mTask

Mart Lubbers & Pieter Koopman
{mart,pieter}@cs.ru.nl

Radboud University

17th June 2019

Schedule

TOP

iTasks

mTask

Architecture

Thermostat

Task Oriented Programming (TOP)

Concept

Coordinate collaboration between people and machines to reach common goal.

Task Oriented Programming (TOP)

Concept

Coordinate collaboration between people and machines to reach common goal.

Components

Declarative paradigm:

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

Task Oriented Programming (TOP)

Concept

Coordinate collaboration between people and machines to reach common goal.

Components

Declarative paradigm:

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

Implementation

iTasks Generates a multi-user web application from the TOP specification to do the work.

Task Oriented Programming (TOP)

Concept

Coordinate collaboration between people and machines to reach common goal.

Components

Declarative paradigm:

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

Implementations

iTasks Generates a multi-user web application from the TOP specification to do the work.

$\widehat{TOP}$ Formally calculus for tasks including operational semantics.

mTask TOP language and ecosystem for microcontrollers.

Tasks

- Model collaboration and interaction

Tasks

- Model collaboration and interaction
- Represents the actual work

Tasks

- Model collaboration and interaction
- Represents the actual work
- Observable value during evaluation

Tasks

- Model collaboration and interaction
- Represents the actual work
- Observable value during evaluation
- Task can emit no value

Tasks

- Model collaboration and interaction
- Represents the actual work
- Observable value during evaluation
- Task can emit no value
- Event based rewriting

Tasks

- Model collaboration and interaction
- Represents the actual work
- Observable value during evaluation
- Task can emit no value
- Event based rewriting
- Automatically divide up work in slices:

Tasks

- Model collaboration and interaction
- Represents the actual work
- Observable value during evaluation
- Task can emit no value
- Event based rewriting
- Automatically divide up work in slices:
- {i,m}Tasks use an optional stability to model side effects:

Tasks

- Model collaboration and interaction
- Represents the actual work
- Observable value during evaluation
- Task can emit no value
- Event based rewriting
- Automatically divide up work in slices:
- {i,m}Tasks use an optional stability to model side effects:

Tasks

- Model collaboration and interaction
- Represents the actual work
- Observable value during evaluation
- Task can emit no value
- Event based rewriting
- Automatically divide up work in slices:
- {i,m}Tasks use an optional stability to model side effects:

$$\begin{tikzcd}
\mathit{NoValue} \arrow[r, leftrightarrow] \arrow[rr, bend left] & \mathit{Unstable} \arrow[r] & \mathit{Stable}
\end{tikzcd}$$

Shared Data Sources

- Share data between tasks

Shared Data Sources

- Share data between tasks
- Models — possibly impure — data

Shared Data Sources

- Share data between tasks
- Models — possibly impure — data
  - Files
  - Memory
  - Randomness
  - Introspection in the host
  - Time

Shared Data Sources

- Share data between tasks
- Models — possibly impure — data
  - Files
  - Memory
  - Randomness
  - Introspection in the host
  - Time
- Lean and mean publish subscribe system

iTasks

What is iTasks

- DSL in Clean

What is iTasks

- DSL in Clean
- TOP for the web

What is iTasks

- DSL in Clean
- TOP for the web
- Heavily depends on:

What is iTasks

- DSL in Clean
- TOP for the web
- Heavily depends on:
  - Polytypic functions (generics)

What is iTasks

- DSL in Clean
- TOP for the web
- Heavily depends on:
  - Polytypic functions (generics)
  - Dynamic typing (dynamics)

What is iTasks

- DSL in Clean
- TOP for the web
- Heavily depends on:
  - Polytypic functions (generics)
  - Dynamic typing (dynamics)
- Generates a multi-user web application from the specification

What is iTasks

- DSL in Clean
- TOP for the web
- Heavily depends on:
  - Polytypic functions (generics)
  - Dynamic typing (dynamics)
- Generates a multi-user web application from the specification
- Support for distributed operation

What is iTasks

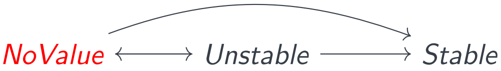
- DSL in Clean
- TOP for the web
- Heavily depends on:
  - Polytypic functions (generics)
  - Dynamic typing (dynamics)
- Generates a multi-user web application from the specification
- Support for distributed operation
- Limited support for peripherals

Basic Tasks

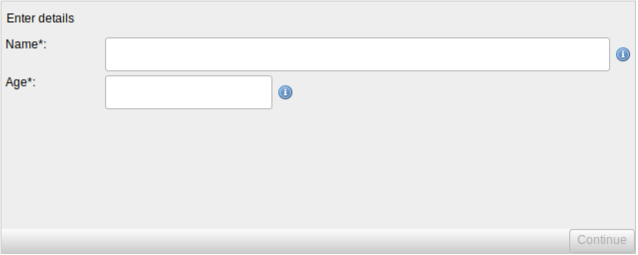
```
return :: a → Task a | iTask a

enterInformation  :: d [EnterOption m]        → Task m | toPrompt d & iTask m
updateInformation :: d [UpdateOption m m] m → Task m | toPrompt d & iTask m
viewInformation   :: d [ViewOption m]     m → Task m | toPrompt d & iTask m


:: ViewOption a  =∃v: ViewAs    (a → v)            & iTask v
                 |∃v: ViewUsing  (a → v) (Editor v) & iTask v
:: EnterOption a =∃v: EnterAs    (v → a)            & iTask v
                 |∃v: EnterUsing (v → a) (Editor v) & iTask v
:: UpdateOption a b
  =∃v: UpdateAs    (a → v) (a v → b)            & iTask v
  |∃v: UpdateUsing (a → v) (a v → b) (Editor v) & iTask v
```

Example Task

```
:: Person = {name :: String, age :: Int}
derive class iTask Person

personTask :: Task Person
personTask = enterInformation "Enter details" []
         >>= viewInformation  "Hello" []
         >>= return
```



