Getting a Grip on Tasks that Coordinate Tasks

Rinus Plasmeijer

Radboud University Nijmegen

http://wiki.clean.cs.ru.nl/Clean

http://wiki.clean.cs.ru.nl/ITasks

Peter Achten (ru) - Pieter Koopman (ru) - Thomas van Noort (stw) - John van Groningen (ru) - Bas Lijnse (navy) - Jan Martin Jansen (navy) - Steffen Michels (SITPro) - Jeroen Henrix (SITPro) - Laszlo Domoszlai (ELTE)

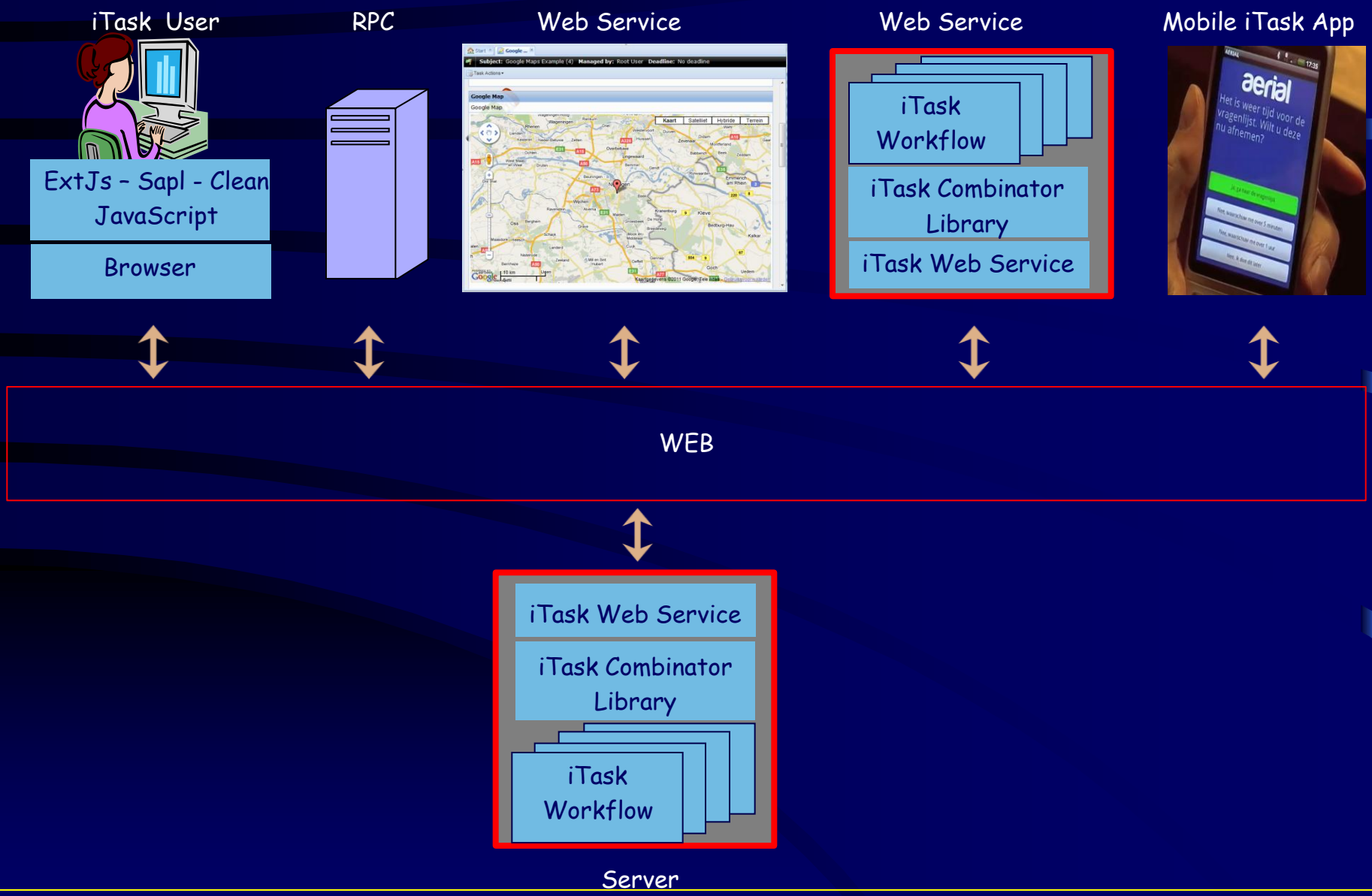
(ICFP 2007)

