Task Oriented Programming

with

-

A Domain Specific Language embedded in

Rinus Plasmeijer – Bas Lijnse
Peter Achten – Pieter Koopman - Steffen Michels
Jurriën Stutterheim - Markus Klinik - Tim Steenvoorden - Mart Lubbers – Arjan Oortgiesen
Jan Martin Jansen (NLDA) – John van Groningen - Laszlo Domoszlai (ELTE)
Radboud University Nijmegen

Overview

- Introduction to Task Oriented Programming / iTasks
- iTask Overview
  - Task Values
  - Basic Tasks: Editors
  - Task Combinators
    - Sequential Combinators
    - Parallel Combinators
  - Shared Data Sources
  - Current Research

Modern Ways of Working

Computer

Laptop

Tablet

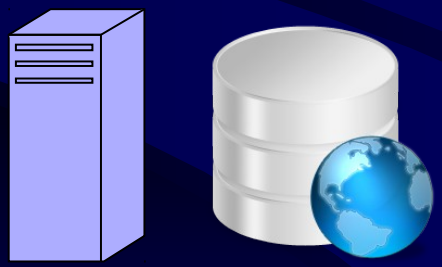
Smart Phone

Smart Everything / IoT

Internet

Server

Cloud / Database

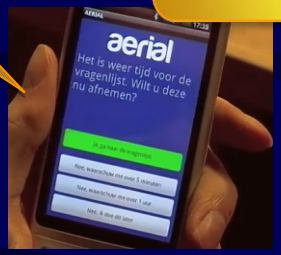


Modern Ways of Working

Software for Browser

Apps on Client

"Cool" User Interfaces



Computer

Laptop

Tablet

Smart Phone

Smart Everything / Ic

Small Apps on Client

"Cool" User Interface

Internet

Coordinate Collaboration All Devices

Handle Communication

Server

Cloud / Database

Synchronize Clients - Servers

Handle Data Storage Data Retrieval

Notify everyone when data changes

Software on Server

Software Pollution

- Functionality –Alg
- User Interface– C nanded
- Storage - Persiste
- Many Componen needed
- Multi Platform – D ages / …
- Efficiency / Maintainability / Scalability / Security / Integration with other systems /…

All these things have to be controlled by software !

Too many things to handle, software gets very messy !
Hard to find out what an application actually is doing …

How to structure the software, achieve separation of concerns ?

Software should focus on the tasks to do (what), not on the technical realization (how)

Task Oriented Programming ?

- New *flavor* of Functional Programming, even a *new* Programming Paradigm…

  for developing applications *coordinating* people and systems collaborating on the internet.

- *Tasks* as central concept
  - Tasks are a common notion in daily life / in any organization
  - Computers often play a central role when performing a Task
  - Many programming concepts can be seen as a Tasks to be executed by a computer

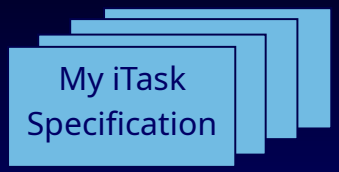
- *Tasks* are described on a high level of abstraction
  - Reactive: a task produce an observable value cha over time
  - Combinators to define tasks in terms of sub-task
  - Data flow: tasks are automatically updated when ved values are changing

  - *Don't worry about the technology realization* !

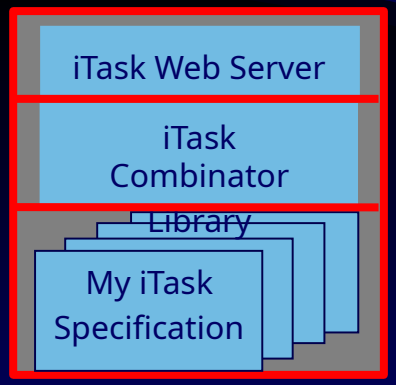
What is the iTasks System ?

- Implementation of the Task Oriented Programming paradig
- It is an Embedded Domain Specific Language, embedded in
- Uses *advanced* FPL techniques to generate the required software technically needed
- First concept 2007 (ICFP), Operation Semantics TOP 2012 (PPDP)
- about 30 papers on TOP / iTasks
- Version 4.x of the iTask System

iTasks Architecture



iTasks Architecture



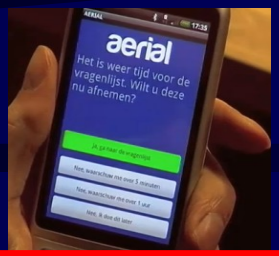
iTasks Architecture

Computer

iTask Client

Laptop

iTask Client



Phone

iTask Client

Tablet

iTask Client

Sensor

iTask Client

iTask User

iTask User

iTask User

iTask User

iTask User

WEB

Web Service

Web Service

Web Service

RPC

Cloud / Database

iTask Web Server

iTask Combinator

Library

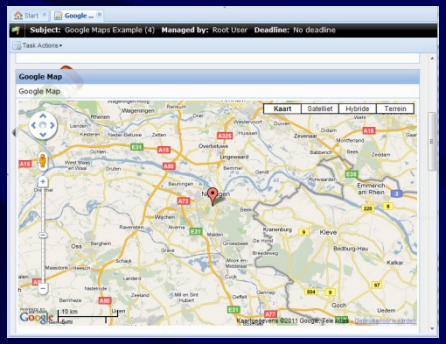
My iTask Specification

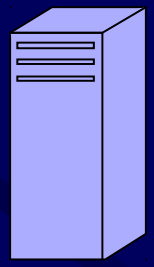
iTask Web Server

iTask Combinator

Library

iTask Specification





Technical issues the iTask system will do for you …

- Generates Web-Service for Server which coordinates the tasks, as described
- One language approach (Clean), generates code for Server(s) and for Clients
  - Multi-platform: Windows & MacOS (Intel), Android & Raspberry Pi (ARM), Browser (Java script)
  - Any Closure (function to be evaluated) can be send from one platform to another
  - We can distribute Servers over Clients
- Generates GUI's on clients, handles interaction, updates clients
- Offers an up-to-date dedicated Model-View to end-users of information shared between tasks
  - Distributed spreadsheet-like behavior (publish-subscribe system)
- Persistent: remembers the work taken place & done so far
  - Any system may be switched of any time
- Generate as much as possible automatically, but one can fine-tune when needed

Papers on iTasks

First paper on iTasks:

- iTasks: Executable Specifications of Interactive Work Flow Systems for the Web (ICFP 2007)

General:

- An Introduction to Task Oriented Programming (CEFP 2013)
- Static and Dynamic Visualisations of Monadic Programs (CEFP 2015)

GUI:

- iTask as a new paradigm for building GUI applications (IFL 2010)
- Task Oriented Programming with Purely Compositional Interactive Scalable Vector Graphics (IFL 2014)
- Towards Customizable Generic Task User-Interfaces in iTasks (IFL 2016)
- Towards the Layout of Things (IFL 2016)

Tooling:

- GiN: a graphical language and tool for defining iTask workflows (TFP 2011)
- Tonic: An Infrastructure to Graphically Represent the Definition and Behaviour of Tasks (TFP 2013)

Semantics:

- Task Oriented Programming in a Pure Functional Language (PPDP 2012)
- iTasks for a change - Type-safe run-time change in dynamically evolving workflows (PEPM 2011)
- Evolution of a Parallel Task Combinator (Springer 2013)
- Getting a grip on tasks that coordinate tasks (LDTA 2011)
- Parametric Lenses: change notification for bidirectional lenses (IFL 2014)
- Using DSLs to help people solve rule-based problems (TFP 2016)
- An Executable and Testable Semantics for iTasks (IFL 2008)
- Predicting Resource Consumption of Higher-Order Workflows (PEPM 2016)

Client site evaluation of Tasks:

- Transparant Ajax and Client-Site Evaluation of iTasks (IFL 2007)
- iEditors: Extending iTask with Interactive Plug-ins (IFL 2008)
- Editlets: type based client side editors for iTasks (IFL 2014)
- A Portable VM-based implementation Platform for non-strict Functional Programming Languages (IFL 2016)
- Task Oriented Programming on a Distributed Server Architecture (draft)