NoValue ⟷ *Unstable* ⟶ *Stable*



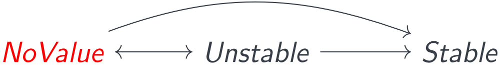
Enter details

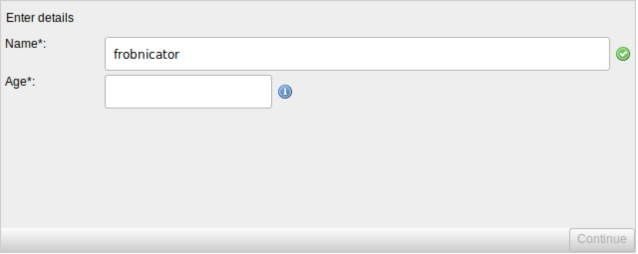
Name*:

Age*:

Example Task

```
:: Person = {name :: String, age :: Int}
derive class iTask Person

personTask :: Task Person
personTask = enterInformation "Enter details" []
        >>= viewInformation  "Hello" []
        >>= return
```





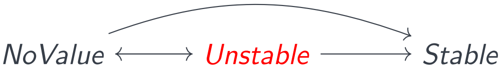
Enter details

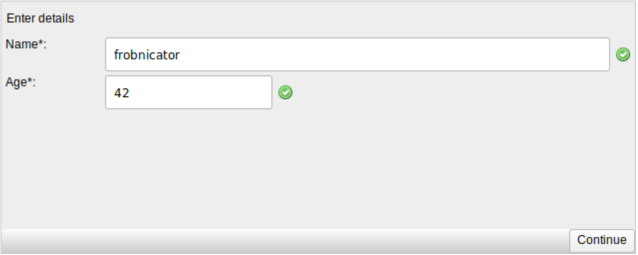
Name*: frobnicator ✓

Age*:

Example Task

```
:: Person = {name :: String, age :: Int}
derive class iTask Person

personTask :: Task Person
personTask = enterInformation "Enter details" []
         >>= viewInformation  "Hello" []
         >>= return
```



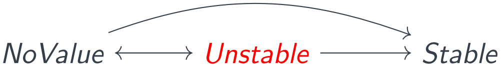

Enter details

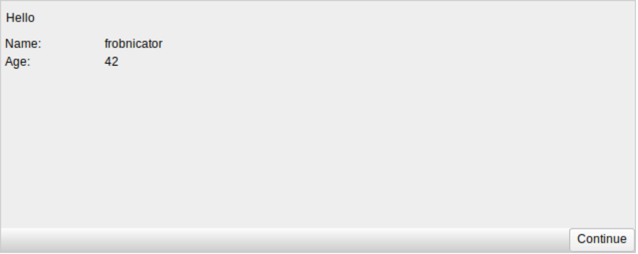
Name*: frobnicator

Age*: 42

Example Task

```
:: Person = {name :: String, age :: Int}
derive class iTask Person

personTask :: Task Person
personTask = enterInformation "Enter details" []
         >>= viewInformation  "Hello" []
         >>= return
```

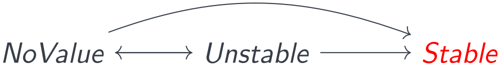



Hello

Name: frobnicator
Age: 42

Example Task

```
:: Person = {name :: String, age :: Int}
derive class iTask Person

personTask :: Task Person

personTask = enterInformation "Enter details" []
          >>= viewInformation  "Hello" []
          >>= return
```



Combinators

Parallel Combinators

```
(-&&-) infixr 4 :: (Task a) (Task b) → Task (a,b) | iTask a & iTask b
(-|| ) infixl 3 :: (Task a) (Task b) → Task a     | iTask a & iTask b
( ||-) infixr 3 :: (Task a) (Task b) → Task b     | iTask a & iTask b
(-||-) infixr 3 :: (Task a) (Task a) → Task a     | iTask a
```

Combinators

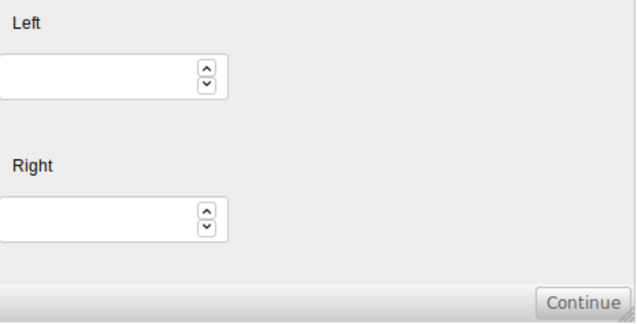
Parallel Combinators

```
t :: Task Int
t = enterInformation "Left" [] -||- enterInformation "Right []
  >>= viewInformation "Result" []
```

Combinators

Parallel Combinators

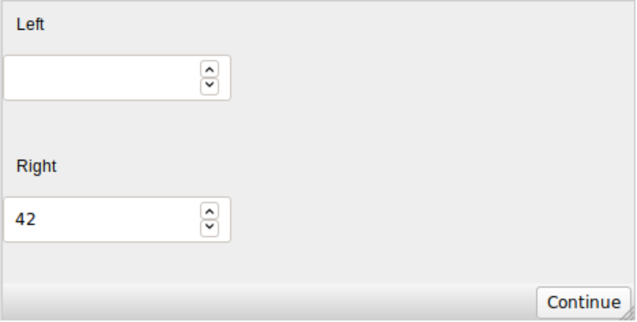
```
t :: Task Int
t = enterInformation "Left" [] -||- enterInformation "Right []
  >>= viewInformation "Result" []
```



Combinators

Parallel Combinators

```
t :: Task Int
t = enterInformation "Left" [] -||- enterInformation "Right []
  >>= viewInformation "Result" []
```



Left

Right

42

Combinators

Parallel Combinators

```
t :: Task Int
t = enterInformation "Left" [] -||- enterInformation "Right []
  >>= viewInformation "Result" []
```

