Green Computing for the Internet of Things

Lecture 7

Mart Lubbers Pieter Koopman
mart@cs.ru.nl pieter@cs.ru.nl

Sustrainable 2022 summer school
Rijeka, Croatia
July 7th, 2022

Radboud University

Part I

Internet of Things

Internet of Things

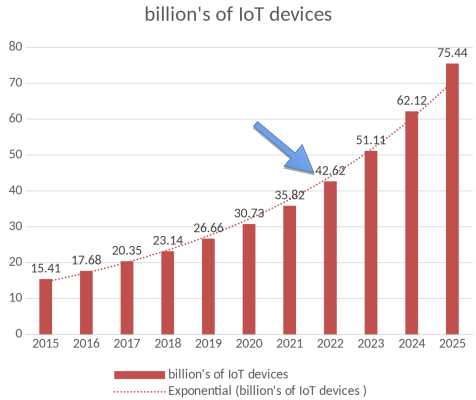
IoT edge devices

Programming IoT devices using C++

Internet of Things

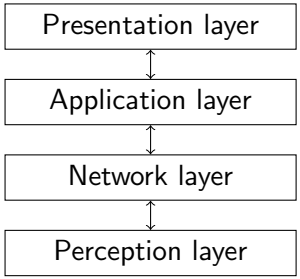
[0]https://statinvestor.com/data/33967/iot-number-of-connected-devices-worldwide/

Internet of Things



[0]https://statinvestor.com/data/33967/iot-number-of-connected-devices-worldwide/

IoT Architecture



Web browsers, mobile/desktop apps, javascript, html, css...

Servers, databases, caches, storage, high level languages...

MQTT, Zigbee, Bluetooth, WiFi, LoRa, ...

Microprocessors, SBCs, sensors, C/C++ code, ...

Types of IoT edge devices

Single board computers

Photo by Micheal H. ("LaserLicht") (CC BY-SA 4.0)

- 30×65mm
- 0.5–8GiB RAM
- 32GiB storage
- 1–1.5GHz clock
- 70€
- Raspbian OS (Linux)
- **4–6W**

Microprocessors

Photo by Tpkull (CC BY-SA 4.0)

- 34×25mm (14×25mm)
- 50KiB (0.00005 GiB) RAM
- 4MiB (0.004 GiB) storage
- 80MHz (0.08GHz) clock
- 5€
- no OS (sometimes FreeRTOS)
- **0–0.4W**

Power consumption of IoT edge devices

Why lower energy?

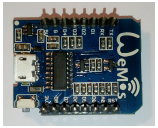
- Battery powered nodes
- Sustainability: no 5W 70€ board in a 10W LED light bulb.
- Sheer number of devices...
  - If all IoT devices were raspberry pi's: $5\text{W} \times 24 \times 356 \times 40 \cdot 10^9 = 1.7\text{PWh/year}$.
  - Annual worldwide electrical production: 27 PWh/year (IoT needs 6%).
  - Dutch nuclear power plant: 3.3TWh/year

(IoT needs 515 of those).

Use microprocessors

Sleeping modes of microprocessors

ESP8266



item	active	modem sleep	light sleep	deep sleep
WiFi radio	on	**off**	off	off
CPU	on	on	**pending**	**off**
RAM	on	on	on	**off**
current used	100–240mA	15mA	0.5mA	0.002mA

Sleeping modes of microprocessors

Adafruit Feather M0 Wifi

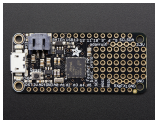


Photo by Adafruit industries (CC BY-NC-SA)

item	active	modem sleep	idle	deep sleep
WiFi radio	on	**off**	off	off
CPU	on	on	**idle**	**idle**
RAM	on	on	on	**low power**
current used	90–300mA	5mA	2mA	0.002mA

RAM is saved in deep sleep on this microprocessor

Highly model specific

Sleep more → consume less

Easier said than done

Sleep in practice

1. The dogs have to be fed
   - WiFi needs periodic execution time
   - No threading support
2. Some tasks may not be interrupted
   - Some task rely on timing
   - 1-Wire, I^2C, SPI, USB, WiFi, PWM, ...
3. We do not know the other tasks
   - Tasks scheduling may depend on runtime
   - Tasks may be uploaded dynamically

Hello world!

