Sustainable Internet of Things Computing

Mart Lubbers[0000−0002−4015−4878] and Pieter Koopman[0000−0002−3688−0957]

Institute for Computing and Information Sciences,
Radboud University, Nijmegen, The Netherlands
firstname@cs.ru.nl

1 Exercises

1.1 Hello world!

The first program anyone writes when trying a new programming language is the so-called *Hello World!* program. While the program only prints *Hello World!* to the screen and exits, it allows you to verify that the toolchain is working. Typical mTask devices, e.g. microcontrollers, only have a very simple 1×1 monochrome screen, the built in LED. Therefore, the best we can do is let the world know that the program is working by turning this screen on or off. On the other hand, mTask's toolchain is a lot more elaborate.

First the Clean program representing the iTask application is compiled. During the execution of the resulting program, mTask devices may be connected. Once connected, an mTask is compiled at runtime to a specialised byte code that is sent to the device dynamically. In turn, the device interprets the byte code to create a task tree, a runtime representation of a task-oriented program. Step by step, the task tree is rewritten, yielding an observable value at every step. This observable value, and information about the used shared data sources (SDSs) is communicated with the server.

Listing 1.1 shows the complete iTask part of the *Hello World!* program. After the module heading and the imports, line 8 contains the standard way of starting an iTask engine with task `main` as the argument. The `main` task shows the main iTask task. First, the device information is asked from the user (line 11). With this information, `withDevice` is used to connect the device (line 12). The second argument of the `withDevice` function, the body, just lifts the `blink` mTask task to an iTask task (line 18). This sets the machinery in motion described above and results in the mTask device executing the program. To allow the user to prematurely end the program (and gracefully disconnect the device), line 13 uses the `>>*` combinator to add a button to the user interface to terminate the task.

```
1  module helloworld
2
3  import StdEnv, iTasks
4  import mTask.Interpret, mTask.Interpret.Device.TCP
5  import Device
6
```

```
7  Start :: !*World → *World
8  Start w = doTasks (main <<@ ApplyLayout frameCompact) w
9
10 main :: Task ()
11 main =         enterDeviceInfo
12     >>? \spec→withDevice spec deviceTask
13     >>* [ OnAction (Action "Stop") (always (shutDown 0))
14         , OnAction (Action "Reset") (always main)
15         ]
16 where
17     deviceTask :: MTDevice → Task ()
18     deviceTask dev = liftmTask blink dev
```

Listing 1.1. *Hello World!*, the iTask part

The `blink` task shown in listing 1.2 first defines the NeoPixel peripheral on line 20. Then a helper function is defined that converts a boolean to a suitable light intensity value (line 21). The light intensity can range from 0 to 255[1]. The lines 22 to 25 define the blinking function. The blinking function takes one argument, the current state and performs several actions combined using the sequence operator (`>>|`). Using this argument it first sets the first pixel on the shield to the correct value using the `b2i` helper (line 23). Then it waits for 500 ms (see line 24). Finally, it calls itself recursively with the inverse of the state (line 25) to achieve the blinking behaviour. The `main` record denotes the main expression, in here, the `blinkfun` is simply called. The `lit` function transforms a Clean value to a literal in mTask.

```
19 blink :: Main (MTask v ()) | mtask, NeoPixel v
20 blink = neopixel neopixelWemosRGBLEDShield \neo→
21     fun \b2i=(\b→If b (lit 10) (lit 0))
22     In fun \blinkfun=(\st→
23               setPixelColor neo (lit 0) (b2i st) (b2i st) (b2i st)
24         >>|. delay (ms 500)
25         >>|. blinkfun (Not st)
26     ) In {main=blinkfun true}
```

Listing 1.2. *Hello World!*, the mTask part

Use `cloogle.org` to lookup details about functions in Clean, iTask and mTask.