Result

42

Combinators

Parallel Combinators

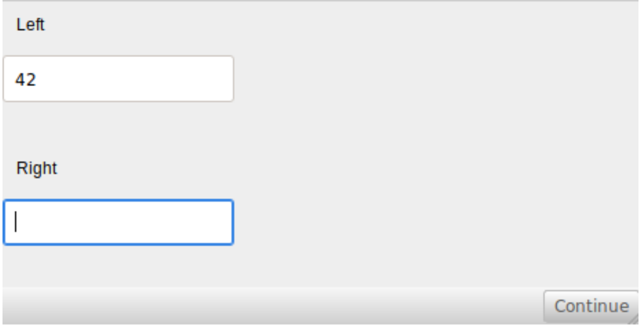
```
t :: Task Int
t = enterInformation "Left" [] -||- enterInformation "Right []
  >>= viewInformation "Result" []
```

```
t :: Task (Int, Int)
t = enterInformation "Left" [] -&&- enterInformation "Right []
  >>= viewInformation "Result" []
```

Combinators

Parallel Combinators

```
t :: Task (Int, Int)
t = enterInformation "Left" [] -&&- enterInformation "Right []
  >>= viewInformation "Result" []
```



Left

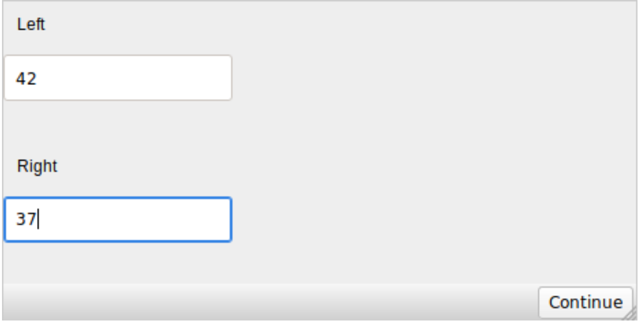
42

Right

Combinators

Parallel Combinators

```
t :: Task (Int, Int)
t = enterInformation "Left" [] -&&- enterInformation "Right []
  >>= viewInformation "Result" []
```



Left

42

Right

37

‘

Combinators

Sequential

```
(>>*) infixl 1 :: (Task a) [TaskCont a (Task b)] → Task b | iTask a & iTask b
:: TaskCont a b
  = OnValue         ((TaskValue a) → Maybe b)
  | OnAction Action ((TaskValue a) → Maybe b)

:: Action = Action String //button
```

Combinators

Sequential

```
always       :: b                  (TaskValue a) → Maybe b
never        :: b                  (TaskValue a) → Maybe b
hasValue     :: (a → b)            (TaskValue a) → Maybe b
ifStable     :: (a → b)            (TaskValue a) → Maybe b
ifUnstable   :: (a → b)            (TaskValue a) → Maybe b
ifValue      :: (a → Bool) (a → b) (TaskValue a) → Maybe b
ifCond       :: Bool b             (TaskValue a) → Maybe b
withoutValue :: (Maybe b)          (TaskValue a) → Maybe b
withValue    :: (a → Maybe b)      (TaskValue a) → Maybe b
withStable   :: (a → Maybe b)      (TaskValue a) → Maybe b
withUnstable :: (a → Maybe b)      (TaskValue a) → Maybe b
```

Combinators

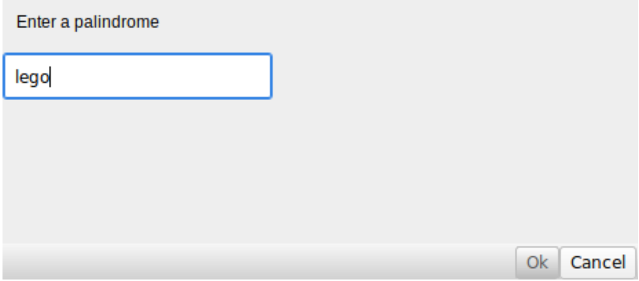
Sequential

```
palindrome :: Task (Maybe String)
palindrome = enterInformation "Enter a palindrome" []
  >>* [ OnAction (Action "Ok")     (ifValue palindrome (λv → return (Just v)))
      , OnAction (Action "Cancel") (always (return Nothing))]
  >>= viewInformation "Result is:" []
where
  palindrome s = s == reverse s
```

Combinators

Sequential

```
palindrome :: Task (Maybe String)
palindrome = enterInformation "Enter a palindrome" []
  >>* [ OnAction (Action "Ok")     (ifValue palindrome (λv → return (Just v)))
      , OnAction (Action "Cancel") (always (return Nothing))]
  >>= viewInformation "Result is:" []
where
  palindrome s = s == reverse s
```



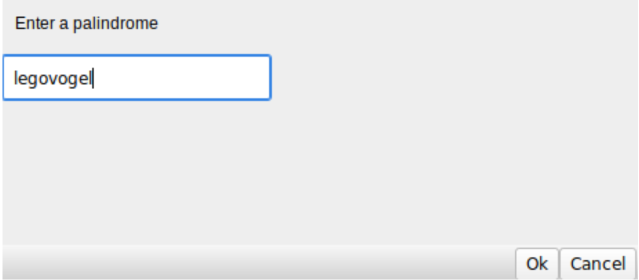
Enter a palindrome

lego

Combinators

Sequential

```
palindrome :: Task (Maybe String)
palindrome = enterInformation "Enter a palindrome" []
  >>* [ OnAction (Action "Ok")     (ifValue palindrome (λv → return (Just v)))
      , OnAction (Action "Cancel") (always (return Nothing))]
  >>= viewInformation "Result is:" []
where
  palindrome s = s == reverse s
```



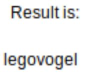
Enter a palindrome

legovogel

Combinators

Sequential

```
palindrome :: Task (Maybe String)
palindrome = enterInformation "Enter a palindrome" []
  >>* [ OnAction (Action "Ok")     (ifValue palindrome (λv → return (Just v)))
      , OnAction (Action "Cancel") (always (return Nothing))]
  >>= viewInformation "Result is:" []
where
  palindrome s = s == reverse s
```



Result is:

legovogel

Combinators

Derived Sequential Combinators

```
(>>=) infixl 1 :: (Task a) (a → Task b) → Task b | iTask a & iTask b
(>>!) infixl 2 :: (Task a) (a → Task b) → Task b | iTask a & iTask b
(>>-) infixl 1 :: (Task a) (a → Task b) → Task b | iTask a & iTask b
(>-|) infixl 1
(>>~) infixl 1 :: (Task a) (a → Task b) → Task b | iTask a & iTask b
(>>^) infixl 1 :: (Task a) (Task b) → Task a| iTask a & iTask b
sequence :: [Task a] → Task [a] | iTask a
```