itasks

dynamic workflow system

- It is a combinator library written in Clean
- It is a toolbox for the rapid development of WFMSs
- It is a Domain Specific Language embedded in Clean

iTask Server Coordinates Tasks executed by Clients



It is declarative...

- *"Declarative specification*
  of data and tasks
  is sufficient for generating
  an executable workflow"

- Abstract from implementation details as much as we can using *type driven generic functions*

  I/O handling,
  communication,
  JSON / XML exchange,
  web form generation, web form updating,
  persistent storage, …

i - Tasks - Embedded Workflow Description Language

basic tasks: Task a - unit of work delivering a value of type a

- Filling in a web form, web-service, OS-call (time, date), application call, database access

+ combinators for combining tasks

- *Common usage*
  - define order of tasks (sequential, parallel)
  - assign properties to tasks (worker, priority, deadline),
- *Exceptional usage*
  - workflow / task process handling (create, waitFor, suspend, kill)
  - exception handling

+ Clean host language features

- recursive -, higher order -, polymorphic -, overloaded -, generic - functions
- strongly typed + dynamic typing

Examples of basic tasks for filling in forms

```
enterInformation    :: d      → Task a    | descr d & iTask a
updateInformation   :: d a    → Task a    | descr d & iTask a
```

```
class iTask a
    | gVisualize {|*|}          // information for form creation
    , gUpdate {|*|}             // form update
    , gEq {|*|}                 // equality test
    , gDefaultMask {|*|}        // form status
    , gVerify {|*|}             // predicate value has to obey
    , JSONEncode {|*|}
    , JSONDecode {|*|}          // JSON encoding - decoding
    , XMLEncode {|*|}
    , XMLDecode {|*|}           // XML encoding - decoding

    , TC a                      // serialization – de-serialization
```

A very small *complete* example I

```
module example

import iTasks

Start :: *World → *World
Start world = startEngine [workflow "demo task" myTask] world

myTask :: Task Int
myTask = enterInformation "Please fill in the form:"
```

iTask Client

delegate - Mozilla Firefox

Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help

http://localhost/

Meest bezocht Aan de slag Laatste nieuws

delegate | Reisplanner › NS reizigers

Tasks

Demo Example Int

Refresh worklist

Welcome rinus<rinus> Logout Debug...

Subject	Priority	Progress	Managed by	Date	Deadline
Demo Example Int	Normal	Active	rinus<rinus>	04 Mar 2011 12:00:39	No deadline

Start | Demo Ex...

Subject: Demo Example Int (1) **Managed by:** rinus **Deadline:** No deadline

Task Actions

Demo Example Int

Please fill in the form:

Ok

Task description

Demo Example Int

Fill in the form

Start task

Klaar

A very small *complete* example II

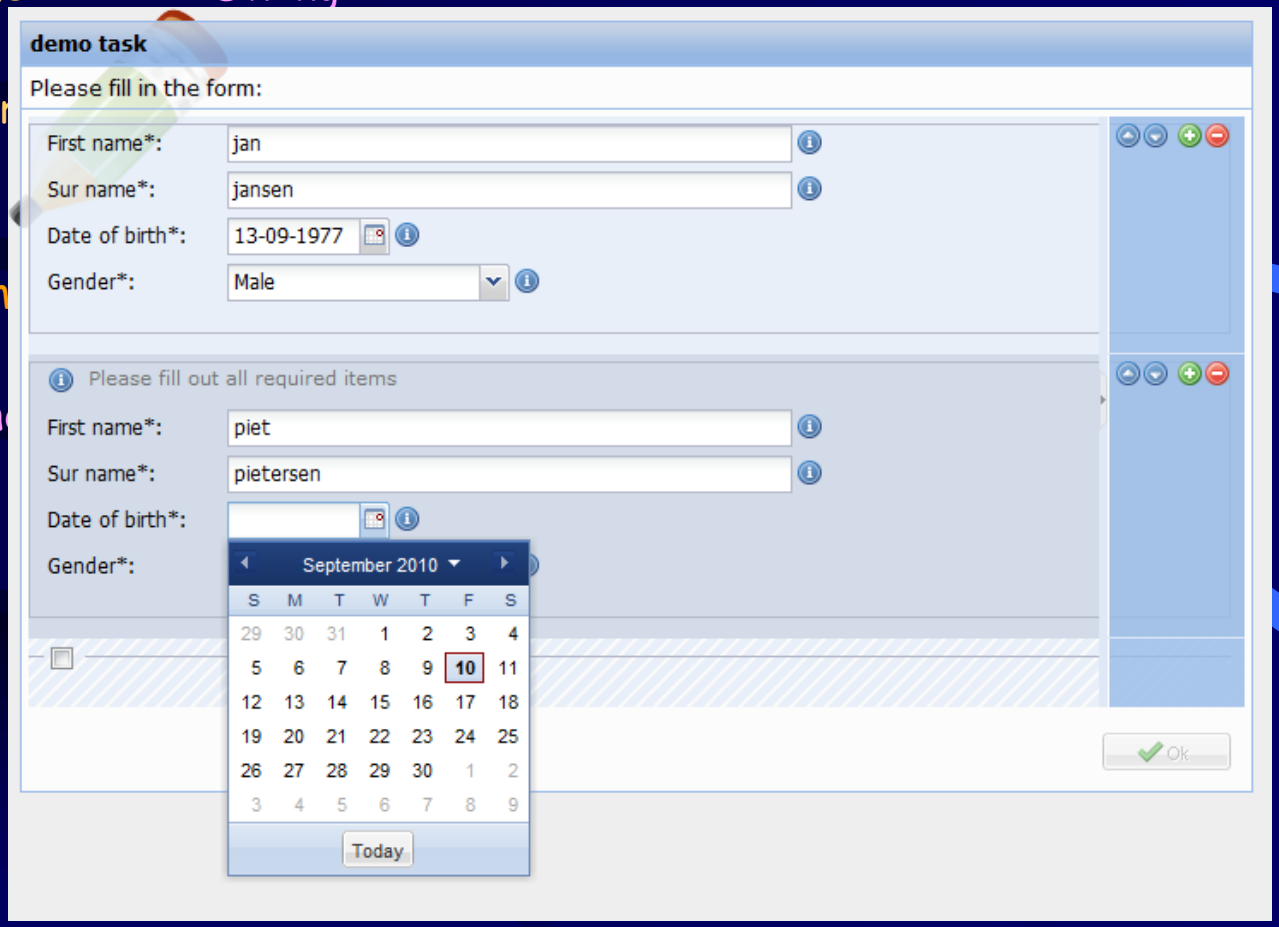
```
myTask = enterInformation "Please fill in the form:"
```

A very small *complete* example II

```
myTask :: Task [Person]
myTask = enterInformation "Please fill in the form:"

:: Person  =        { firstName        :: String
                    , surName
                    , dateOfBi
                    , gender
                    }
:: Gender  =        Male | Fem

derive class iTask Person, Gen
```



Core Combinators

Basic combinator: interactive editor for filling in forms of a certain type:

```
updateInformation :: d a                    → Task a        | iTask a & descr d
```

Main task: define task properties (who has to work on it, priority, deadline):

```
(@:)  infix 3  :: p (Task a)                → Task a        | iTask a & property p
```

Sequencing of tasks using monadic bind >>= and return:

```
(>>=)  infix  1  :: (Task a) (a → Task b)   → Task b        | iTask a & iTask b
return           :: a                       → Task a        | iTask a
```

Parallel evaluation of tasks:

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

With just a few combinators *many* frequently occurring flows can be defined

semantics: term rewriting system (IFL 2008, PEPM 2011)

Open question: What kind of combinators do we *really* need ?