Applicability:

- A Conference Management System based on the iData Toolkit (IFL 2007)
- Web Based Dynamic Workflow Systems for C2 of Military Operations (ICCRTS 2010)
- Managing COPD exacerberations with telemedicine (AIME 2010)
- Towards Dynamic Workflows for Crisis Management (ISCRAM 2010)
- Capturing the Netherlands Coast Guard's SAR Workflow with iTasks (ISCRAM 2011)
- A Task-Oriented Incident Coordination Tool (ISCRAM 2012)
- Using Process-Oriented Interfaces for Solving the Automation Paradox in Highly Automated Navy Vessels (AMT 2014)
- Maintaining Separation of Concerns in Distributed Multi-User Web Applications (draft)

- (Possible) Application Domains
  - Command and Control systems (NLDA, Navy, TNO, Coast Guard)
  - Health care (COPD measurements Smart Phones)
  - Internet of Things (Koopman, New Device Lab, RUN)
  - Any domain where people and systems closely work together on the net…

"Real" Prototype Applications using iTasks

Simple workflow:

- Aerial project: Home Healthcare project (Peter Lucas, Bas Lijnse, e.a.)
  - Testing chronically lung diseases caused by smoking
  - Testing pregnancy disease

Home Healthcare project

"Real" Prototype Applications using iTasks

Real real-life workflow:

- Crisis Management:

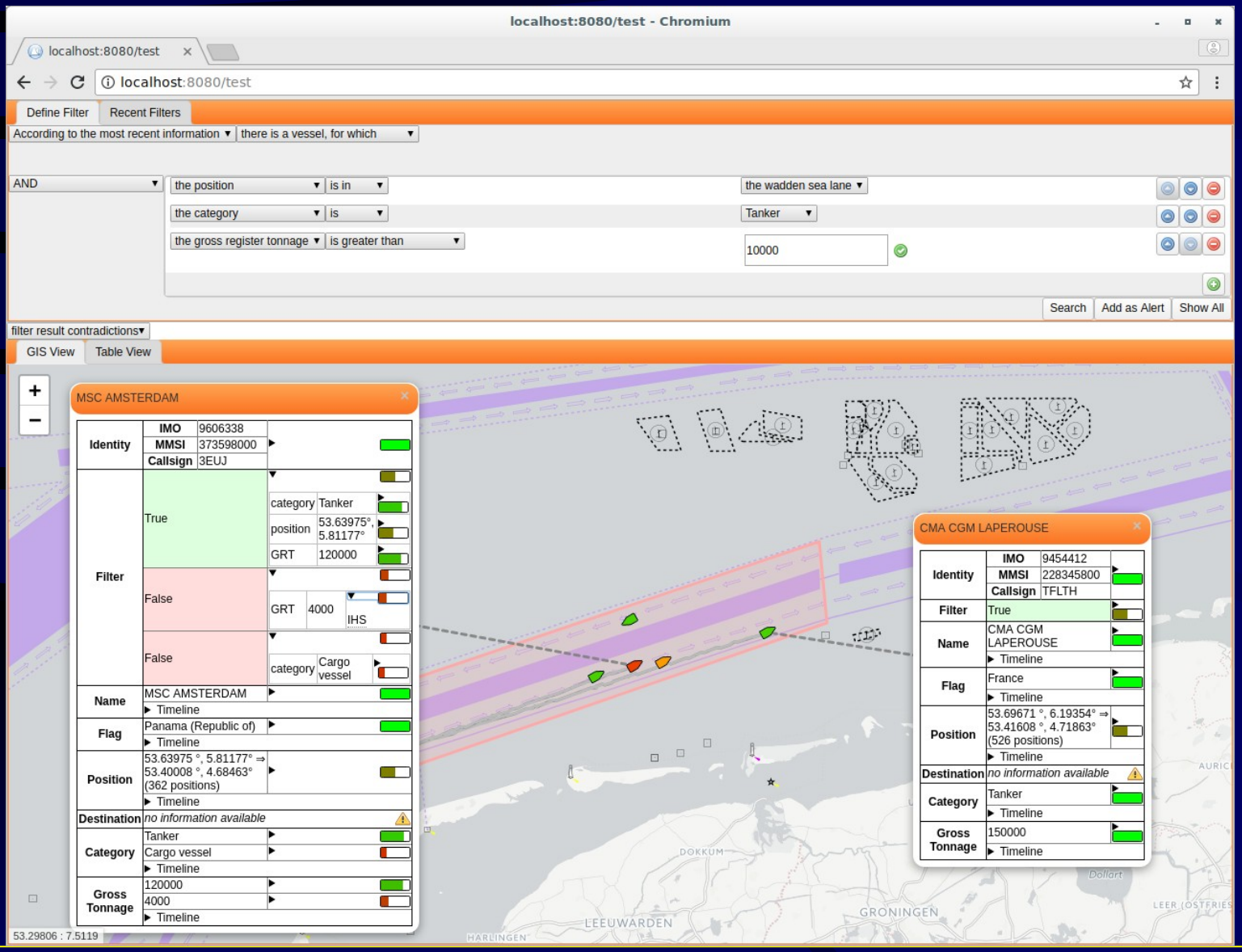
  Capturing the Netherlands Coast Guard's Search And Rescue Workflow

  (ISCRAM 2011, Bas Lijnse, Jan Martin Jansen, Ruud Nanne, Rinus Plasmeijer)

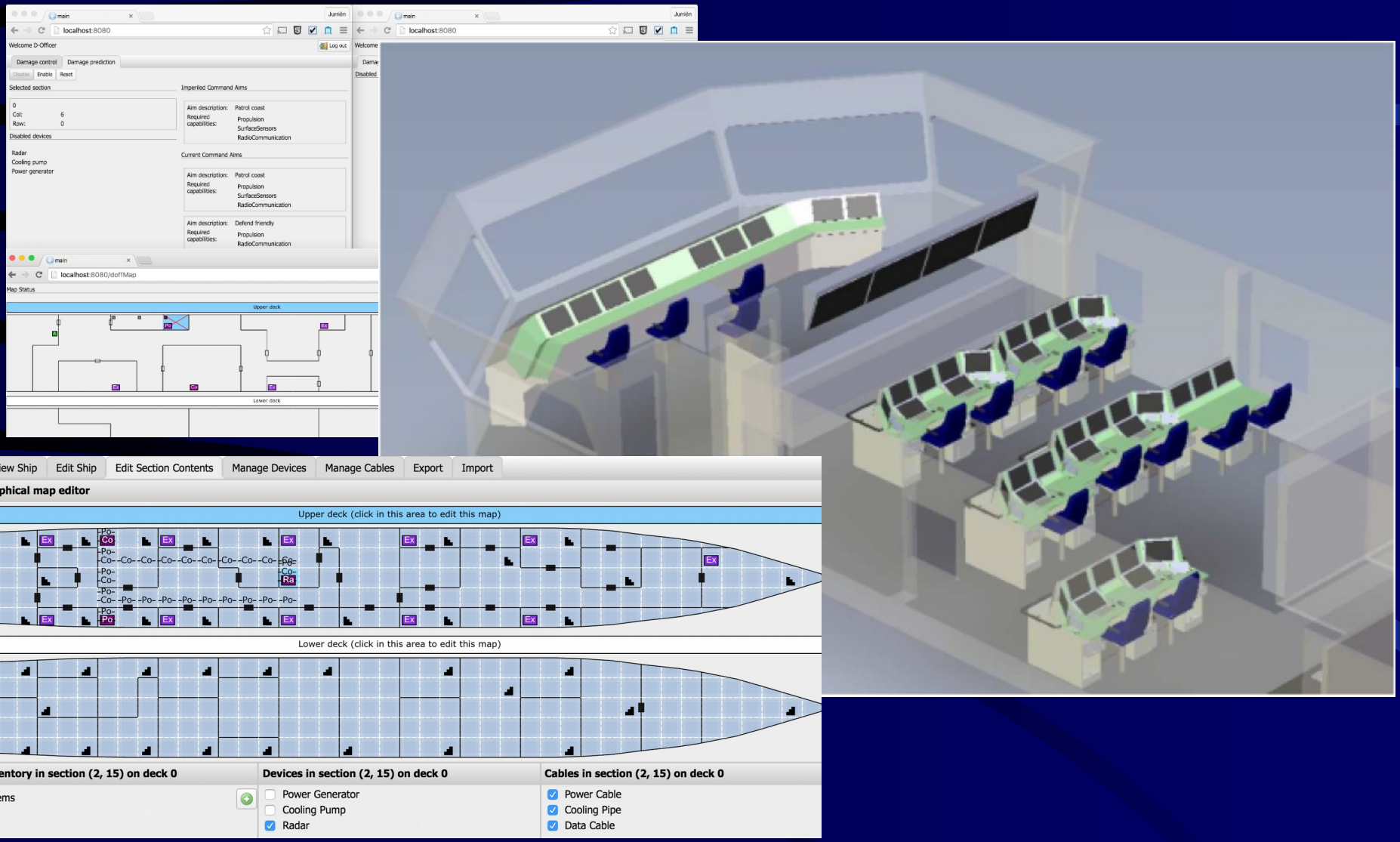
Prototype : Coast Guard Search And Rescue