SDSs

Defining SDSs

```
sharedStore :: String a → SimpleSDSLens a | JSONEncode{|*|} a & JSONDecode{|*|} a & TC a
withShared  :: b ((SimpleSDSLens b) → Task a) → Task a | iTask a & iTask b
```

SDSs

Access Tasks

```
get :: (sds () a w) → Task a | iTask a & Readable sds & TC w
set :: a (sds () r a)  → Task a | iTask a & TC r & Writeable sds
upd :: (r → w) (sds () r w) → Task w | iTask r & iTask w & RWShared sds
watch :: (sds () r w) → Task r | iTask r & TC w & Readable, Registrable sds
```

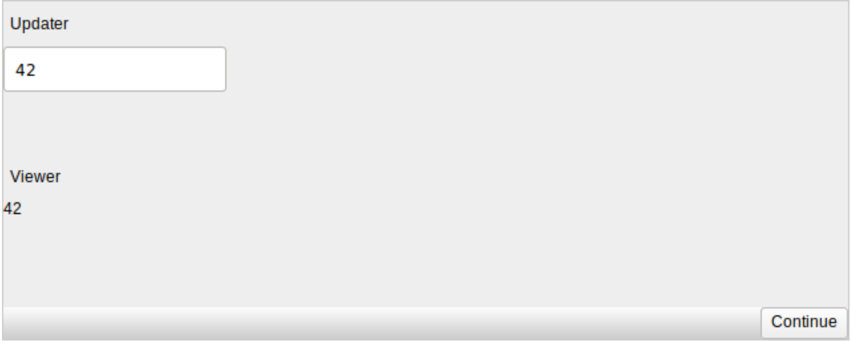
SDSs

Shared Editors

```
updateSharedInformation :: d [UpdateOption r w] (sds () r w) → Task r | ...
viewSharedInformation   :: d [ViewOption r]     (sds () r w) → Task r | ...

sharedUpdate :: Task Int
sharedUpdate = withShared 42 λsharedInt→
       updateSharedInformation () [] sharedInt
  -||- updateSharedInformation () [] sharedInt
```

Example SDS usage

```
shareTask :: Task Int
shareTask = withShared 42 λsi→
     updateSharedInformation "Updater" [] si
  -|| viewSharedInformation   "Viewer" [] si
```



Updater

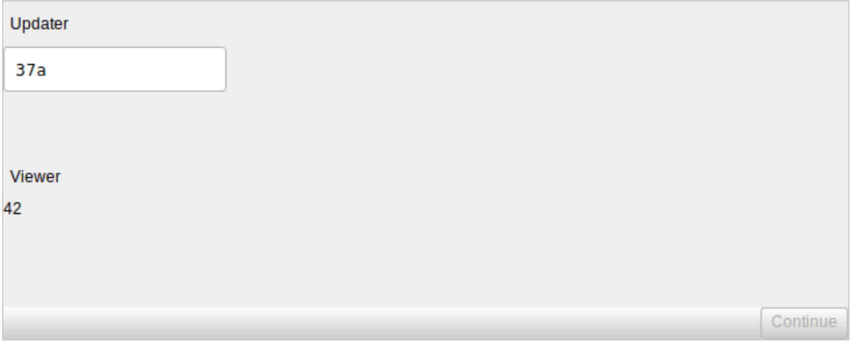
42

Viewer

42

Example SDS usage

```
shareTask :: Task Int
shareTask = withShared 42 λsi→
     updateSharedInformation "Updater" [] si
  -|| viewSharedInformation   "Viewer" [] si
```



Updater

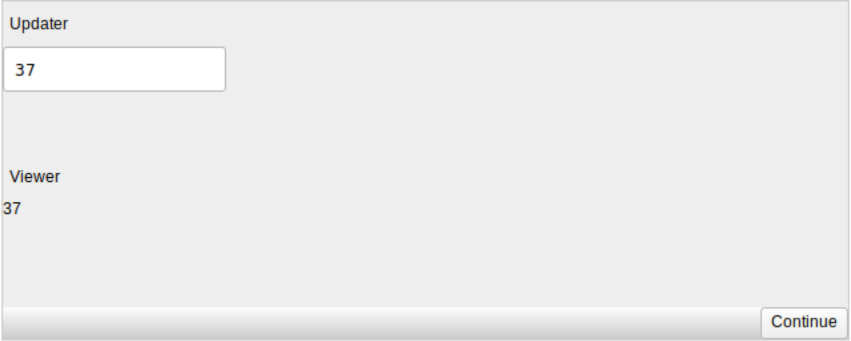
37a

Viewer

42

Example SDS usage

```
shareTask :: Task Int
shareTask = withShared 42 λsi→
     updateSharedInformation "Updater" [] si
  -|| viewSharedInformation   "Viewer" [] si
```



Updater

37

Viewer

37

mTask

mTask design

- Brings TOP to the IOT

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT
- TOP abstractions, IOT needs abstraction

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT
- TOP abstractions, IOT needs abstraction
- Class based shallow EDSL

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT
- TOP abstractions, IOT needs abstraction
- Class based shallow EDSL
- Embedded in Clean

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT
- TOP abstractions, IOT needs abstraction
- Class based shallow EDSL
- Embedded in Clean
- Integration with iTasks

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT
- TOP abstractions, IOT needs abstraction
- Class based shallow EDSL
- Embedded in Clean
- Integration with iTasks
- Multiple backends:

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT
- TOP abstractions, IOT needs abstraction
- Class based shallow EDSL
- Embedded in Clean
- Integration with iTasks
- Multiple backends:
  - pretty printing

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT
- TOP abstractions, IOT needs abstraction
- Class based shallow EDSL
- Embedded in Clean
- Integration with iTasks
- Multiple backends:
  - pretty printing
  - symbolic simulation

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT
- TOP abstractions, IOT needs abstraction
- Class based shallow EDSL
- Embedded in Clean
- Integration with iTasks
- Multiple backends:
  - pretty printing
  - symbolic simulation
  - resource analysis

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT
- TOP abstractions, IOT needs abstraction
- Class based shallow EDSL
- Embedded in Clean
- Integration with iTasks
- Multiple backends:
  - pretty printing
  - symbolic simulation
  - resource analysis
  - C code generation

mTask design

- Brings TOP to the IOT
- Tasks are intuitive for IOT
- TOP abstractions, IOT needs abstraction
- Class based shallow EDSL
- Embedded in Clean
- Integration with iTasks
- Multiple backends:
    - pretty printing
    - symbolic simulation
    - resource analysis
    - C code generation
    - bytecode generation.

