in the menu.
5.7. MOBILE PHONE/EMERGENCY PAGER

Phone book
The user enters the phone book by selecting Names. Entries in the phone book consist of name, number and any associated profile.

Menu navigation
The (\), (/), (Up) and (Down) keys are used to navigate and backtrack through the menu. The (End) key can be used at any time to escape the menu and return to the Ready screen. A confirmation message briefly displays each time a setting is changed.

PIN security
Entering the wrong PIN three times in a row locks up the phone, requiring the simulation to be restarted.

Key guard
Pressing the (*) key within five seconds of entering the menu switches on the key guard which locks the keypad. Incoming calls can still be received while the key guard is on. To release the key guard, the (*) key must be pressed within five seconds of selecting Unlock.

Welcome message
If a welcome message has been set up in the menu, the message briefly displays each time the phone is switched on.
5.7.6 Software

Figure 5.21 shows a sample of the software that controls the handset.

Figure 5.21: Handling character entry on the mobile phone keypad (1 of 2)
This sample brick handles character entry and what displays on the screen when the user is performing such tasks as entering messages and editing the names of phone book entries. The five labelled bricks that it contains highlight the modular nature of Meshoil programming. Each of these bricks handles the single, clearly identified function described in the label.
One example of this modularity and ability to add detail to a design is the brick called Make pending character blink that forms part of the Display newly entered character brick. A newly entered character remains pending until either character acceptance times out or the user presses a different key. The Make pending character blink brick, as its name suggests, simply causes the pending character to blink. Without it, character entry will still work, but it has been added as a refinement to let the user know when a character has been accepted and the same key can be pressed again to enter another character rather than scroll the existing one.

### 5.7.7 Caller-specific call divert

Being able to link profiles that include a Call divert setting to entries in the phone book means that the phone is able to handle a form of caller-specific call divert, a feature that existing mobile phone networks are incapable of supporting.

In current mobile phone technology, diverting a particular call is managed by the network and not by the handset. When the user changes the call divert setting, the setting is transparently uploaded to the network where it is stored and used to route calls accordingly. Typical settings are ‘Divert all voice calls’, ‘Divert if out of reach’ and ‘Divert if not answered’. The user is not granted any finer control over divert behaviour, such as specifying whether certain incoming calls are to be diverted.

In the example here, caller-specific call divert is supported within the handset itself, despite the limitations of current network technology. First, the user sets the divert setting to Divert if not answered. When an incoming call has an associated profile that is set to Divert all voice calls, the user interface is programmed to reject the incoming call automatically without alerting the user. The network registers that the call has not been answered and diverts the call accordingly, unaware that call rejection was an automatic procedure.

### 5.7.8 Emergency pager features

When operating as the Emergency pager, the handset is designed to be left close at hand for an elderly or disabled user (referred to from here on as the ‘patient’). It no longer operates as an ordinary phone and cannot be used for voice calling. Its purpose is now two-fold: to allow the patient to notify a carer of the need for immediate assistance, and to allow a carer to check that the patient is OK.

The pager’s user interface has been designed to match the patient’s ability, allowing a more able patient to access more of the pager’s features but in a way that does not detract from its use at the simplest level.

**Message book**

The pager has a message book but no phone book. Each entry in the message book holds a preset message and the number to send the message to.

The message book holds five entries with messages such as Alert! Meg Kerby to request immediate hospital assistance, and various coded messages for the carer such as Come urgently!, Please visit and I’m OK now.

**Emergency message sending**

In the simplest mode of operation, the patient is able to send a preset message to a preset phone number by simply picking up the handset and pressing any key on the keypad (except the (Power)
key). The handset does not even need to be on. The first entry in the message book is used in emergency message sending.

The patient can stop the call going ahead if a key was pressed by mistake, since once a key has been pressed, there is a ten second delay before the message is actually sent. During this time pressing any key will abort the send.

**Selective message sending**

If the patient has the cognitive ability to do more than reach for the handset and randomly press a key, the handset can be used to select which message to send from the message book. The patient first switches on the handset by a double key press using any key. The patient then selects which of the numbered message in the message book to send by pressing the key with the corresponding number.

The procedure is no different from emergency message sending, since the double key press that switches the handset on is just the standard way to initiate, then abort, emergency message sending. Since the procedure is no different, there is still the ten second delay before any message is actually sent during which time sending can be aborted.

After no key has been pressed for ten seconds, the pager automatically returns to emergency message sending, ready to send the emergency message regardless of which key is pressed.