1.2 Colors

The mTask system integrates with iTask using only three integration functions, simplified types of these are shown in listing 1.3. `withDevice` is used to connect an mTask device to an iTask server so that tasks can be executed. `liftmTask` is used to compile, send and execute a task on a device, it lifts an mTask task to an iTask task. Finally, `lowerSds` is used to connect iTask SDSs to mTask SDSs.

[1] Remember to put on sunglasses before ramping the brightness to 255.

Exercise 1 Hello World! file: `helloworld.icl`

Compile and run your first mTask program (see `README.html` for instructions on how to install Clean, iTask and mTask).
On Linux or through a visual studio code *devcontainer* you run in a shell:

```
nitrile build --only=helloworld
./helloworld
```

On windows you run in a PowerShell:

```
nitrile build --only=helloworld
.\helloworld
```

Once your program is up and running, navigate to `localhost:8080` to see the iTask interface. Here you enter the IP address shown on the OLED screen of the device in the *Host:* field. Leave the *Port* and *Ping timeout* fields for what they are. When you have pressed continue, the middle LED of the RGB LED shield should blink to tell you hello!
Hint: *Try changing the frequency or position of the LED.*

```
withDevice :: TCPSettings (MTDevice → Task b) → Task b | ···
liftmTask :: (Main (MTask BCInterpret u)) MTDevice → Task u | ···
lowerSds :: ((v (Sds t)) →In (Shared sds t) (Main (MTask v u)))
    → Main (MTask v u) | ···
```

Listing 1.3. Integration functions of mTask with iTask.

While tasks have an observable task value, some collaboration patterns benefit a lot from the *many-to-many* communication SDSs offer. As SDSs from iTask can be accessed by mTask, the powerful web editor infrastructure of iTask can be used to communicate with mTask tasks.

Exercise 2 Colors file: `colors.icl`

Use `lowerSds` to access the color values that are stored in the iTask SDS. See the lowering of `redShareI` for an example.
Then read the SDSs in mTask and provide the values to `setPixelColor`.
Hint: *The* `>>~` *combinator is used because* `getSds` *always yields an* unstable *value.*

1.3 Walking

As seen in listing 1.4, the interface to the NeoPixel peripheral only contains one function besides the constructor. This function, `setPixelColor` requires five arguments. The first argument is the handle to the peripheral, obtained by defining it using the constructor. The second argument is the index of the LED you want to address. The RGB LED shield contains seven LEDs so indices 0 to 6 are valid here. The other arguments are the RGB color components.

```
class NeoPixel v where
    neopixel      :: NeoInfo ((v NeoPixel) → Main (v b)) → Main (v b)
    setPixelColor :: (v NeoPixel) (v Int) (v Int) (v Int) (v Int)
        → MTask v ()
```

Listing 1.4. NeoPixel mTask interface.

Exercise 3 Walking file: `walking.icl`

Fill in the gaps in the `walk` mTask task so that instead of blinking one LED, it it walks through the full range of LEDs.
First create `blinkonce` mTask function by inserting the correct `setPixelColor` calls. Then implement the `walkfun` mTask function that iterates over the LEDs and calls `blinkonce` for each one.
Hint: *There are* 7 *LEDs on the board so the valid indices range from* 0 *to* 6.

1.4 Sensors

Attached to the microcontroller is a digital temperature and humidity (DHT) sensor, sht30x, that connects via I^2C to the board. The interface is shown in listing 1.5. The constructor (`dht`) works very similar to the NeoPixel constructor. `temperature` yields the temperature as an unstable value in °C. `humidity` yields the relative humidity as an unstable value in %.

```
class dht v where
    dht :: DHTInfo ((v DHT)→Main (v b)) → Main (v b)
    temperature :: (v DHT) → MTask v Real
    humidity :: (v DHT) → MTask v Real
```

Listing 1.5. DHT mTask interface.