Blinking an LEDtwo LEDS

```
const int ledPin    = D4;
const int interval = 500;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);
  delay(interval);
  digitalWrite(ledPin, LOW);
  delay(interval);
}
```

```
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(otherPin, OUTPUT);
}

void blink(int pin, int interval) {
  digitalWrite(pin, HIGH);
  delay(interval);
  digitalWrite(pin, LOW);
  delay(interval);
}

void loop() {
  blink(ledPin, interval);
  blink(otherPin, 2500);
}
```

Solve scheduling

Manual interleaving

.

```
long *lr1 = 0;      long *lr2 = 0;
bool *st1 = false; bool *st2 = false;

void setup() { ... }

void blink(int pin, int interval, long *lastrun, bool *state) {
  if (millis() - *lastrun > interval) {
    digitalWrite(pin, *state);
    *st = !(*state);
    *lastrun += interval;
  }
}

void loop() {
  blink(ledPin1, 500,  &lr1, &st1);
  blink(ledPin2, 2500, &lr2, &st2);
}
```

Interrupts

Polling

```
const int pir = D3;
const int led = D4;

void setup() {
  pinMode(pir, INPUT_PULLUP);
  pinMode(led, OUTPUT);
  digitalWrite(led, HIGH);
}

void loop() {
  digitalWrite(led,
    digitalRead(pir));
  delay(100);
}
```

Interrupts

```
volatile bool changed = false;
IRAM_ATTR void isr() { ... }

void setup() {
  ...
  attachInterrupt(..., isr, ...);
}

void loop() {
  delay(60000); // or even light/deep sleep
  if (changed) {
    changed = false;
    digitalWrite(led, digitalRead(pir));
  }
}
```

Solve scheduling

Compiler/OS support

- A sufficiently smart compiler™[1] can generate this interleaving.
  Difficult to determine when it's safe.
  Impossible to do if the scheduling depends on runtime values.
- Use OS that supports preemptive multithreading
  Requires multithreading annotations (`wait`, `yield`, `mutex`, `lock`).
  Requires library support.
  Increases the hardware requirements.
- ...

Task-oriented programming!

[1] https://wiki.c2.com/?SufficientlySmartCompiler

Part II

Task-oriented programming

Task-oriented programming

Programming in iTasks

Task-oriented programming for the IoT

Programming in mTask

Task-oriented programming (TOP)

SUS Trainable

Coordinate collaboration between people and machines to reach common goal.

Components

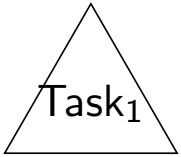
- Basic tasks: input/output (e.g. web editors, sensors)
- Composition: sequential, parallel
- Communication: task results, shared data sources

Implementations

iTask	Generates a multi-user web application from the TOP specification to do the work.
$\widehat{TOP}$	Formal calculus for tasks including several semantics.
mTask	TOP language and ecosystem for microcontrollers.

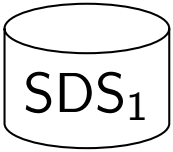
Tasks

- Represent the actual work
- Observable value during evaluation, or no value
- Value may change over time
- Event based rewriting
- Task combinators
  - Parallel
  - Sequential
  - Conditional
- Collaboration and interaction



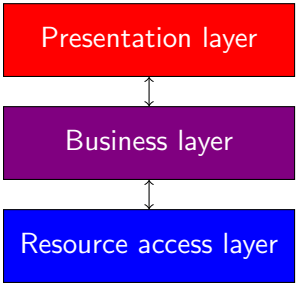
Shared Data Source (SDS)

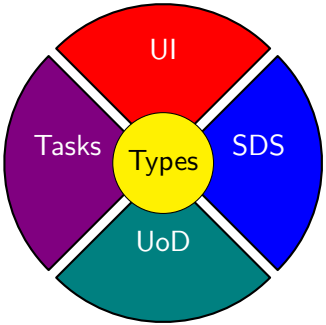
- Read/write interface over arbitrary data
- Memory, databases, files, time, etc. . .
- Share combinators



Task-oriented software development (TOSD)