**Checkup calling**

The carer can call the patient on the pager to see if they are OK by initiating a checkup call. On receiving a checkup call, the patient indicates that all is well by simply pressing any key.

To trigger a checkup call, the carer sends a message to the pager consisting of the checkup password that is set up in the menu under the **Checkup password** option. The pager only responds to checkup calls, ignoring all incoming message calls where the message does not match this password.

On receiving the checkup call, a message on the screen invites the patient to respond by pressing any key. The handset starts ringing at the current **Ring volume** setting set in the menu. The patient has 20 seconds in which to respond. If no response is made after ten seconds, the ring volume increases to the maximum level. If no response is made after a further ten seconds, the handset sends the message **No response** back to the phone that made the checkup call. If the patient presses any key during the allowed 20 second response time, the message **I am OK!** is sent instead.

**Remote notification of low battery status**

To let the carer know when the pager’s battery is running low, the pager automatically sends a warning message when the battery level drops below 25%. The warning message and the phone number to send to is set up as the fifth message book entry. In the brief period while the message is being sent, the handset cannot be used by the patient and a busy message displays on the screen.

**Administrator mode**

To gain access to the menu for setting up pager operation, the handset needs to be in Administrator mode. This is done by switching the handset on in the usual way for a mobile phone by holding down the (Power) key.

Navigating the menu is just the same as for the mobile phone. The menu holds settings for **Messages** (the message book), **Checkup password** used in checkup calling, **Auto off** for controlling whether the handset automatically switches off when not being used, **Ring tone** and **Ring volume**.
5.7.9 Running the mobile phone/emergency pager user interface

Figure 5.22 shows a sequenced example of running the mobile phone/emergency pager user interface.

Frame 1  The controls in the Handset hardware window are loosely arranged to represent the Nokia 6210 handset. At the top is the (Power) key. This is followed by the screen, the (\textup{\textbackslash{}}) and (/) selection keys, the (Up) and (Down) scroll keys, the (Talk) and (End) call control keys, and finally the numeric keypad. Underneath this is the group of audio hardware controls, followed by four controls that represent the two communication lines between the handset and communications device.

The Communications device window represents the simulated, non-Meshable Nokia 6210 handset when used just as communications device.

The handset is currently switched off. The communications device is powered up but its status is IDLE showing that it is not currently transmitting or receiving a call from the network.

Figure 5.22: Running the mobile phone/emergency pager user interface (1 of 10)
5.7. MOBILE PHONE/EMERGENCY PAGER

Frame 1  The Handset user interface window displays controls used in both Mobile phone and Emergency pager operation. A typical example is the ‘- PIN’ control that holds the current PIN needed to switch on the handset when used as a phone. Controls that begin with Mes hold details of message book entries. Controls that begin with Nam hold details of phone book entries.

Frame 2  The simulation starts with the handset working as a Mobile phone. The user switches the handset on by holding down the (Power) key (simply ‘pressing’ (Power) by a single mouse click does nothing, the button on the mouse has to be held down).

The screen for PIN entry displays and the user starts entering the 4-digit PIN which echoes with asterisks.

Frame 3  The user completes entry of the PIN and presses the (\) key for OK. The PIN acceptance message briefly displays. Entering the wrong PIN would have resulted in an error message and the user having to enter the PIN again. After three incorrect tries, the phone displays the message Phone now disabled. PUK required and locks up, requiring the simulation to be restarted.

Frame 4  Since a welcome message has been set up in the menu, it briefly displays too.

Figure 5.22: Running the mobile phone/emergency pager user interface (2 of 10)
Chapter 5. Case Studies

Frame 5

The Ready screen displays, indicating that the handset is now ready to make or receive calls. The two options for the selection keys are Menu, at the bottom left of the screen to enter the menu, and Names, at (what would be if possible!) the bottom right of the screen to enter the phone book.

The top of the screen displays the current signal quality and battery strength in the communications device. Their actual values are held in the Battery % and Signal (0-31) hardware controls shown in the Communications device window, but the handset can only determine what they are by sending AT commands to the communications device and waiting for it to respond.

The handset stores the latest values obtained from the communications device in its own % Battery and % Signal controls shown in the Handset user interface window. Before the handset was switched on and the communications device interrogated, these controls were blank and undefined (shown in Frame 1). The handset chooses to store these values as percentages. Since the AT command for obtaining signal quality returns values in the range 0 to 31, the values obtained from the communications device need to be converted to their equivalent percentages.

Signal quality and battery strength information is repeatedly updated in the handset by commands sent and received on Line-2.