Exercise 4 Temperature file: `temperature.icl`

The program consists of two functions. `measureTemp` measures the temperature and sets the color of the pixel according to the limits. `setColor` reads the lowered SDSs and determines the color of the pixel. Blue if it is too cold, green if it's within limits and red if it's too hot. Implement the logic for setting the pixel colors and observe the behaviour[2].
Hint: *Use the conditional statement* `If` *and one of the comparison operations to calculate the color. In mTask, all arithmetic operations are suffixed with a* `.` *(a period), e.g.* `+.`, `-.`, `*.`, `/.`, `>.`, `<.`, `>=.`, `<=.`. *The* if*-expression is written with the* `If` *function. Furthermore, literals must always be lifted so* 100 *in mTask is written as* `lit 100`.

[2] If you look for a challenge, you can also implement a gradient.

The SGP30 is an air quality sensor that is connected to the WEMOS D1 mini as an expansion shield. It communicates via I^2C to the mainboard and can report equivalent CO_2 measurement (in ppb) and a total voltile organic compounds measument (in ppm).

The interface is shown in listing 1.6.

```
class AirQualitySensor v where
    airqualitySensor :: AirQualitySensorInfo
        ((v AirQualitySensor) → Main (v a)) → Main (v a) | ···
    setEnvironmentalData :: (v AirQualitySensor) (v Real) (v Real)
        → MTask v ()
    tvoc :: (v AirQualitySensor) → MTask v Int
    co2 :: (v AirQualitySensor) → MTask v Int
```

Listing 1.6. Air quality sensor mTask interface.

As the air quality is not bound to change very quickly, the default refresh rate of airquality tasks is one minute. To increase this for the purpose of the assignment, we use the `co2`` variant that takes a refresh interval parameter.

Exercise 5 Air quality — file: `airquality0.icl`

The skeleton program contains an iTask system that measures the equivalent CO_2 and places this, when it is changed, in a SDS. Typical air quality sensors need some time to warm up. The measurement will start at the minimum (400 ppm) and this will, after a thirty minutes or so to the actual measurement. When observing the terminal, you see that after the sensor has warmed up enough, the measurement fluctuates very quickly. Breathing on the sensor will also show immediate results.
Adapt the `differs` function so that only values that differ more than ϵ ppm are reported.
Hint: `differs` *is a Clean function, the host language will inline the code. The first argument is a Clean value so you have to use* `lit` *here.*

As we have seen before, the device is equipped with an SHT3X temperature and humidity sensor. Most air quality sensors can be made more accurate by feeding it with the current temperature and humidity using the `setEnvironmentalData` ↪ task.

Exercise 6 Environment — file: `airquality1.icl`

Adapt the program so that it sets the environmental values before measuring the air quality.
Hint: *Use the correct sequential combinator.* `temperature` *and* `humidity` *yield unstable values so use* `>>~`.

Running multiple tasks on mTask devices is as simple as combining tasks with one of the parallel combinators. Listing 1.7 shows the types of the two

parallel combinators. Use `.&&.` if you want to combine the values of the two tasks. Use `.||.` if you are only interested in one of the values.

```
class (.&&.) infixr 4 v :: (MTask v a) (MTask v b) → MTask v (a, b) | ···
class (.||.) infixr 3 v :: (MTask v a) (MTask v a) → MTask v a      | ···
```

Listing 1.7. Parallel task combinators in mTask.

Exercise 7 Changing environments — file: `airquality2.icl`

Adapt the program so that it the environmental value is set each time either the humidity or the temperature changes.
To give some feedback to the user that the environmental values have changed, blink the LED after setting the environmental values.

A Solutions

Solution 1 Hello World!

See listings 1.1 and 1.2.