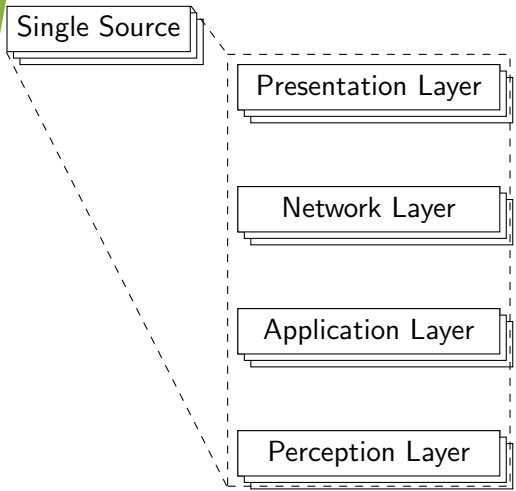
Tierless programming





Tierless Architectures

What is a tierless architecture?



Tiered architecture

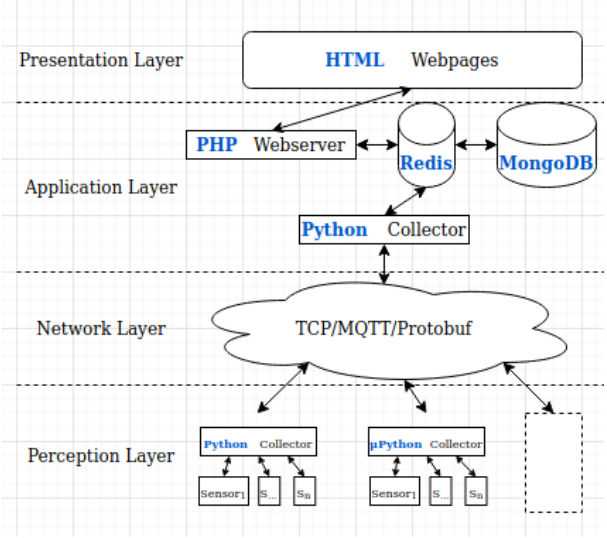
- Layered architecture
- Separate sources per layer
- Differences between layers
- Semantic friction

Tierless architecture

- Layered architecture
- Separate sources per layer
- Generated from a single source
- Hop/Links/Haskino/Potato...
  iTask/mTask.

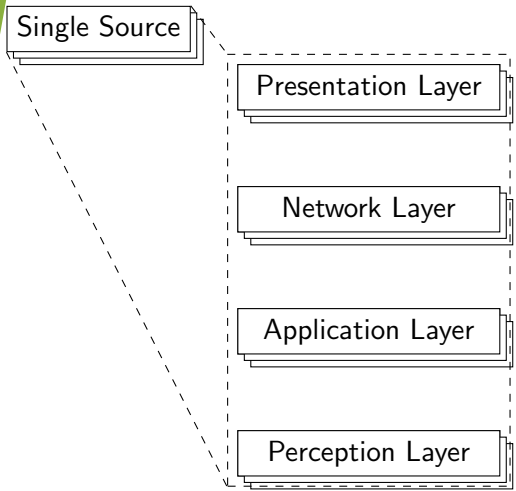
Tierless Architectures

Example architecture



Tierless Architectures

What is a tierless architecture?



Tiered architecture

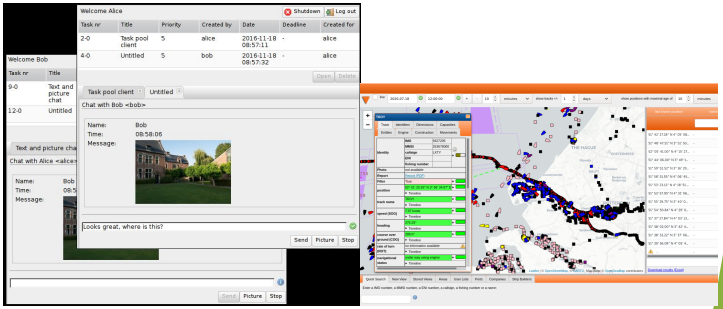
- Layered architecture
- Separate sources per layer
- Differences between layers
- Semantic friction

Tierless architecture

- Layered architecture
- Separate sources per layer
- Generated from a single source
- Hop/Links/Haskino/Potato...
  iTask/mTask.