Figure 5.22: Running the mobile phone/emergency pager user interface (3 of 10)
In the simulation, all communication between the threads representing the handset and communications device is by information embedded in AT commands. Two lines are needed, since Line-1 needs to be kept free for asynchronous sending of AT commands involved in call handling.

The % Signal value currently stands at 100%, displayed to the user as four bars at the top left of the screen. This indicates that the last reading from the communications device was the maximum value of 31 (-51 dBm or greater).

This frame shows the start of the signal quality update cycle: the handset has just output the CSQ command on Line-2 out. This will be detected by the communications device which will return the current signal quality on Line-2 in. The response will be CSQ:14, representing the current signal quality of 14. This indicates that signal quality has dropped to less than 50% (shown in Frame 12) since the last cycle. When the handset receives the response, the number of bars on the screen will drop to two (shown in Frame 13).

The update cycle is similar for battery strength which currently stands at 70%, equivalent to three bars on the screen. The handset alternates between sending CSQ commands for signal quality, and CBC commands for battery strength. The two update cycles run automatically and continuously whenever the handset is powered up and operating, whether as Mobile phone or Emergency pager.

The user wants to send the message Meet you at 5pm to Angela, who is set up as an entry in the phone book. The user starts the procedure by pressing the (\) key to select Menu.

The Messages option displays at the top level of the menu.

The user presses the (\) key to select Messages. The Write message option displays in the Messages submenu.
CHAPTER 5. CASE STUDIES

Frame 8  The user presses the (\) key to select **Write message**.
The character entry screen displays (this screen represents the sample of software shown in Figure 5.21 on page 220). *abc* at the top left of the screen indicates that lower case characters are currently selected. The top right of the screen shows the number of characters that can still be entered for the message, the maximum being 160.
The user starts to enter the message, using the (#) key to switch between upper and lower case characters as necessary.

Frame 9  On completing the message, the user presses the (Talk) key.
The number entry screen displays.

Frame 10  The user presses the (\) key to select **Names**.
The first entry in the phone book displays.

Figure 5.22: Running the mobile phone/emergency pager user interface (5 of 10)
The user presses the (\) key to select OK and send the message (alternatively, the (Talk) key could have been pressed here).

The sending screen displays. The handset outputs the ATCMGS command on Line-1 out to instruct the communications device to send the message. The communications device decodes the number and message embedded in the ATCMGS command, and starts transmitting to the network at 9600 bps, as indicated by the change in its status to CONNECT 9600.

Figure 5.22: Running the mobile phone/emergency pager user interface (6 of 10)
CHAPTER 5. CASE STUDIES

Frame 12  On completion of the transmission, the communications device returns to IDLE status and sends the OK response to the handset on Line-1 in. The handset, informed that transmission was successful, briefly displays Message sent.

Frame 13  The Ready screen displays once again.

Figure 5.22: Running the mobile phone/emergency pager user interface (7 of 10)
5.7. MOBILE PHONE/EMERGENCY PAGER

Frame 14  The user, with no more calls to make and wanting to change the handset from Mobile phone to Emergency pager, switches the handset off by holding down the (End) key instead of the (Power) key.
A message indicating the change of device briefly displays before the handset powers down and the screen goes blank. The handset is now ready to be used as an Emergency pager.

Frame 15  The user, now in the role of carer, leaves the handset by the bedside of the patient. The carer does a double-key press, using any key, to switch the pager on so that it will be ready to receive a checkup call later on.
The screen shows that the pager is on and ready for emergency message sending. At any time the patient can press a key at random that will send a preset message to the local medical facility to request immediate assistance.

Figure 5.22: Running the mobile phone/emergency pager user interface (8 of 10)
Some time later away from the patient, the carer decides to do a checkup call to make
sure that the patient is OK. The carer uses another phone to send the coded message meg

(To simulate an incoming call, the message and number are directly entered in the fields
for the Message and Number controls displaying in the Communications device window.
Each entry needs to be completed by pressing the Enter key on the computer running
the simulation. Entering the Number triggers the communications device to process
the incoming call. This frame shows the result of having entered meg in Message and
83039919 in Number, and having waited for this entered information to be detected by
the communications device.)

After downloading the message from the network, the communications device sends the
CRING command on Line-1 in to inform the handset that an incoming call needs to be
processed. The handset sends the CLCC command on Line-1 out to request information
about the incoming call.

The communications device responds with the CNMI command (as shown in this frame).
The handset recognises the message embedded in the CNMI command as matching the
setting in the Password control that was set up in the Check password menu option. It
starts the phone ringing according to the settings of the Ring vol and Ring tone controls.