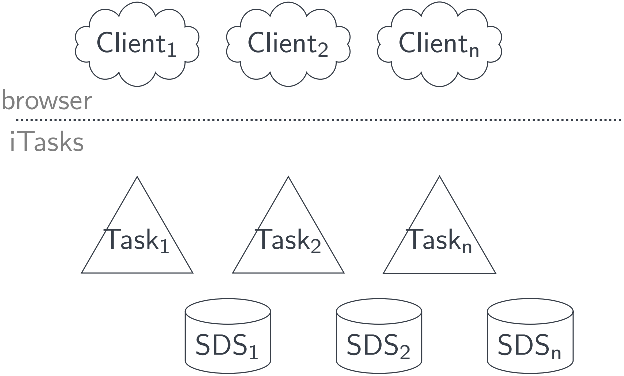
Architecture

Architecture

browser

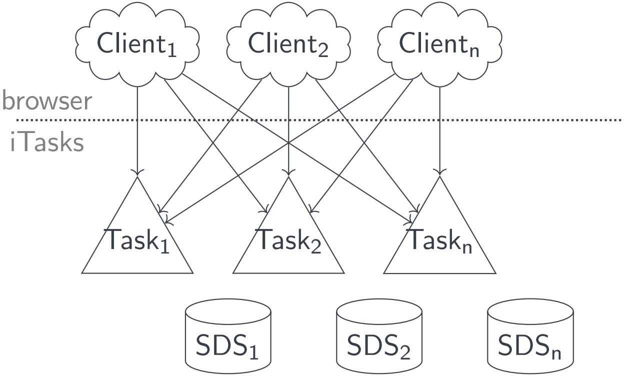
iTasks

Architecture



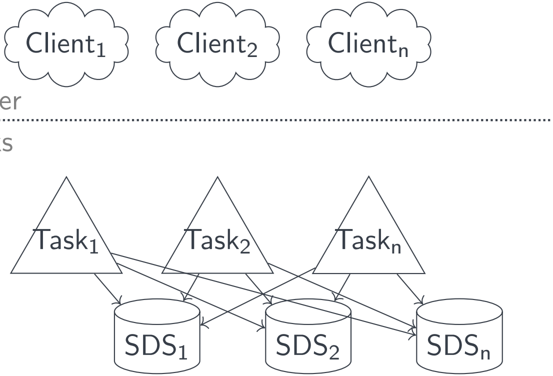
- Javascript
- Clean
- Shared Stores

Architecture



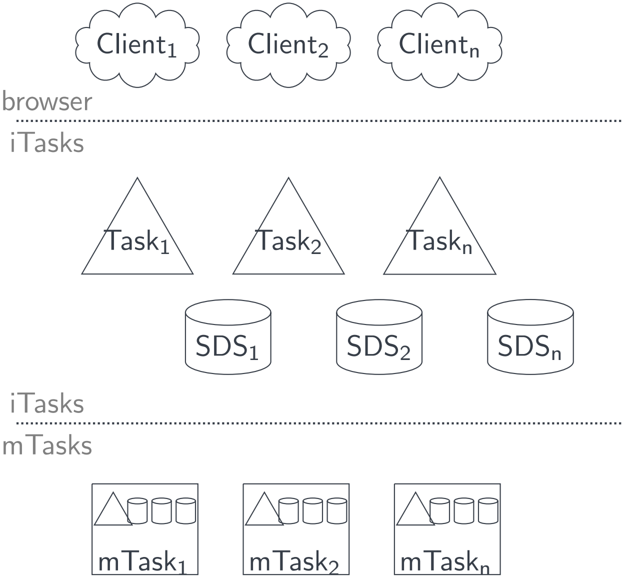
- Type driven UI

Architecture



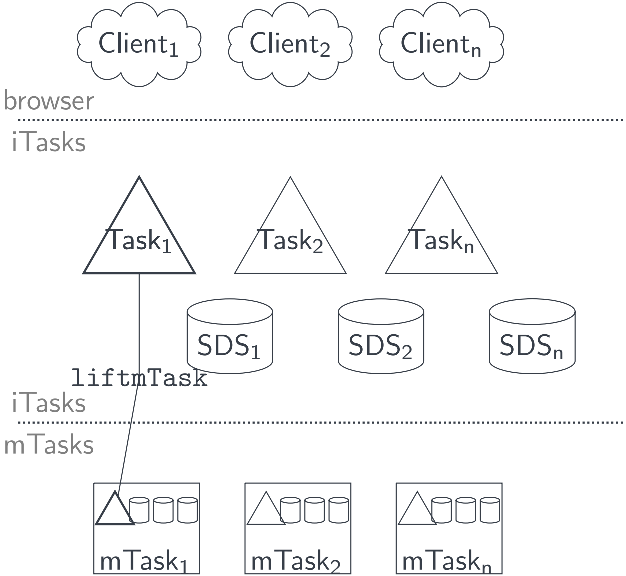
- Synchronization
- Events

Architecture



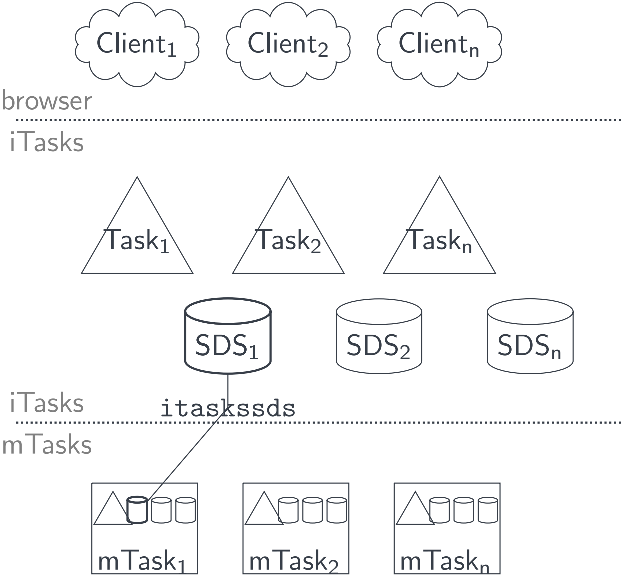
- Devices
- Tasks
- Shared Stores
- RTS/Interpreter