Prototype : Incidone (Bas Lijnse)

Coast Guarding (Steffen Michels)



Prototype : Fregat Optimization – Navy -TNO



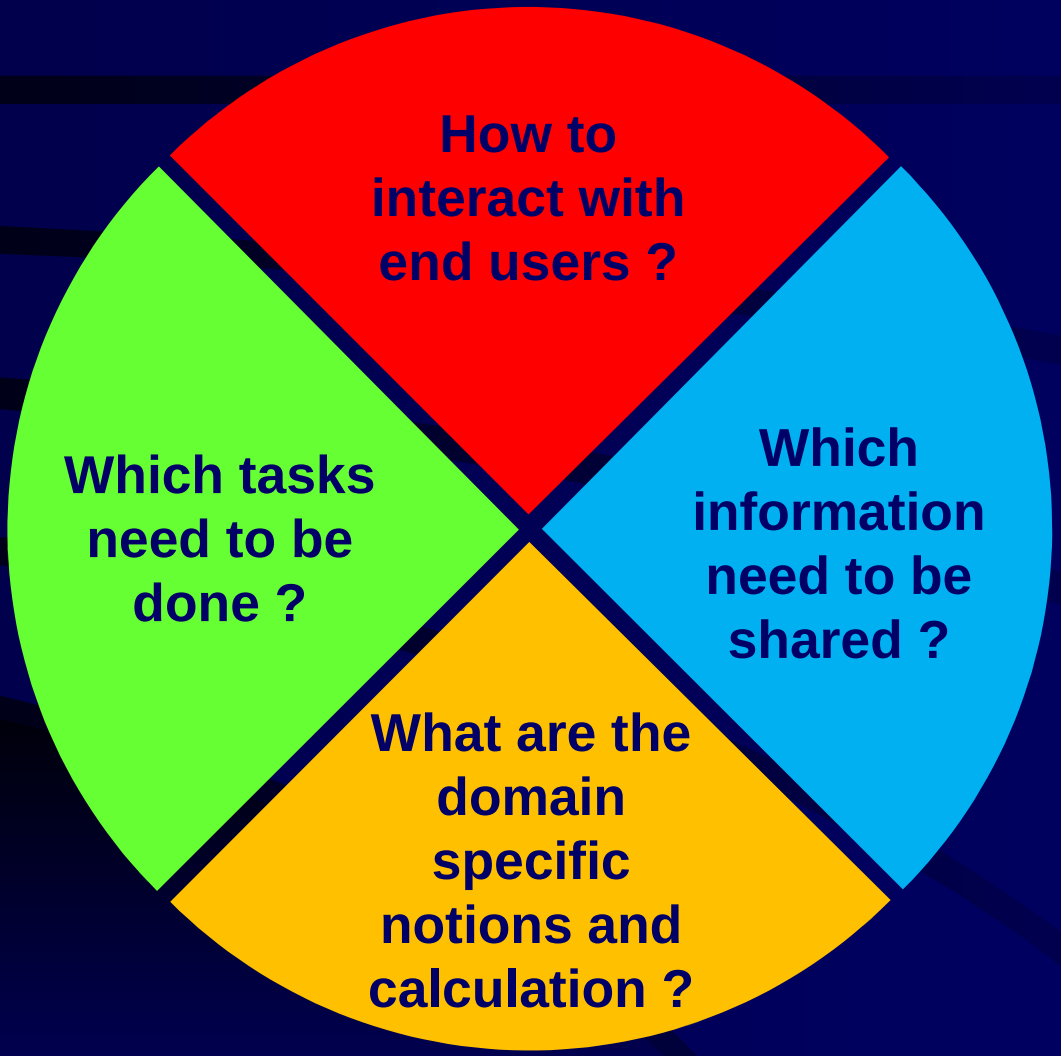
Task Oriented Programming

- (Possible) Application Domains
  - Command and Control systems (NLDA, Navy, TNO, Coast Guard)
  - Health care (COPD measurements Smart Phones)
  - Internet of Things (Koopman, New Device Lab, RUN)
  - Any domain where people and systems closely work together on the net…
- (Possible) Type of Applications :
  - for developing Web-based Multi-User Applications
  - for Rapid Prototyping (NLDA, TNO, Coast Guard, RUN)
  - to Formalize how work should /could be organized (NLDA, Coast Guard)
  - for Simulation (agents / decision support systems): what is the best way of working (Groenouwe, Meyer, ATIA)
  - for Training: mix of real people and agents (Coastguard, RUN)
  - to Show properties by testing, analysis or by formal proof (Klinik RUN)
  - as common language for communication between domain experts, programmers, end-users (Naus, Jeuring, Groenhouwe, Meyer, Stutterheim)
    - Compile-time: *how to organize work to obtain the desired result* ?
    - Run-time: *what is actually happening, and what is the best way to proceed* ?