The iTask system

- Long history
- Clean server
- Javascript client
- Client-side execution
- Distributed execution
- Automatic UI generation
- Used in industry



Programming in iTasks

Basic tasks

- enter typed data: `enterInformation`
- update given data: `updateInformation`
- display data: `viewInformation`

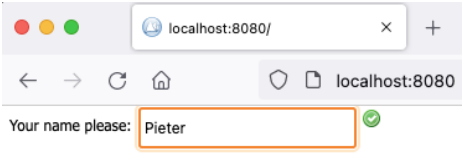
See cloogle.org for all definitions in Clean.

```
module itask1

import iTasks

Start world = doTasks nameTask world

nameTask :: Task String
nameTask = enterInformation []
         <<@ Label "Your name please"
```



Task Values

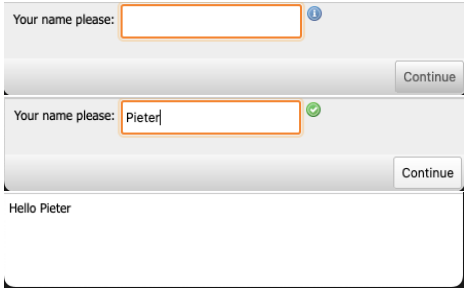
```
nameTask :: Task String
nameTask = enterInformation [] <<@ Label "Your name please"
```

We can update the value in the editor as long as we have it

- Empty editor **NoValue**
- Data entered **UnStable** value

`>>?` makes `Continue` button to progress when left-hand side has a value

```
helloTask :: Task String
helloTask =
  nameTask >>? \name →
  viewInformation [] ("Hello " +++ name)
```



Programming in iTasks

Conditional step ≫∗: add a list of continuations

```
greeter :: Task String
greeter =
    nameTask ≫∗
        [OnAction (Action "Done")  (ifValue (\n → size n > 2) (\n → return n))
        ,OnAction (Action "Clear") (always greeter)
        ,OnValue  (ifValue (\name→size name > 9) \n → return n)
        ]
        ≫- \name → viewInformation [] ("Hello " +++ name)
```

Programming in iTasks

SUS Trainable

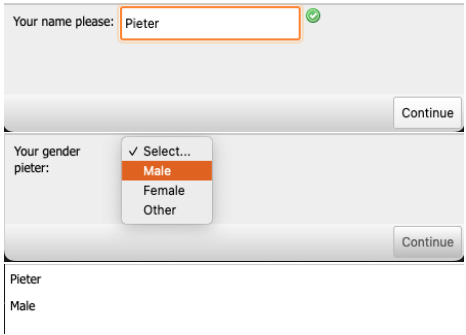
User defined datatypes

```
:: Gender = Male | Female | Other
derive class iTask Gender
```

The **derive class** `iTask` makes all iTask magic available for type `Gender`

Combining values

```
nameAndGender :: Task (String, Gender)
nameAndGender =
  nameTask >>? \name →
  (enterInformation [] <<@
    Label ("Your gender " +++ name)) >>?
      \gender →
  viewInformation [] (name, gender)
```



Programming in iTasks

Running tasks in parallel

- Need both tasks `-&&-`
- Waits until subtasks are **Stable**, yields **Stable** result

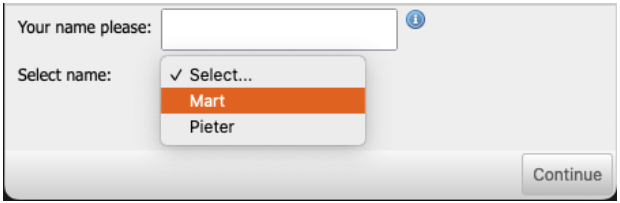
```
nameAndGender2 :: Task (String, Gender)
nameAndGender2 = (enterInformation [] -&&- enterInformation []) >>?
    viewInformation []
```