Architecture



- iTasks task as mTask task
- Rewrite task
- Synchronize task value

Architecture



- Synchronize Shared Store
- Publish Subscribe

iTasks interface

Connecting a device: `withDevice`

```
:: MTDevice
withDevice :: (MTDevice → Task b) a → Task b | channelSync a & ...

instance channelSync TCPDevice
instance channelSync TTYDevice
```

iTasks interface

Connecting a device: withDevice

```
:: MTDevice
withDevice :: (MTDevice → Task b) a → Task b | channelSync a & ...

instance channelSync TCPDevice
instance channelSync TTYDevice
```

- Literally a single `parallel`

iTasks interface

Connecting a device: `withDevice`

```
:: MTDevice
withDevice :: (MTDevice → Task b) a → Task b | channelSync a & ...

instance channelSync TCPDevice
instance channelSync TTYDevice
```

- Literally a single `parallel`
- Create Channels (`withShared`)

iTasks interface

Connecting a device: `withDevice`

```
:: MTDevice
withDevice :: (MTDevice → Task b) a → Task b | channelSync a & ...

instance channelSync TCPDevice
instance channelSync TTYDevice
```

- Literally a single `parallel`
- Create Channels (`withShared`)
- Setup the connection by running the channel sync

iTasks interface

Connecting a device: `withDevice`

```
:: MTDevice
withDevice :: (MTDevice → Task b) a → Task b | channelSync a & ...

instance channelSync TCPDevice
instance channelSync TTYDevice
```

- Literally a single `parallel`
- Create Channels (`withShared`)
- Setup the connection by running the channel sync
- Ask for a specification (embedded in the `MTDevice`)

iTasks interface

Connecting a device: withDevice

```
:: MTDevice
withDevice :: (MTDevice → Task b) a → Task b | channelSync a & ...

instance channelSync TCPDevice
instance channelSync TTYDevice
```

- Literally a single `parallel`
- Create Channels (`withShared`)
- Setup the connection by running the channel sync
- Ask for a specification (embedded in the `MTDevice`)
- Monitor the channels

iTasks interface

Connecting a device: `withDevice`

```
:: MTDevice
withDevice :: (MTDevice → Task b) a → Task b | channelSync a & ...

instance channelSync TCPDevice
instance channelSync TTYDevice
```

- Literally a single `parallel`
- Create Channels (`withShared`)
- Setup the connection by running the channel sync
- Ask for a specification (embedded in the `MTDevice`)
- Monitor the channels
- Run the device task

iTasks interface

Connecting a device: withDevice

```
:: MTDevice
withDevice :: (MTDevice → Task b) a → Task b | channelSync a & ...

instance channelSync TCPDevice
instance channelSync TTYDevice
```

- Literally a single `parallel`
- Create Channels (`withShared`)
- Setup the connection by running the channel sync
- Ask for a specification (embedded in the `MTDevice`)
- Monitor the channels
- Run the device task
- Play some trickery to clean up when the argument task is destroyed

iTasks interface

Connecting a device: `withDevice`

```
:: MTDevice
withDevice :: (MTDevice → Task b) a → Task b | channelSync a & ...

instance channelSync TCPDevice
instance channelSync TTYDevice
```

- Literally a single `parallel`
- Create Channels (`withShared`)
- Setup the connection by running the channel sync
- Ask for a specification (embedded in the `MTDevice`)
- Monitor the channels
- Run the device task
- Play some trickery to clean up when the argument task is destroyed
- Close the connection when done

iTasks interface

Lifting SDSs: liftSds

```
class liftsds v where
  liftsds :: ((v (Sds t))→In (Shared t) (Main (MTask v u))) → Main (MTask v u) | ...
```

iTasks interface

Lifting SDSs: liftSds

```
class liftsds v where
  liftsds :: ((v (Sds t))→In (Shared t) (Main (MTask v u))) → Main (MTask v u) | ...


:: MTLens sds :== Shared sds String255
lens :: ((Shared s1 a) → MTLens s2) | type, iTask a & RWShared s1 & RWShared s2
lens = mapReadWriteError
  ( λr→Ok (fromString (toByteCode{|*|} r))
  , λw r→Just <$> iTasksDecode (toString w)
  ) Nothing

iTasksDecode :: String → MaybeError TaskException a | type a
```

iTasks interface

Lifting an mTask to iTasks: `liftmTask`

```
liftmTask :: MTDevice (Main (MTask BCInterpret u)) → Task u | iTask, type u
```

iTasks interface

Lifting an mTask to iTasks: `liftmTask`

```
liftmTask :: MTDevice (Main (MTask BCInterpret u)) → Task u | iTask, type u
```

- Literally a sequence

iTasks interface

Lifting an mTask to iTasks: `liftmTask`

```
liftmTask :: MTDevice (Main (MTask BCInterpret u)) → Task u | iTask, type u
```

- Literally a sequence
- Compile the task

iTasks interface

Lifting an mTask to iTasks: `liftmTask`

```
liftmTask :: MTDevice (Main (MTask BCInterpret u)) → Task u | iTask, type u
```

- Literally a sequence
- Compile the task
- Retrieve all SDS values

iTasks interface

Lifting an mTask to iTasks: `liftmTask`

```
liftmTask :: MTDevice (Main (MTask BCInterpret u)) → Task u | iTask, type u
```