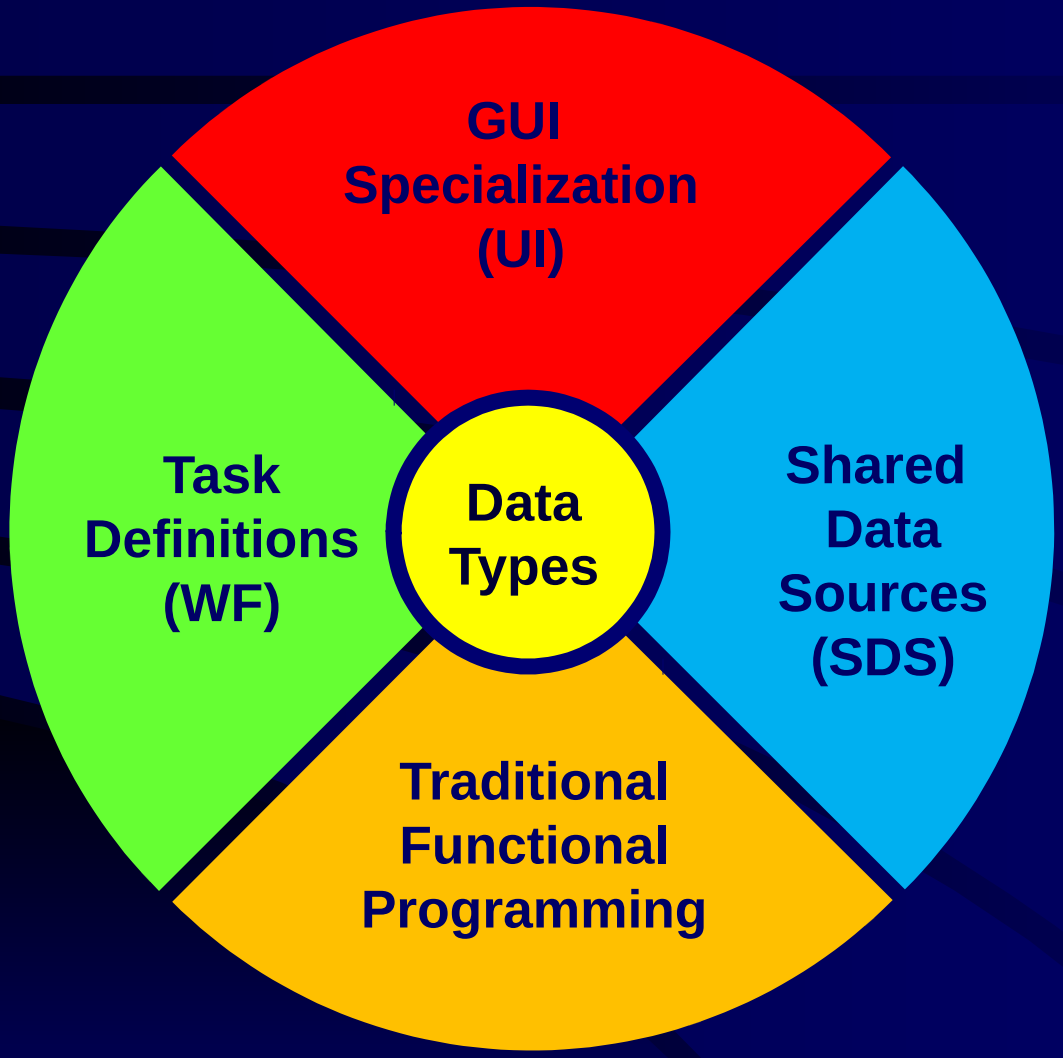
TOP

Task **O**riented **P**rogramming

Domain Specific Questions



Separations of Concerns in TOP



itasks
dynamic workflow system

What does TOP adds on top of FP ?

FP	TOP
Function	Task
One Result	One Result which varies over time

TAS

G

GUI Combinators

Standard not supported Shares : uniform resource interface

Publish-Subscribe System

SDS

pure FPL

Automatic Synchronize others on change

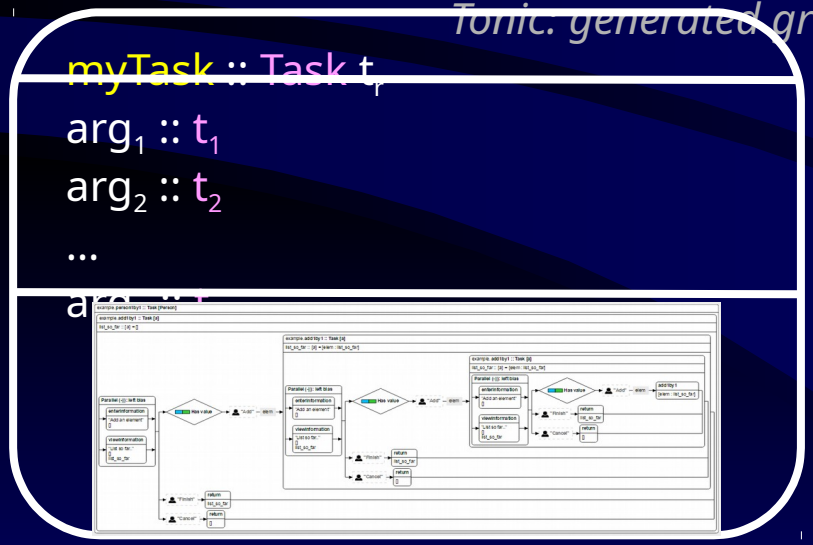
SDS Combinators

TOP – Tasks

A task is a unit of work which delivers results of a certain type.
They can be defined by “common” Clean functions delivering a result of type :: Task a

myTask :: t_1 t_2 .... t_n → Task t_r
myTask arg_1 arg_2 ... arg_n = some-combination-of-sub-tasks

Tonic: generated graphical representation



Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

```
:: TaskValue a          =    NoValue
                        |    Value a Stability
:: Stability            :==  Bool
```

- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

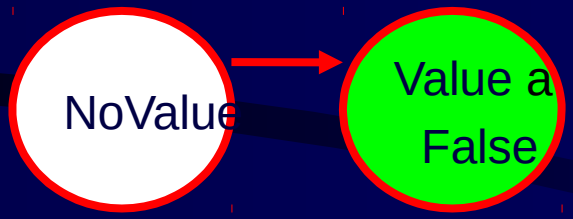
```
:: TaskValue a        =    NoValue
                    |    Value a Stability
:: Stability          :==  Bool
```

- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

```
:: TaskValue a        =    NoValue
                    |    Value a Stability
:: Stability          :==  Bool
```

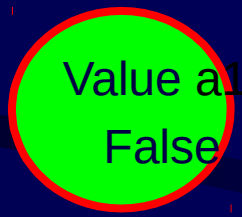


- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

```
:: TaskValue a        =    NoValue
                    |    Value a Stability
:: Stability          :==  Bool
```

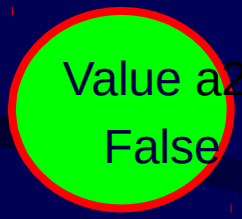


- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

```
:: TaskValue a          =    NoValue
                        |    Value a Stability
:: Stability            :==  Bool
```

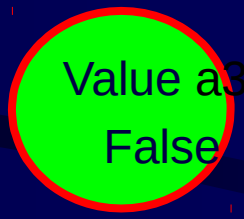


- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

```
:: TaskValue a          =    NoValue
                        |    Value a Stability
:: Stability            :==  Bool
```

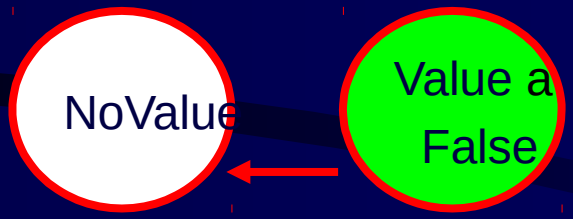


- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

```
:: TaskValue a        =    NoValue
                    |    Value a Stability
:: Stability          :==  Bool
```



- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

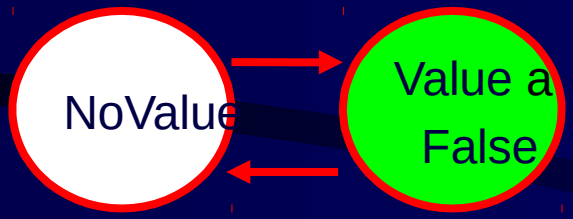
```
:: TaskValue a        =    NoValue
                    |    Value a Stability
:: Stability          :==  Bool
```

- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

```
:: TaskValue a        =    NoValue
                    |    Value a Stability
:: Stability          :==  Bool
```

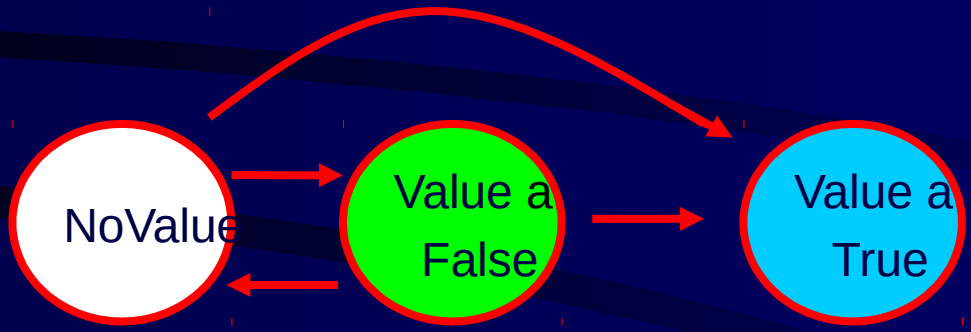


- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

```
:: TaskValue a        =    NoValue
                    |    Value a Stability
:: Stability          :==  Bool
```