Solution 2 Colors

```
color :: Int → Main (MTask v ()) | mtask, lowerSds, NeoPixel, AirQualitySensor v
color lednumber = neopixel neopixelWemosRGBLEDShield \neo→
       lowerSds \red=redShareI
    In lowerSds \grn=grnShareI
    In lowerSds \blu=bluShareI
    In {main=rpeat (
                delay (ms 100)
        ≫|.     getSds red
        ≫~. \r→getSds grn
        ≫~. \g→getSds blu
        ≫~. \b→setPixelColor neo (lit lednumber) r g b
    )}
```

Solution 3 Walking

```
walk :: Main (MTask v ()) | mtask, NeoPixel v
walk = neopixel neopixelWemosRGBLEDShield \neo→
    fun \blinkonce=(\i→
             setPixelColor neo i level level level
        ≫|. delay (ms 1000)
        ≫|. setPixelColor neo i off off off
    ) In fun \walkfun=(\i→
        If (i ==. lit 7)
            (walkfun (lit 0))
            (blinkonce i ≫|. walkfun (i +. lit 1))
    ) In {main=walkfun (lit 0)}

off = lit 0
level = lit 10
```

Solution 4 Temperature

```
temperatureTask :: Main (MTask v ()) | mtask, lowerSds, dht, NeoPixel v
temperatureTask =
    neopixel neopixelWemosRGBLEDShield \neo→
    dht dhtWemosSHT30Shield \dht→
       lowerSds \highTempM=highTempI
    In lowerSds \lowTempM=lowTempI
    In lowerSds \curTempM=curTempI
    In fun \setColor=(\x→
                getSds lowTempM
        ≫~. \l→getSds highTempM
        ≫~. \h→setPixelColor neo (lit 0)
            (If (x >. h)             (lit 50) (lit 0))
            (If (x >=. l &. x <=. h) (lit 50) (lit 0))
            (If (x <. l)             (lit 50) (lit 0))
    ) In fun \measureTemp=(\old→
             temperature dht
        ≫~. \x→setSds curTempM x
        ≫|. setColor x
        ≫|. delay (ms 250)
        ≫|. measureTemp x
    ) In {main=measureTemp (lit 0.0)}
```

Solution 5 Airquality

```
    differs :: Int (v Int) (v Int) → v Bool | mtask v
    differs eps old new =
        If (old >. new)
            (old -. new >. lit eps)
```

Solution 6 Environment

```
    ) In {main=
                temperature dht
        ≫~. \t→humidity dht
        ≫~. \h→setEnvironmentalData aqs t h
        ≫|.     getSds airqualityShareM
        ≫~. \v→measureAirquality (lit 0)
    }
```

Solution 7 Changing environments

```
airqualitymTask :: Main (MTask v ()) | mtask, lowerSds, dht, NeoPixel, AirQualitySensor v
airqualitymTask =
    airqualitySensor airqualitySensorWemosSGP30Shield \aqs→
    dht dhtWemosSHT30Shield \dht→
    neopixel neopixelWemosRGBLEDShield \neo→
    lowerSds \airqualityShareM=airqualityShareI
    In fun \calibrate=(\(oldtemp, oldhumid)→
             temperature dht .&&. humidity dht
        >>*. [IfValue (tupopen \(t, h)→differsTH (t, h) (oldtemp, oldhumid)) rtrn]
        >>=. tupopen \(t, h)→setEnvironmentalData aqs t h
        >>|. setPixelColor neo (lit 1) (lit 50) (lit 50) (lit 50)
        >>|. delay (ms 200)
        >>|. setPixelColor neo (lit 1) (lit 0) (lit 0) (lit 0)
        >>|. calibrate (t, h)
    ) In fun \measureAirquality=(\old→
             co2` (BeforeSec (lit 1)) aqs
        >>*. [IfValue (\x→differs 50 old x) (\nv→setSds airqualityShareM nv)]
        >>=. \nv→delay (ms 200)
        >>|. measureAirquality nv
    ) In {main=
             temperature dht
        >>~. \t→humidity dht
        >>~. \h→setEnvironmentalData aqs t h
        >>|.     getSds airqualityShareM
        >>~. \v→measureAirquality (lit 0) .||. calibrate (t, h)
    }
where
    differsTH (newT, newH) (oldT, oldH)
        =  differs 1.0  newT oldT
        |. differs 10.0 newH oldH

    differs eps l r = If (l >. r) (l -. r >. lit eps) (r -. l >. lit eps)
```