Programming in iTasks

Running tasks in parallel

- First task finished determines result `-||-`
- Waits until first subtask is **Stable**, yields **Stable** result

```
nameTask2 :: Task String
nameTask2 =
  (nameTask -||-
  (editChoice [] ["Mart","Pieter"] ?None <<@ Label "Select name")) >>? \name →
   viewInformation [] name
```

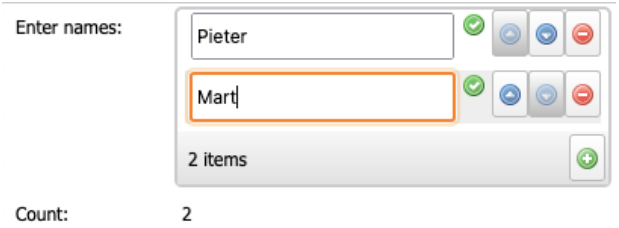


Programming in iTasks

Shared Data Sources

- Task communication via SDS
- Automatic update on changes of SDS

```
sharedNames :: Task [String]
sharedNames =
    withShared [] \sds →
    (updateSharedInformation [] sds <<@ Label "Enter names") -||
    (viewSharedInformation [ViewAs length] sds <<@ Label "Count")
```



Programming in iTasks

Persistent Shared Data Sources

```
nameGenderSDS :: SimpleSDSLens [(String, Gender)]
nameGenderSDS = sharedStore "sdsIdentfier" initialValue
where initialValue = []
```

Programming in iTasks

SUS Trainable

Reading combinator types

type	behaviour
`return :: a→Task a`	Lifts a value to a task value
`(-&&-)` **infixr** `:: (Task a) (Task b)→Task (a,b)`	Task results are combined
`(-||-)` **infixr** `:: (Task a) (Task a)→Task a`	Disjunction of parallel subtasks
`(-|| )` **infixr** `:: (Task a) (Task b)→Task a`	Returns left result
`( ||-)` **infixr** `:: (Task a) (Task b)→Task b`	Returns right result
`(≫?)` **infix** `:: (Task a) (a→Task b)→Task b`	Sequentially, rhs takes result of task

There are many more useful operators
cloogle.org knows them all by name and type!

Programming in iTasks

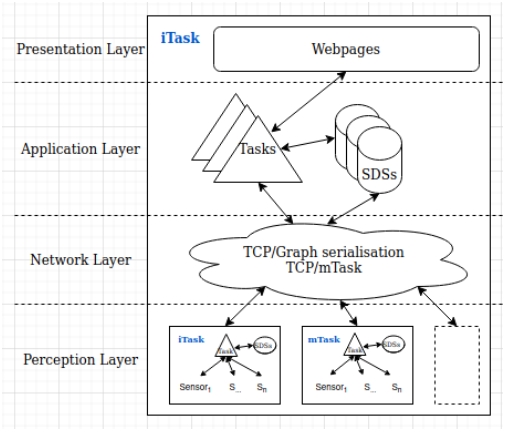
SUS Trainable

Recap

- iTasks enables consize specification of tierless web-systems
- Generates web-server and all communication
  - Code can run in the browser (web assembly)
  - We can specify a multi-user program in iTasks
  - Program can even be multi server
- A web-server is too big and heavy for an IoT node
  Such a web-server is not needed for an IoT node
- We made a TOP language for the IoT embedded in iTask

mTask

Tasks on tiny computers



Properties

- Tagless final embedding
- Type-safe, task-oriented
- Abstractions for peripherals
- RTS for many microprocessors
- Tailor-made at runtime
- Interpreted on the device
- Multi tasking support
- Integrated in iTasks

mTask/iTask Interface

- Interact with the device
- Language
- Lift SDSs
- Lift tasks

iTasks Interface

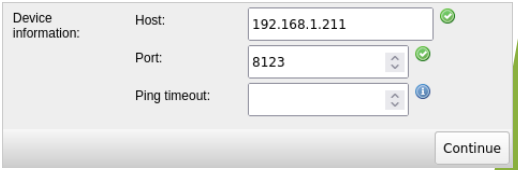
Interact with the device

Definition

```
withDevice :: a → (MTDevice → Task b) → Task b | iTask b & channelSync a
:: MTDevice //Opaque
```