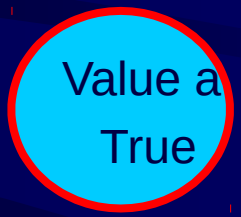


- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

```
:: TaskValue a          =    NoValue
                        |    Value a Stability
:: Stability            :==  Bool
```

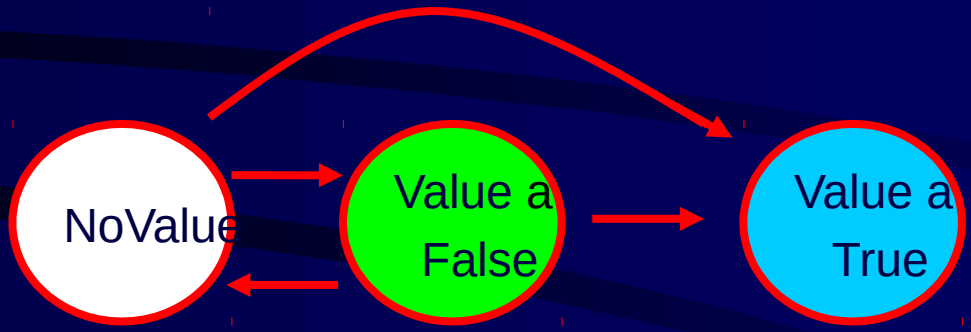


- While the task is going on, its value may change over time

Core - Task Values

A task of type :: Task a produces :: TaskValue a as result:

```
:: TaskValue a        =    NoValue
                    |    Value a Stability
:: Stability          :==  Bool
```



- While the task is going on, its value may change over time
  - Values can be of *any* type: user defined, higher order (e.g another task or a function)
  - The current value can be observed by other tasks
  - Tasks never terminate, but are garbage collected if they are not needed anymore
- A task actually returns a value of type TaskResult

```
:: TaskResult a  =       ValRes TimeStamp (TaskValue a)
                 |  ∃ e: ExcRes e                    & iTask e
```

- It either returns a time stamped (version number) TaskValue,
  or it may raise an exception of *any* type (for which the class iTasks exists)

Basic Tasks and Tasks Combinators

- Tasks
  - Basic Tasks
    - Non-interactive
      - return, throw, ...
    - Interactive editors
      - interact , and derived combinators like enterInformation , showInformation , ...
  - Combinators
    - Sequential
      - step, and derived combinators like >>* , >>= , >>|,...
    - Parallel
      - parallel, and derived combinators like -&&-, -||- , ...

Non-Interactive Basic Tasks

- Lift an ordinary expression to the task domain, delivers a Stable value :

```
return :: a → Task a   | iTask a
```

- Raise an exceptional result (ExcRes)

```
throw :: a → Task e    | iTask e
```

Set of generic functions
Works for any first order type

- Conversion of Task Values

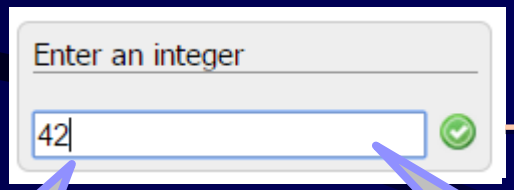
```
(@?) infixl 1 :: (Task a)  ((TaskValue a) → (TaskValue b))  → Task b       | iTask a & iTask b

(@)  infixl 1 :: (Task a)   (a → b)                         → Task b       | iTask a & iTask b
```

Interactive Basic Tasks: Editors

```
module example

import iTasks

Start :: *World → *World
Start world = startEngine myTask world

myTask :: Task Int
myTask = enterInformation "Enter an integer" []
```

Editors

```
module example

import iTasks

Start :: *World → *World
Start world = startEngine myTask world

myTask :: Task Int
myTask = enterInformation "Enter an integer" []
```



Prompt

option to map a task value to a view value of another type

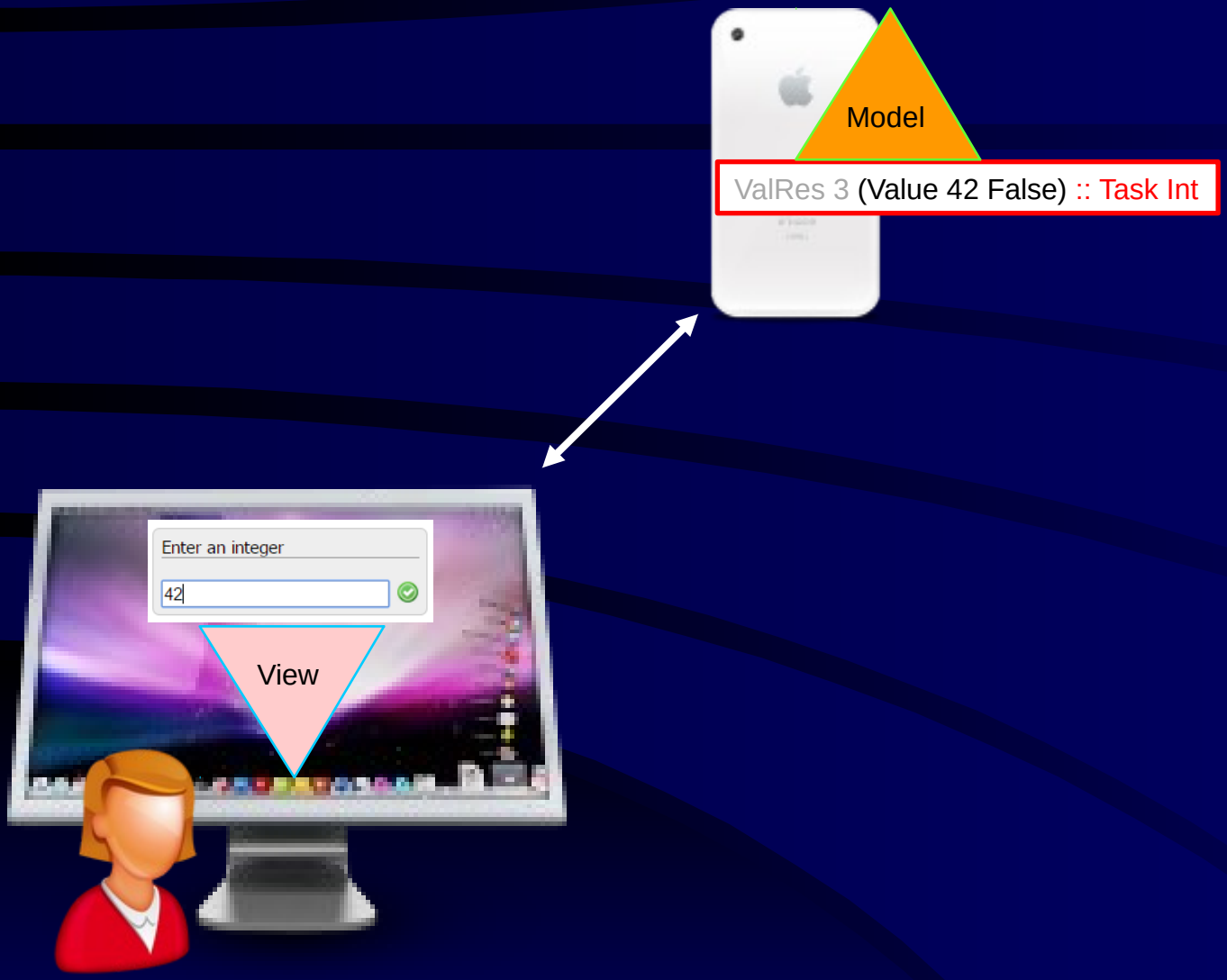
Model

View

With the editor one can constantly change the current value

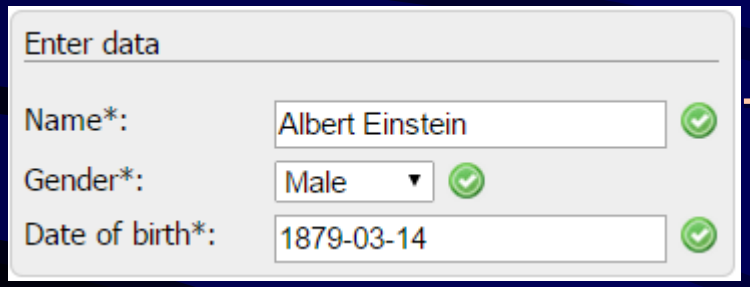
Editors therefore never deliver a Stable value