- Literally a sequence
- Compile the task
- Retrieve all SDS values
- Ask the device to prepare (slow comm, small buffers)

iTasks interface

Lifting an mTask to iTasks: `liftmTask`

```
liftmTask :: MTDevice (Main (MTask BCInterpret u)) → Task u | iTask, type u
```

- Literally a sequence
- Compile the task
- Retrieve all SDS values
- Ask the device to prepare (slow comm, small buffers)
- Send the task

iTasks interface

Lifting an mTask to iTasks: `liftmTask`

```
liftmTask :: MTDevice (Main (MTask BCInterpret u)) → Task u | iTask, type u
```

- Literally a sequence
- Compile the task
- Retrieve all SDS values
- Ask the device to prepare (slow comm, small buffers)
- Send the task
- Wait for it to return

iTasks interface

Lifting an mTask to iTasks: `liftmTask`

```
liftmTask :: MTDevice (Main (MTask BCInterpret u)) → Task u | iTask, type u
```

- Literally a sequence
- Compile the task
- Retrieve all SDS values
- Ask the device to prepare (slow comm, small buffers)
- Send the task
- Wait for it to return
- Watch all linked SDSs both ways

iTasks interface

Lifting an mTask to iTasks: `liftmTask`

```
liftmTask :: MTDevice (Main (MTask BCInterpret u)) → Task u | iTask, type u
```

- Literally a sequence
- Compile the task
- Retrieve all SDS values
- Ask the device to prepare (slow comm, small buffers)
- Send the task
- Wait for it to return
- Watch all linked SDSs both ways
- Relay the task value to the task itself

Thermostat

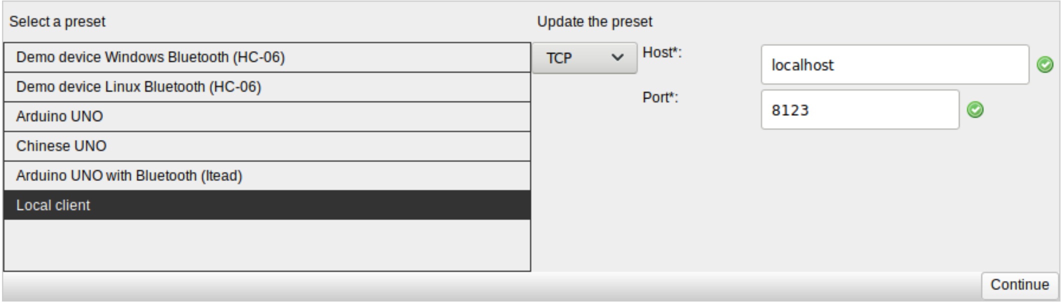
Example: Thermostat

The iTasks part

Example: Thermostat

The iTasks part

```
main :: Task ((), ())
main = enterDevice
  >>= withDevice λdev→
     withShared (160,220) λtargets→
     withShared 420 λtemp→
         updateSharedInformation "Targets" [targetUpdater] targets
      ||- viewSharedInformation "Current" [ViewAs targetView] temp
      ||- liftmTask dev (mTask targets temp)
```



Select a preset

- Demo device Windows Bluetooth (HC-06)
- Demo device Linux Bluetooth (HC-06)
- Arduino UNO
- Chinese UNO
- Arduino UNO with Bluetooth (Itead)
- Local client

Update the preset

TCP

Host*: localhost

Port*: 8123

Example: Thermostat

The iTasks part

```
main :: Task ((), ())
main = enterDevice
  >>= withDevice λdev→
     withShared (160,220) λtargets→
     withShared 420 λtemp→
         updateSharedInformation "Targets" [targetUpdater] targets
      ||- viewSharedInformation "Current" [ViewAs targetView] temp
      ||- liftmTask dev (mTask targets temp)
```

- Connect to the device
- Start the synchronization task
- Ask for a specification
- Wait for the specification to return

Example: Thermostat

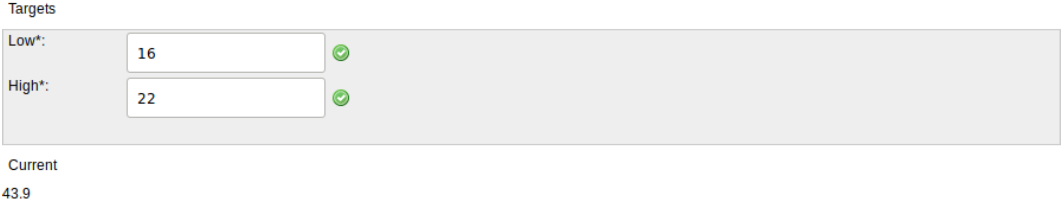
The iTasks part

```
main :: Task ((), ())
main = enterDevice
  >>= withDevice λdev→
     withShared (160,220) λtargets→
     withShared 420 λtemp→
         updateSharedInformation "Targets" [targetUpdater] targets
      ||- viewSharedInformation "Current" [ViewAs targetView] temp
      ||- liftmTask dev (mTask targets temp)
```

Example: Thermostat

The iTasks part

```
main :: Task ((), ())
main = enterDevice
  >>= withDevice λdev→
     withShared (160,220) λtargets→
     withShared 420 λtemp→
        updateSharedInformation "Targets" [targetUpdater] targets
      ||- viewSharedInformation "Current" [ViewAs targetView] temp
      ||- liftmTask dev (mTask targets temp)
```



Targets

Low*: 16 ✓

High*: 22 ✓

Current

43.9

Example: Thermostat

The iTasks part

```
main :: Task ((), ())
main = enterDevice
  >>= withDevice λdev→
     withShared (160,220) λtargets→
     withShared 420 λtemp→
         updateSharedInformation "Targets" [targetUpdater] targets
      ||- viewSharedInformation "Current" [ViewAs targetView] temp
      ||- liftmTask dev (mTask targets temp)
```

- Compile the task
- Send the task
- Wait for acknowledgement
- Synchronize lifted SDSs