Example

```
main :: Task t
main = enterTCPInfo
  >>= \x→withDevice x ( \dev →···)
where
  enterTCPInfo :: Task TCPSettings
  enterTCPInfo = enterInformation []
    <<@ Label "Enter TCP settings"
```



mTask/iTask Interface

Language (1)

Clean source code

```
blink :: Int → Main (MTask v Bool) | mtask v
blink wait = declarePin D4 PMOutput \d4→
  fun \blinkfun=(\x→
        delay (lit wait)
    >>|. writeD d4 x
    >>|. blinkfun (Not x))
  In {main=blinkfun true}
```

mTask expression

- Bytecode compiler
- Pretty printer
- Symbolic simulation
- Resource analysis†
- ...

mTask/iTask Interface

Structuro of a task

Language

```
someTask :: MTask v Int | mtask v
someTask =
  sensor1 config1 \sns1→
  sensor2 config2 \sns2→
     fun \fun1= (···)
  In fun \fun2= (···)
  In {main=mainexpr}
```

Classes

```
class mtask v | expr v & sds v & pinMode v &···


class expr v where
  lit :: t → v t | ..
  (+.) infixl 6 :: (v t) (v t) → v t |···
 ···
```

More on this later

iTasks Interface

Lifting SDSs

Definition

```
class liftsds v where
  liftsds :: ((v (Sds t)) → In (Shared sds t) (Main (MTask v u)))
    → Main (MTask v u) |··· & RWShared sds & ···
```

Example

```
sdsI :: SimpleSDSLens Int
sdsI = sharedStore "some identifier" 42

someTask :: MTask v Int | mtask, liftsds v
someTask = ···
  liftsds \sdsM →
  In {main= getSds sdsM >>=. \x → ··· }
```

iTasks Interface

Lifting tasks (1)

Definition

```
liftmTask :: (Main (MTask BCInterpret u)) MTDevice → Task u |···
```

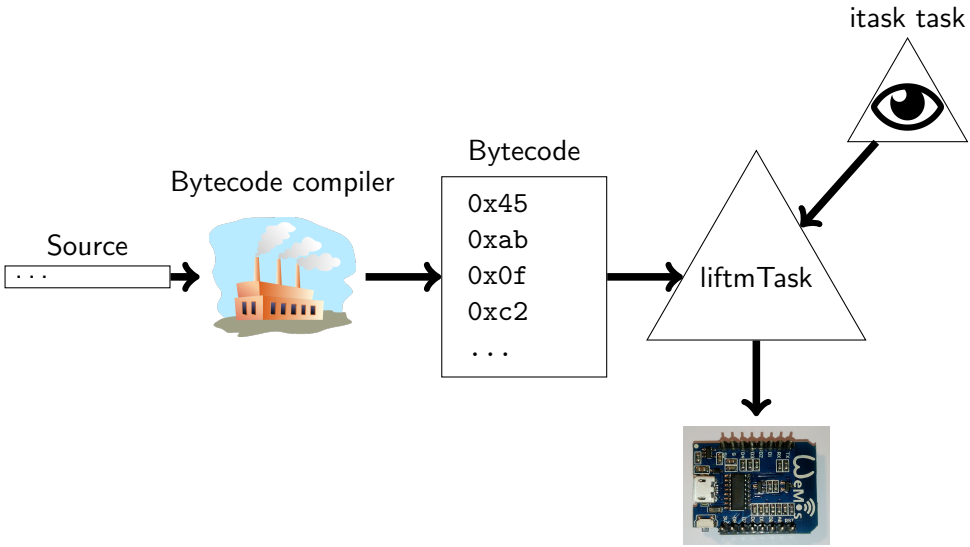
Example

```
taskI :: Task t
taskI = enterTCPInfo >>= \x→withDevice x ( \dev → liftmTask taskM dev )
where
  enterTCPInfo :: Task TCPSettings
  enterTCPInfo = enterInformation [] <<@ Label "Enter TCP settings"

  taskM :: Main (MTask v t) | mtask v
  taskM =
      ···
    In {main=mainexpr}
```

iTasks Interface

Lifting tasks (2)