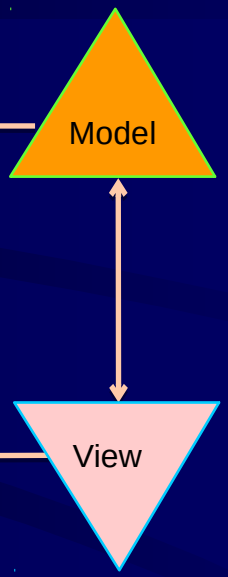
Editors – Task Value on Server kept synchronized with the View on Client



Generic Editors

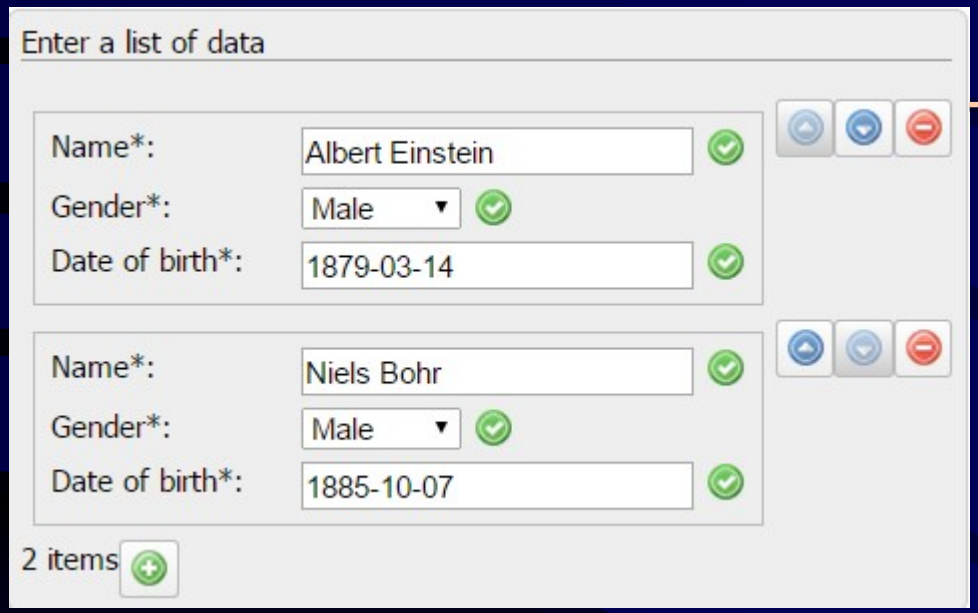
```
:: Person     = { name :: String, gender :: Gender, dateOfBirth :: Maybe Date}
:: Gender     = Male | Female

derive class iTask Person, Gender

myTask :: Task Person
myTask = enterInformation "Enter data" []
```

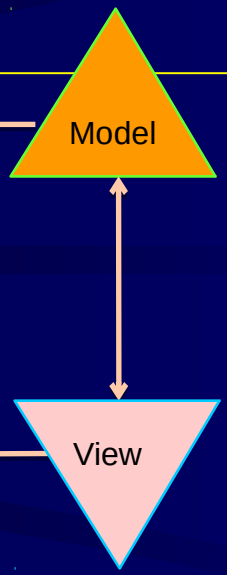




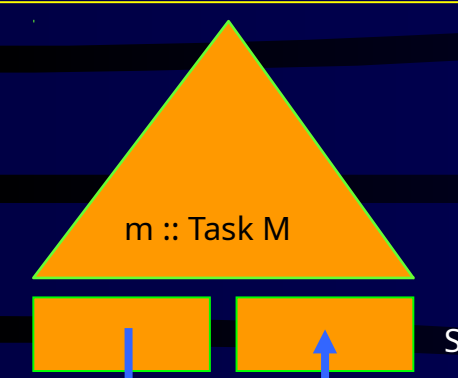
Generic Editors

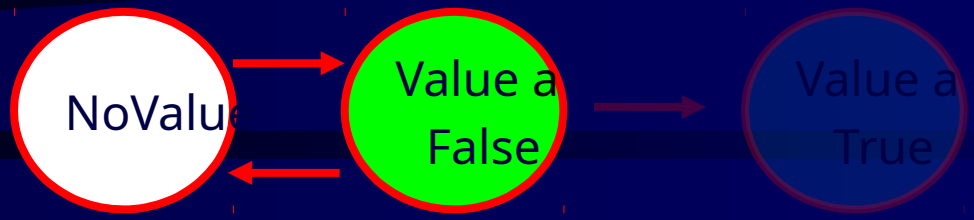
```
myTask :: Task [Person]
myTask = enterInformation “Enter a list of data" []
```





