Lafros GUI-App: a monitoring and control-oriented Scala-Swing application framework

Rob Dickens

```
// Latterfrosken
software.development(limited);
```

Walsall, West Midlands, UK

Contents

- 15 mins Slides
- 10 mins Demo
- 5 mins Answers

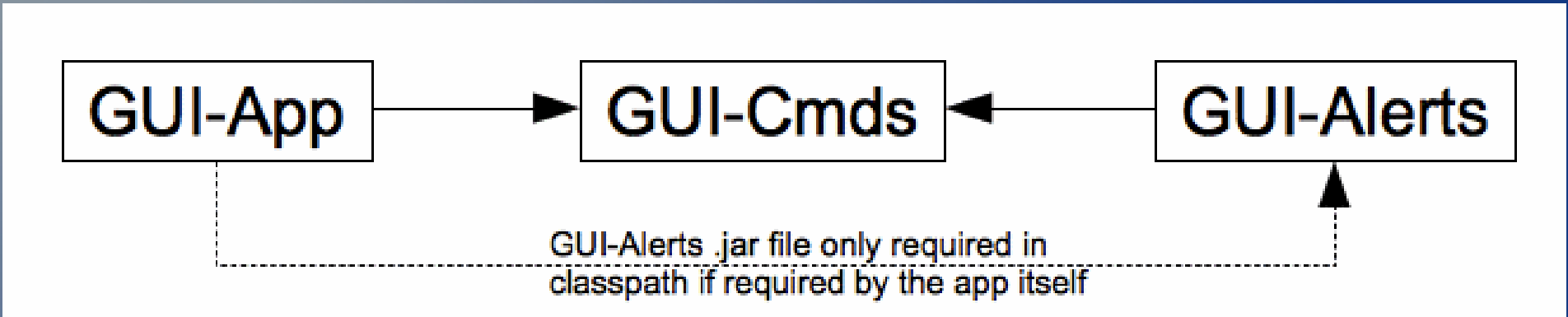
What this software is for

- intended to simplify writing monitoring and control-oriented user interfaces for the desktop (in Scala)
  - intend to use it to write the user interface for an impl'n of the Lafros MaCS remote monitoring and control API
- however, ought to be worth considering when writing any kind of desktop user interface

How it came to be written

- 1995: began writing C++/Motif framework as part of User Monitoring and Control module for EISCAT Svalbard Radar
- 1998: began writing Java version
- 2002: dev't of Java version continued inde-pendently => JUICe libraries
- 2009: JUICe libraries rewritten in Scala => Lafros GUI-App, GUI-Cmds, GUI-Alerts

A framework depending on two sub-frameworks



- GUI-App: deploy same code as application or applet
- GUI-Cmds: define units of code to be invoked interactively
- GUI-Alerts: monitor changeable values interactively

Deploy same code as application or applet

```scala
object app extends App {
  def init(context: Context) {...}
}
```

```
<applet code="com.lafros.gui.app.Applet"
 ...
 <param name="App" value="org.myorg.myapp.app">
</applet>
```

- only need supply an init() method – default impl'ns are provided for the other methods the framework uses: displayApplication(), start(), stopApplet(), restartApplet(), terminate()
- all will be called from a java.awt.EventQueue dispatch-thread

Define your GUI Cmds

- functions of type, Unit => Option[String]
  - return an optional feedback message
- define by extending Cmd trait, or one or more of its sub-traits
  - CheckFirstCmd: requests confirmation
  - EventDependentCmd: supplies the Event
  - PwdProtectedCmd: requests password
  - SeqBgCmd: executed in the background
  - TogCmd: flips toggle upon return

Executing GUI Cmds

- executed indirectly, via an Exer
- associate an AbstractButton via the Trig mix-in

```
val myCmd = new Cmd {
  def apply() = { // exceptions will be caught
    ...
    Some("useful thing done")
  }
}
val myCmd2 = Cmd {
  ...
  Some("useful thing done")
}
...
val myExer: Exer = Exer(myCmd)
val togExer: Tog.Exer = Exer(togCmd)
val togExer2: Tog.Exer = Exer(setCmd, resetCmd)
myExer.executeCmd() // guaranteed to return,
...                 // immediately
```

```
val but = new Button with Trig {
  exer = myExer // also sets cmd property
}
val but2 = new Button with Trig {
  cmd = myCmd // also sets exer property
}
val togBut = new ToggleButton with Trig {
  // button only toggles if togCmd returns
  exer = togExer
}

val but = new Button with Trig {
  cmdReaction = { // exceptions caught
    case ActionEvent(_) =>
      ...
    Some("useful thing done")
  }
}
```

```
// configure multiple Trigs
// in one place
val trigProps = new Trig.Props {
  title = "toggle"
  title0 = "set toggle"
  title1 = "reset toggle"
}
val but3 = new Button with Trig {
  action = trigProps
  exer = togExer
}
val but4 = new ToggleButton with Trig {
  action = trigProps
  exer = togExer
}
```

GUI-Cmds benefits

- common functionality provided for you
- conditional toggles
  - only flip if Cmd returns
- robust
  - exceptions caught
- promote user feedback
  - feedback and exception messages relayed via TheCmdsController to, for example, a Gui-App's MsgLine

The GUI-Alerts MonField

- a scala.swing.Label, specialised for displaying values that are updated, and to which the user's attention might need to be drawn
- has alert property, with the following values:
  - NoAlert: normal background colour
  - NonIntrusive: red background
  - Intrusive: alternating background accompanied by alert sound
  - Acknowledged: red background

Other MonField properties

- valueToAlert: Any => Alert
- value: Any
  - text = value.toString
  - alert = valueToAlert(value)
- templateText: String
  - determines width (except when "")

GUI-App/Scala vs JUICe/Java: usage 1

+ the declarative style afforded by Scala's properties makes GUI code far easier to understand

```
val a = new A {
  b = new B {
    c = new C {
      ...
    }
  }
}
```

```
C c = new C();
B b = new B();
b.setC(c);
A a = new A();
a.setB(b);
```

GUI-App/Scala vs JUICe/Java: usage 2

- less work to define GUI Cmds, by virtue of their default method impl'ns
- None preferable to null when GUI Cmds return no feedback message
- the main method required of applications is provided for you

GUI-App/Scala vs JUICe/Java: framework impl'n

+ whereas JUICe.cmds had to supply a corresponding class for every AbstractButton, GUI-Cmds needed supply only a single mix-in, Trig
+ was able to implement SeqBgCmds in GUI-Cmds without recourse to synchronized blocks, resulting in code which is straightforward to reason about

GUI-App/Scala vs JUICe/Java: deployment

- dependency on scala-library.jar and scala-swing.jar has implications for downloadable apps (applets or Java Web Start applications)
  - can extract only those classes which are actually required, using e.g. ProGuard – see http://lafros.com/maven/plugins/proguard

Availability

- Maven project on github
- GPL or purchase per-app exception
- see http://lafros.com/gui for further details...