mTask/iTask Interface

Language (3): Expressions

Definition

```
class arith v where
  lit :: ···
  (+) :: ···
  ··· :: ···
```

Examples

```
c42 :: v Int | mtask v
c42 = lit 6 *. lit 7

cond :: v Int | mtask v
cond = If (lit 42 ==. c42) (lit 38) (lit 2 *. (lit 18 +. lit 1))
```

mTask/iTask Interface

Language (3): Data types

mTask	C/C++	stack cells
`Bool`	`bool`	1
`Char`	**char**	1
`Int`	`int16_t`	1
`Long`	`int32_t`	2
`Real`	**float**	2
`(a, b)`	**struct** ?	$a + b$
`(a, b, c)`	**struct** ?	$a + b + c$
`:: T = A | B | C`	**enum**	1

```
:: DPin = D0 | D1 |···
:: APin = A0 | A1 |···
:: PinMode = PMOutput | PMInput |···
```

mTask/iTask Interface

Language (3): Functions

Definition

```
class fun a v where
  fun :: ···
```

Examples

```
add42 :: v Int | mtask v
add42 =
     fun \add= (\(x, y)→x +. y)
  In {main = add (lit 38, lit 4) }
```

```
fib10 :: Main (v Int) | mtask v
fib10 =
     fun \fibacc= (\(n, acc)→
      If (n ==. lit 0)
        ( acc )
        ( fibacc (n -. lit 1, acc *. n) ))
  In fun \fib   = (\n→fibacc n (lit 1))
  In {main = fib (lit 10)}
```

mTask/iTask Interface

Language (3): Basic tasks

Definition

```
class rtrn v where
  rtrn :: ···
```

Examples

```
return42 :: MTask v Int | mtask v
return42 = rtrn (lit 6 *. lit 7)
```

Definition

```
class delay v where
  delay :: ···
```

Examples

```
wtsecond :: MTask v Int | mtask v
wtsecond = delay (lit 500 *. lit 2)
```

mTask/iTask Interface

Language (3): Sensors

Definition

```
class dht v where
  DHT         :: ···
  temperature :: ···
  humidity    :: ···
```

Examples

```
readTemp :: Main (MTask v Real) | mtask, dht v
readTemp =
  DHT (DHT_SHT (i2c 0x45)) \dht→
  {main=temperature dht}
```

mTask/iTask Interface

Language (3): GPIO tasks

Definition

```
class dio p v | pin p where
  writeD :: (v p) (v Bool) → MTask v Bool
  readD  :: ···

class aio v where
  writeA :: ···
  readA  :: ···

class pinMode v
  pinMode    :: ···
  declarePin :: ···
```

Examples

```
blinkTask :: Main (MTask v Real) | mtask v
blinkTask = declarePin D4 PMOutput \ledPin→
  {main = writeD ledPin true} // or (lit True)
```

mTask/iTask Interface

Language (3): Sequential task combinators

Definition

```
class step v where
  (>>*.) :: (MTask v t) [Step v t u] → MTask v u

:: Step v t u
  = IfValue  ((v t) → v Bool) ((v t) → MTask v u)
  | IfStable ((v t) → v Bool) ((v t) → MTask v u)
  |...


aover42 :: Main (MTask v Int) | mtask v
aover42 = declarePin A3 \apin→
  {main=readA apin >>*. [IfValue (\x→x >. lit 42) (\x→rtrn x)]}

(>>=.) l r = l >>*. [IfStable (\_→true) (\x→r x)]
(>>|.) l r = l >>*. [IfStable (\_→true) (\_→r  )]
(>>~.) l r = l >>*. [IfValue  (\_→true) (\x→r x)]
(>>..) l r = l >>*. [IfValue  (\_→true) (\_→r  )]
```

mTask/iTask Interface

SUS Trainable

Language (3): Parallel task combinators

Definition

```
class .||. v where
  (.||.) :: (MTask v a) (MTask v a) → MTask v a
class .&&. v where
  (.&&.) :: (MTask v a) (MTask v b) → MTask v (a, b)
```

Or example

```
waitorst :: Main (MTask v Int) | mtask v
waitorst = declarePin A3 PMInput \ain→
  {main= wait -||- step }
where wait =   delay (lit 60000)
          ≫|. rtrn (lit 0)
      step = readA ain ≫*. [IfValue
                 (\x→x >. lit 42)
                 (\x→rtrn x)]
```