Editors



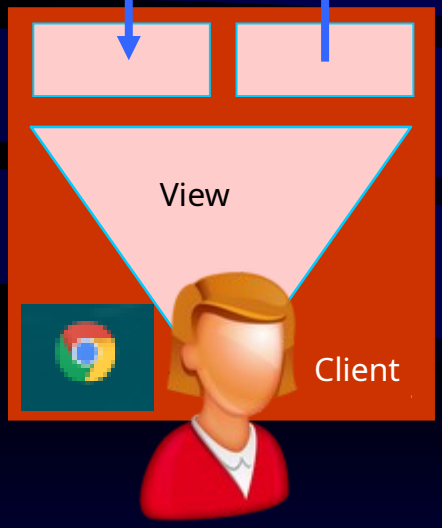


Server - Web Service

view update

Internet

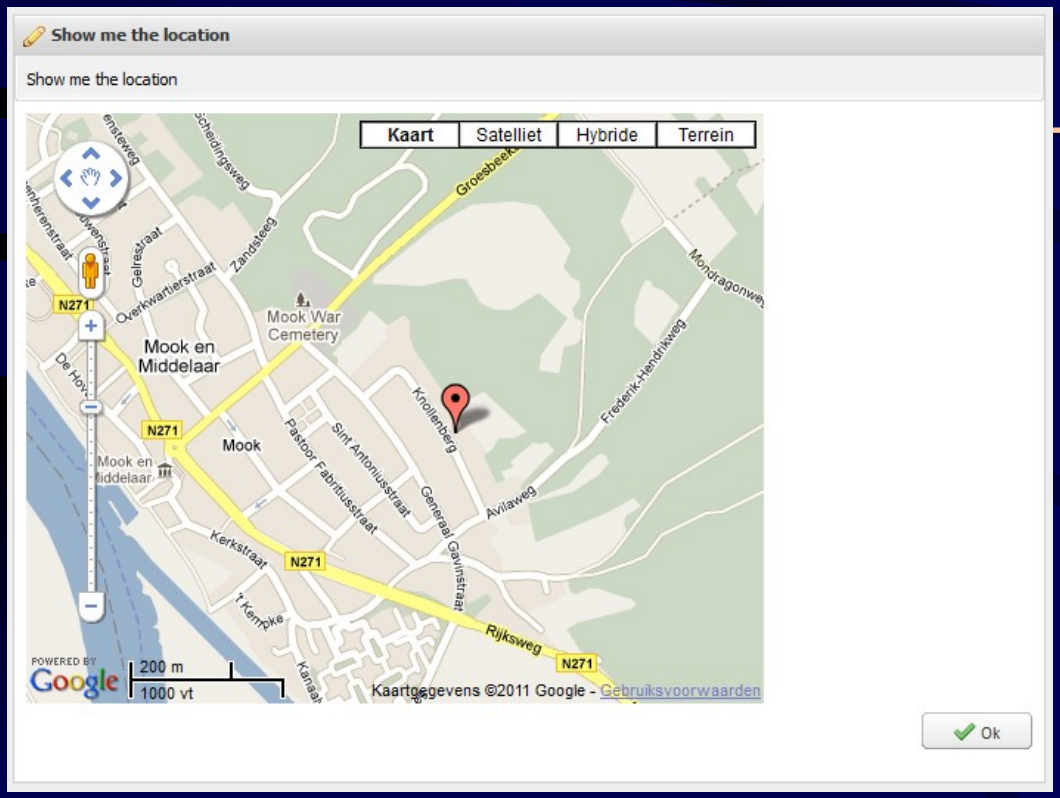
value update

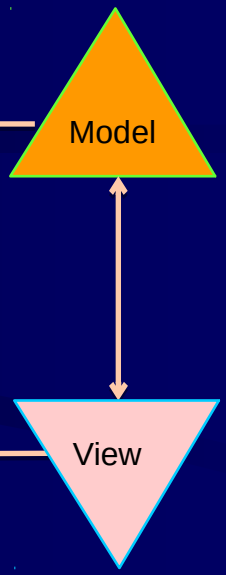


- Editor: interactive basic task producing unstable values
- Task value and client state kept synchronized automatically
- Client efficiently updated on change
- Standard format exchange (JSON)
- Domain Specific Platform independent GUI language
- Makes Platform specific views possible (tablet, smartphone)
- Works with *any* browser (HTML-5 compatible)
- Smart tailor made editors can be defined, local state
- All code defined in *one* Language (Clean)

- Browsers have restrictions (security, limited stack space)
- JavaScript has to be used, and it is very slow (10 times slower than Clean)

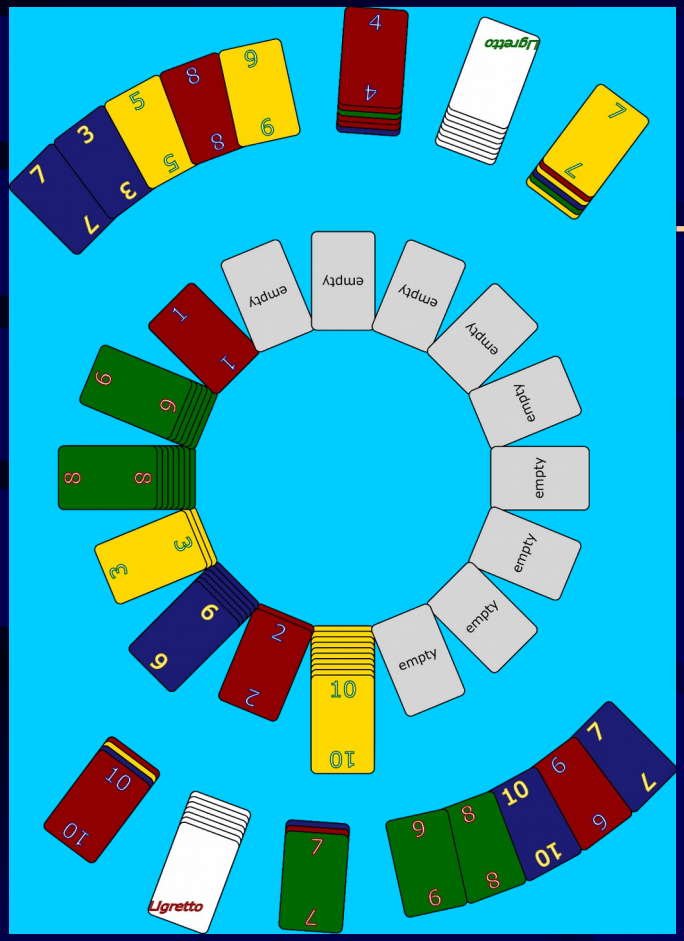
```
import GoogleMaps

pointOnMap :: Task GoogleMap
pointOnMap = enterInformation "Show me the location" []
```

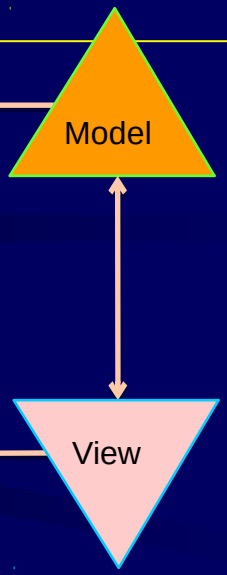




Tailor Made Smart Editors…

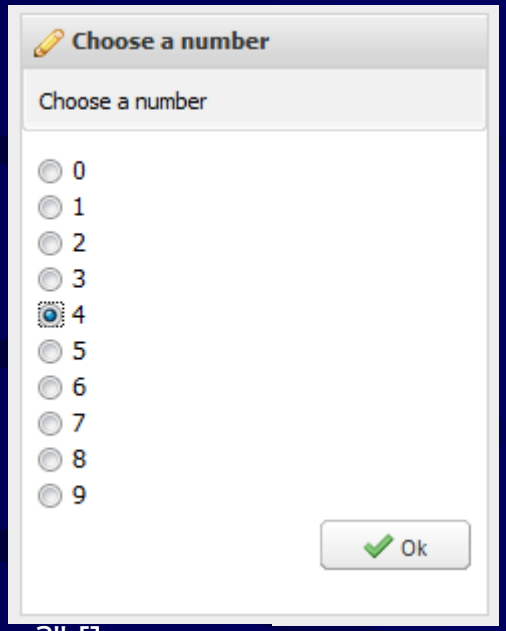
```
playLigretto :: Task Ligretto
playLigretto = enterInformation “Lets Play Ligretto" []
```





Examples of Editors...

```
chooseNumber :: Task Int
chooseNumber = enterChoice "Choose a number" [] [0..9]
```




```
pizzaWith :: Task [String]
pizzaWith = enterMultipleChoice "What do you like on your pizza ?" []
        ["Cheese","Tomato","Ansjofish","Salami"]
```

Interactive Editors (see iTasks.WF.Tasks.Interaction)

Basic tasks: Interactive editor for filling in forms of a certain type:

```
viewInformation     :: p [ViewOption a] a       → Task a        | toPrompt p & iTask a

enterInformation    :: p [EnterOption a]  → Task a        | toPrompt p & iTask a
updateInformation :: p [UpdateOption a a] a       → Task a        | toPrompt p & iTask a

enterChoice         :: p [ChoiceOption a] [a]   → Task a        | toPrompt p & iTask a
updateChoice        :: p [ChoiceOption a] [a] a       → Task a        | toPrompt p & iTask a

enterMultipleChoice :: p [MultiChoiceOption a] [a]        → Task [a]      | toPrompt p & iTask a
updateMultipleChoice :: p [MultiChoiceOption a] [a] [a]  → Task [a]      | toPrompt p & iTask a
```

Interactive Editors (see InteractionTasks.dcl)

Basic tasks: Interactive editor for filling in forms of a certain type:

```
viewInformation :: p [ViewOption a] a → Task a | toPrompt p & iTask a
```

Describe Task to do

Non-standard view of type a

Value of type a

Task returns value of type a

Context Restriction:
It requires instances for type “a”
for a whole *bunch* of predefined
type driven generic functions.
They can all be generated
automatically for any *first order* typ

Types used for interaction tasks...

```
viewInformation    :: p [ViewOption a] a      → Task a    | toPrompt p & iTask a

:: ViewOption a              = E.v: ViewAs   (a → v)                & iTask v
```

Any view type can be chosen for which class iTasks exists

Define how to map value of type a to a value of view type v; identity by default

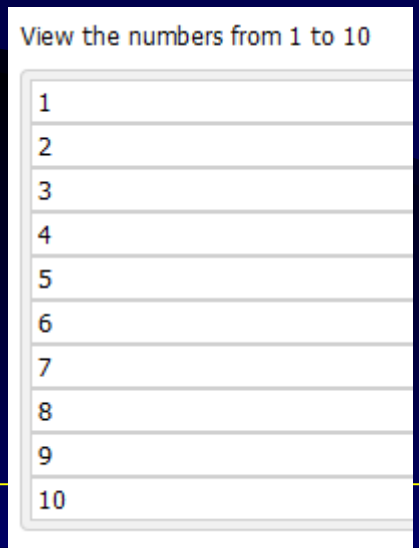
all iTask class functions for type v will be stored in this ADT