Example: Thermostat

The IOT part

Example: Thermostat

The IOT part

```
mTask targetShare tempShare =
  DHT DHTPIN DHT22 λdht→
  liftsds λtarget = targetShare In
  liftsds λtemp   = tempShare   In
  {main
    =    ever (temperature dht >>~. setSds temp >>|. delay (lit 2000))
    .&&. ever (getSds temp .&&. getSds target >>~. tupopen (temp, target)→λv→
             writeD FANPIN  (temp <. second target)
        .&&. writeD HEATPIN (temp >. first  target)
    )}
```

Example: Thermostat

The IOT part

```
mTask targetShare tempShare =
  DHT DHTPIN DHT22 λdht→
  liftsds λtarget = targetShare In
  liftsds λtemp   = tempShare   In
  {main
    =    ever (temperature dht >>~. setSds temp >>|. delay (lit 2000))
    .&&. ever (getSds temp .&&. getSds target >>~. tupopen (temp, target)→λv→
             writeD FANPIN  (temp <. second target)
        .&&. writeD HEATPIN (temp >. first  target)
    )}
```

Example: Thermostat

The IOT part

```
mTask targetShare tempShare =
  DHT DHTPIN DHT22 λdht→
  liftsds λtarget = targetShare In
  liftsds λtemp   = tempShare   In
  {main
    =    ever (temperature dht >>~. setSds temp >>|. delay (lit 2000))
    .&&. ever (getSds temp .&&. getSds target >>~. tupopen (temp, target)→λv→
             writeD FANPIN  (temp <. second target)
        .&&. writeD HEATPIN (temp >. first  target)
    )}
```

Example: Thermostat

The IOT part

```
mTask targetShare tempShare =
  DHT DHTPIN DHT22 λdht→
  liftsds λtarget = targetShare In
  liftsds λtemp   = tempShare   In
  {main
    =    ever (temperature dht >>~. setSds temp >>|. delay (lit 2000))
    .&&. ever (getSds temp .&&. getSds target >>~. tupopen (temp, target)→λv→
             writeD FANPIN  (temp <. second target)
        .&&. writeD HEATPIN (temp >. first  target)
    )}
```

Example: Thermostat

The IOT part

```
mTask targetShare tempShare =
  DHT DHTPIN DHT22 λdht→
  liftsds λtarget = targetShare In
  liftsds λtemp   = tempShare   In
  {main
    =    ever (temperature dht >>~. setSds temp >>|. delay (lit 2000))
    .&&. ever (getSds temp .&&. getSds target >>~. tupopen (temp, target)→λv→
             writeD FANPIN  (temp <. second target)
        .&&. writeD HEATPIN (temp >. first  target)
    )}
```

Example: Thermostat

The IOT part

Example: Thermostat

The IOT part

```
mTask targetShare tempShare =
  DHT DHTPIN DHT22 λdht→
  liftsds λtarget = targetShare In
  liftsds λtemp   = tempShare   In
  {main
    =    ever (temperature dht >>~. setSds temp >>|. delay (lit 2000))
    .&&. ever (getSds temp .&&. getSds target >>~. tupopen (temp, target)→λv→
             writeD FANPIN  (temp <. second target)
        .&&. writeD HEATPIN (temp >. first  target)
    )}
```

Example: Thermostat

The IOT part

```
mTask targetShare tempShare =
  DHT DHTPIN DHT22 λdht→
  liftsds λtarget = targetShare In
  liftsds λtemp   = tempShare   In
  {main
    =    ever (temperature dht >>~. setSds temp >>|. delay (lit 2000))
    .&&. ever (getSds temp .&&. getSds target >>~. tupopen (temp, target)→λv→
              writeD FANPIN  (temp <. second target)
         .&&. writeD HEATPIN (temp >. first  target)
    )}
```

Example: Thermostat

The IOT part

```
mTask targetShare tempShare =
  DHT DHTPIN DHT22 λdht→
  liftsds λtarget = targetShare In
  liftsds λtemp   = tempShare   In
  {main
    =    ever (temperature dht >>~. setSds temp >>|. delay (lit 2000))
    .&&. ever (getSds temp .&&. getSds target >>~. tupopen (temp, target)→λv→
             writeD FANPIN  (temp <. second target)
        .&&. writeD HEATPIN (temp >. first  target)
    )}
```

Example: Thermostat

The IOT part

```
mTask targetShare tempShare =
  DHT DHTPIN DHT22 λdht→
  liftsds λtarget = targetShare In
  liftsds λtemp   = tempShare   In
  {main
    =    ever (temperature dht >>~. setSds temp >>|. delay (lit 2000))
    .&&. ever (getSds temp .&&. getSds target >>~. tupopen (temp, target)→λv→
             writeD FANPIN  (temp <. second target)
        .&&. writeD HEATPIN (temp >. first  target)
    )}
```

Example: Thermostat

The IOT part

```
mTask targetShare tempShare =
  DHT DHTPIN DHT22 λdht→
  liftsds λtarget = targetShare In
  liftsds λtemp   = tempShare   In
  {main
    =    ever (temperature dht >>~. setSds temp >>|. delay (lit 2000))
    .&&. ever (getSds temp .&&. getSds target >>~. tupopen (temp, target)→λv→
                writeD FANPIN  (temp <. second target)
           .&&. writeD HEATPIN (temp >. first  target)
    )}
```

Seminar

- Questions?

Seminar

- Questions?
- Write your own mTask applications

Seminar

- Questions?
- Write your own mTask applications
- Use Cloogle

Seminar

- Questions?
- Write your own mTask applications
- Use Cloogle
- Download the material `https://cloo.gl/ODY4`

Future work

- Exceptions/interrupts

Future work

- Exceptions/interrupts
- Event based rewriting

Future work

- Exceptions/interrupts
- Event based rewriting
- Unified peripheral interface

Future work

- Exceptions/interrupts
- Event based rewriting
- Unified peripheral interface
- Remote monad

Future work

- Exceptions/interrupts
- Event based rewriting
- Unified peripheral interface
- Remote monad
- . . .

Future work

- Exceptions/interrupts
- Event based rewriting
- Unified peripheral interface
- Remote monad
- . . .
- Collaborate?