Figure 5.22: Running the mobile phone/emergency pager user interface (9 of 10)
5.7. MOBILE PHONE/EMERGENCY PAGER

Frame 16 (cont) The screen displays Are you OK? and starts counting down the 20 seconds that the patient has to press any key to respond.

Had the message in the incoming call not matched ‘meg’ held in Password, it would simply have been ignored. Had the call been a voice call, it would automatically have been rejected by the handset.

Frame 17 After 10 seconds, the ring volume increases to the maximum level and a more urgent message displays on the screen. The patient responds just in time before the 20 seconds elapses by randomly pressing a key.

The handset starts the procedure for sending the message I am OK! to the phone that made the checkup call. If the user had not pressed a key, the message No response would have been sent.

Figure 5.22: Running the mobile phone/emergency pager user interface (10 of 10)

5.7.10 Profiles supported by virtual logic

In Meshoil, virtual logic means changing the logic of how a program runs without editing code (see section 3.6.3 on page 58). Virtual logic is demonstrated in this example with the profiles feature.

Being modelled on the Nokia 6210, this user interface supports the use of profiles. But what if this was not the case and the user interface had been written without incorporating any profile functionality? One solution made possible by Meshoil’s plug and play design approach would be to download to the handset a new version of the user interface which did include profiles. But even this is unnecessary if virtual logic is used, as this example shows.

The user interface includes a single brick called Profiles that acts like a plugin, providing all the code needed to support the profiles feature. This is despite the fact that profiles are closely interwoven with other features of the user interface. For example, when profiles are used, the phone’s menu structure is different, phone book entries can be associated with particular profiles, and the incoming call procedure changes if the caller matches a phone book entry that has an associated profile.

In a conventional language, it would be impossible to isolate all the profiles-related code at a single location in this way and be able to switch profiles on and off by simply choosing whether or not to execute an additional block of code. But the Profiles brick, acting like a software overlay, makes this possible.
CHAPTER 5. CASE STUDIES

Figure 5.23 shows the start of the Profiles brick. This brick executes when the setting for the Device control is Mobile phone.

Figure 5.23: The Profiles brick supporting everything to do with profiles

To demonstrate a phone without profiles, Device in the Hardware hardware window can be manually changed from Mobile phone to Phone-no profiles while the handset is switched off. This causes execution of the Profiles brick to be bypassed when the handset is powered up, having the effect of removing all profiles-related code from the user interface. To achieve the same effect in a conventional language, all the relevant bits of code spread out over the user interface program would need to be edited out individually.

The code in the Profiles brick interacts with the user interface code by the process of path hijack and release. For example, when the user interface is running without profiles, the menu offers the
5.7. MOBILE PHONE/EMERGENCY PAGER

user the choice of two main options: Messages and Settings. Pressing the (Down) key cycles through these options repeatedly. The Profiles brick converts this to a choice of three options: Messages, Settings and Profiles. It does this by hijacking the logic at the point that pressing the (Down) key would have selected Messages after Settings. By causing Profiles to be selected instead, the existing menu-handling code is effectively switched off because it has not been written to recognise Profiles as an option. Having hijacked the path of logic, the Profiles brick is now free to steer the user down menu pathways of its own.

Whenever the user returns to the point where path hijack occurred and presses the (Down) key to scroll to the next option, the Profiles brick releases the path by returning to Messages. This effectively switches the existing menu-handling code on again because it recognises the Messages option.

5.7.11 Futures programming as the solution for incoming call options

The Profiles brick also demonstrates the Mesh concept of futures programming—writing programs that anticipate the future by controlling hardware that has yet to exist (see section 3.4.2 on page 46).

Code in the Profiles brick allows the user to set up profiles that specify whether or not incoming calls should be indicated by the handset vibrating. These profiles change the setting of a control called Vibrator alert that is used during an incoming call to switch on the Vibrator hardware control that represents the physical unit for making the handset vibrate.

Profile settings for changing the ring tone and ring volume work correctly because the handset already has the necessary circuitry installed for these features, represented by the Ringer and Volume controls. But the handset shown in this example doesn’t have an installed vibrator unit. The user interface runs perfectly well, switching the Vibrator hardware control on and off at the right times, but nothing happens as Vibrator has no physical association with any existing hardware.

The user interface is thus pre-programmed to manage all situations in which a vibrate alert is used during an incoming call. It’s just that the final piece of the puzzle is missing—the vibrator unit itself. Any time the unit is installed in the handset, and Vibrator is added to the with brick that lists the hardware controls, the feature will work as intended without anything further needing to be done.
CHAPTER 5. CASE STUDIES