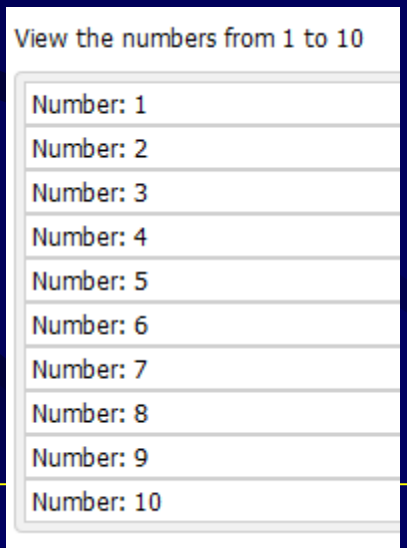
Interactive Editors (see InteractionTasks.dcl)

```
viewInformation :: p [ViewOption a] a → Task a | toPrompt p & iTask a
```

```
myList1 = viewInformation "View the numbers from 1 to 10" [] [1..10]
```

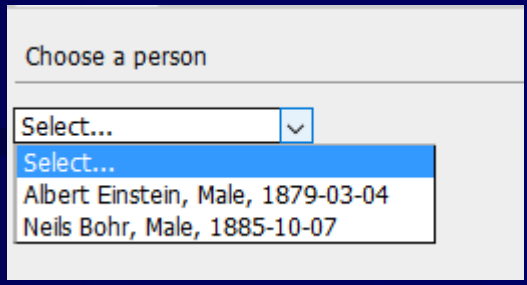
```
myList2 = viewInformation "View the numbers from 1 to 10"
                          [ViewAs (map (\n → "Number: " <+++ n))] [1..10]
```



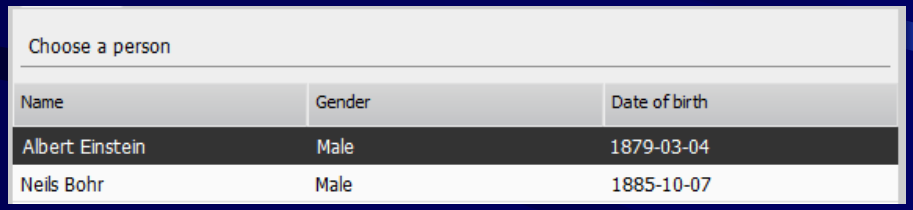


Task first, fine-tune later ...

```
choosePerson :: [Person] → Task Person
choosePerson persons = enterChoice "Choose a person" [] persons
```

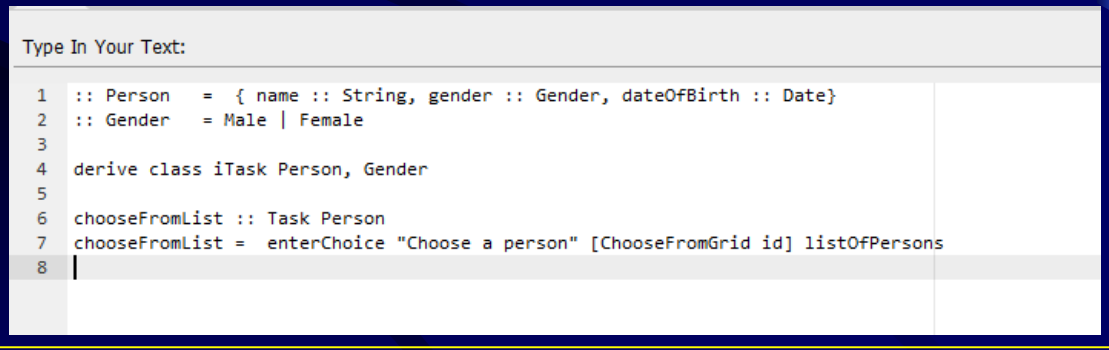



```
choosePerson2 :: [Person] → Task Person
choosePerson2 persons = enterChoice "Choose a person" [ChooseFromGrid id] persons
```




```
myEditor :: Task String
myEditor = enterInformation "Type In Your Text: " [EnterUsing id aceTextArea]
```

GUI's
Editors
Model-View
Lay-out



Types used for interaction tasks...

```
enterInformation   :: p [EnterOption a]  → Task a          | toPrompt p & iTask a

:: EnterOption a              = E.v: EnterAs  (v → a)                  & iTask v
                              | E.v: EnterUsing    (v → a) (Editor v)        & iTask v
```

Use an existing editor as base

With editor combinators one can easily make user definable editor variants

Types used for interaction tasks...

```
updateInformation :: p [UpdateOption a a] a        → Task a        | toPrompt p & iTask a

:: UpdateOption a b = E.v: UpdateAs      (a → v) (a v → b)          & iTask v
```

Combination called a lens

Any view can be chosen

Define how to map value a to

Define how to map the view v back to a value b given the current value a

Context Restriction

Types used for interaction tasks...

```
:: ViewOption a          = E.v: ViewAs   (a → v)                & iTask v
                         | E.v: ViewUsing (a → v) (Editor v)    & iTask v

:: EnterOption a         = E.v: EnterAs  (v → a)                & iTask v
                         | E.v: EnterUsing (v → a) (Editor v)   & iTask v

:: UpdateOption a b = E.v: UpdateAs      (a → v) (a v → b)      & iTask v
                    | E.v: UpdateUsing   (a → v) (a v → b) (Editor v)  & iTask v

:: ChoiceOption a        = E.v: ChooseFromDropdown (a → v)      & iTask v
                         | E.v: ChooseFromCheckGroup (a → v)    & iTask v
                         | E.v: ChooseFromList (a → v)          & iTask v
                         | E.v: ChooseFromGrid (a → v)          & iTask v

...
```

Types used for interaction tasks...

```
viewInformation    :: p [ViewOption a] a      → Task a    | toPrompt p & iTask a

class toPrompt p     where     toPrompt :: p → UI

instance toPrompt (), String, (String, String) , [p] | toPrompt p , ...


class iTask a     | gEditor{|*|}       // Interactive Form generation

                  , gText{|*|}         // Textual visualization

                  , JSONEncode{|*|}       // Serialization
                  , JSONDecode{|*|}// De-serialization

                  , gDefault{|*|}            // Default Value generator
                  , gEq{|*|}                 // Equality

                  , TC                // Conversion to and from type Dynamic
```

Types used for interaction tasks...

```
generic gEditor a           | gText a, gDefault a, JSONEncode a, JSONDecode a
                    :: Editor a

generic gText a             :: TextFormat (Maybe a) → [String]

generic gDefault a          :: a

generic JSONEncode a        :: a                → [JSONNode]
generic JSONDecode a        :: [JSONNode]       → (Maybe a, [JSONNode])

generic gEq a               :: a a              → Bool
```

TC is a special class automatically instantiated by the compiler for every “known” type...

```
toDynamic                   :: a          → Dynamic     | TC a
fromDynamic                 :: Dynamic    → a           | TC a
```

Several Interactive Editors Variants

All instantiations of *one-and-the-same Core – Editor*...

```
viewInformation          :: p [ViewOption a] a                    → Task a      | toPrompt p & iTask a

enterInformation   :: p [EnterOption a]                 → Task a      | toPrompt p & iTask a
updateInformation        :: p [UpdateOption a a] a                → Task a      | toPrompt p & iTask a

enterChoice        :: p [ChoiceOption a] [a]            → Task a      | toPrompt p & iTask a
updateChoice             :: p [ChoiceOption a] [a] a              → Task a      | toPrompt p & iTask a

enterMultipleChoice      :: p [MultiChoiceOption a] [a]                → Task [a]    | toPrompt p & iTask a
updateMultipleChoice     :: p [MultiChoiceOption a] [a] [a]            → Task [a]    | toPrompt p & iTask a

...
```