And example

```
readAD :: Main (MTask v (Bool, Int))
       | mtask v
readAD
  = declarePin A3 PMInput \ain→
    declarePin D3 PMInput \din→
  { main= readD din -&&- readA ain }
```

mTask/iTask Interface

SUS Trainable

Language (3): Shared data sources

Definition

```
class sds v where
  sds    :: ···
  getSds :: ···
  setSds :: ···
  updSds :: ···
class liftsds v where
  liftsds :: ···
```

Local example

```
lcl :: Main (MTask v Int)
lcl = sds \share=42
  In {main=getSds share}
```

Lifted example

```
someShareI :: SimpleSDSLens Bool // from iTask

task :: MTask v Int | mtask, liftsds v
task = liftsds \someShareM=someShareI
  In { main=getSds someShareM >>*. [···] }
```

mTask/iTask Interface

Language (3): Green computing

Scheduling frequency

```
:: TimingInterval v
  = Default
  | BeforeMs (v Int) | BeforeSec (v Int)
  | ExactMs  (v Int) | ExactSec  (v Int)
  | RangeMs  (v Int) (v Int)
  | RangeSec (v Int) (v Int)
```

Examples

//temperature‘, readA‘, readD‘, ...

```
tmin = temperature` (BeforeSec (lit 60))
  >>*. [···]
```

Interrupts

```
class interrupt v where
  interrupt :: ···

:: InterruptMode
  = Change  | Rising
  | Falling | Low
  | High
```

Examples

```
pir = declarePin D3 \pir→
  {main=interrupt pir high >>=. \_→ ··· }
// or (lit High)
```

Arduino examples in mTask

Blink

```
const int ledPin   = D4;
const int interval = 500;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);
  delay(interval);
  digitalWrite(ledPin, LOW);
  delay(interval);
}
```

```
interval :: Int
interval = 500

blinktask :: Main (MTask v Int) | mtask v
blinktask = declarePin D5 \ledPin→
  fun \blink= (\st→
          writeD ledPin st
    >>|. delay (lit interval)
    >>|. blink (not st))
  {main=blink false}
```

Arduino examples in mTask

Blink multiple LEDs

Arduino

```
void setup() {
  pinMode(ledPin, OUTPUT);
}
void blink(int pin, int interval) {
  ...
}
void loop() {
  blink(ledPin, interval);
  blink(otherPin, 2500);
}
```

mTask

```
interval = 500

blinktask :: Main (MTask v Int) | mtask v
blinktask =
  declarePin D5 \ledPin→
  declarePin D9 \otherPin→
  fun \blink= (\(pin, int, st→
         writeD pin st
    ≫|. delay int
    ≫|. blink (not st))
  {main =  blink (ledPin,   lit 500,  false)
      -||- blink (otherPin, lit 2500, false)
  }
```

Arduino examples in mTask

Interrupts

Arduino

```
const int pir = D3;
const int led = D4;

volatile bool changed = false;
IRAM_ATTR void isr() { ... }
void setup() { ...
  attachInterrupt(..., isr, ...)
}
void loop() {
  delay(60000); // or even light/deep sleep
  if (changed) {
    changed = false;
    digitalWrite(led, digitalRead(pir));
  }
}
```

mTask

```
pir = PIR D3 \pir→
  declarePin D4 \led→
  fun \mdec = (\()→
        interrupt change pir
    ≫=. \x→writeD led x
    ≫|. mdec ()
  ) In {main = mdec () }
```

Wrap up

- Sleep more: use less power
- Programming IoT stacks is difficult
- Using a tierless language simplifies it greatly

We are hiring:

pieter@cs.ru.nl

Exercises

Go to this url for all the files:
https://tinyurl.com/greeniot

File	What to do with it
`exercises.zip`	The exercise skeletons[2]
`exercises.pdf`	The exercise handout
`slides.pdf`	The slides

Remember:

- https://cloogle.org for all the language documentation
- Make sure you are on the same wifi network as the device:
  ssid: `sustrainable22-{almere,nijmegen}`, password: `greeniot`

[2] Also includes setup instructions for Clean/iTask/mTask