Core Combinators

Basic combinator: interactive editor for filling in forms of a certain type:

```
updateInformation :: p a                    → Task a        | iTask a & property p
```

Main task: define task properties (who has to work on it, priority, deadline):

```
(@:)  infix 3  :: p (Task a)                → Task a        | iTask a & property p
```

Sequencing of tasks using monadic bind >>= and return:

```
(>>=)  infix  1  :: (Task a) (a → Task b)   → Task b        | iTask a & iTask b
return           :: a                       → Task a        | iTask a
```

Parallel evaluation of tasks:

```
parallel  :: ([a] → Bool) ([a] → b) ([a] → b) [Task a] → Task b   | iTask a & iTask b
```

<u>Open question: What kind of combinators do we *really* need ?</u>

Defined many toy applications: (see iTask distribution)

AllExamples - Mozilla Firefox

Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help

http://localhost/

Meest bezocht Aan de slag Laatste nieuws

AllExamples | Reisplanner › NS reizigers

Tasks

- Admin
- Examples
  - Business
  - Higher order
  - Miscellaneous
  - Geo tracker
  - Communication
    - Newsgroups
    - Mail
    - Broadcast
    - Mail with confirmation
    - Mail with forced reply
    - Delegate Instruction
    - Chat with someone
  - Crisis response
  - Changes
  - Interactive Workflows
  - Shared Variables
    - Text-Lines (grouped tasks)
    - Calculate Sum
    - Balanced Binary Tree
    - Merge Test (List)
    - Merge Test (Documents)
    - Google Maps Example
    - Sorted List
    - Formatted Text
    - Choose or add
  - Graphical Editors
    - Graphical Editor
    - Petri net editor
    - Gin editor
    - Shared Petri net editors
    - Shared Gin editor
  - Workflow starter
- General
  - Ask opinions

Task description

Google Maps Example

Start task

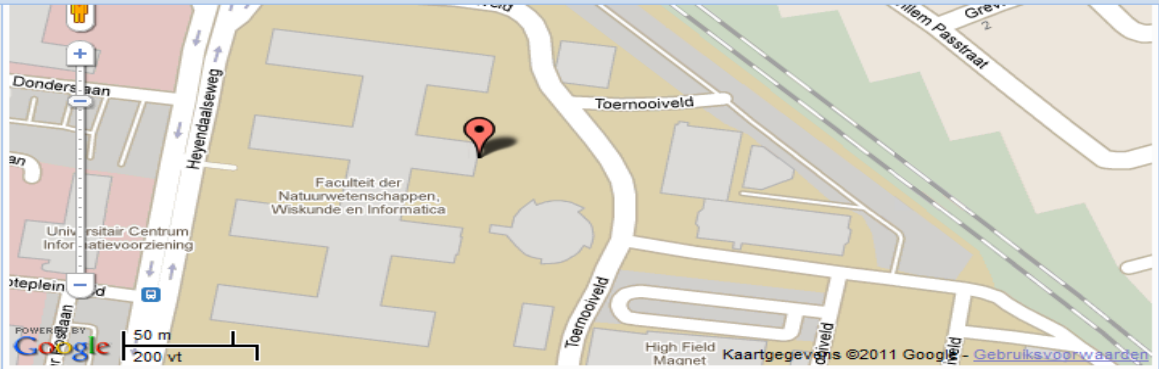
Refresh worklist — Welcome Root User <root> — Logout — Debug...

Subject	Priority	Progress	Managed by	Date	Deadline
Shared Gin editor	Normal	Active	Root User <root>	04 Mar 2011 11:36:55	No deadline
Google Maps Example	Normal	Active	Root User <root>	04 Mar 2011 11:50:32	No deadline

Start | Shared ... | Google ...

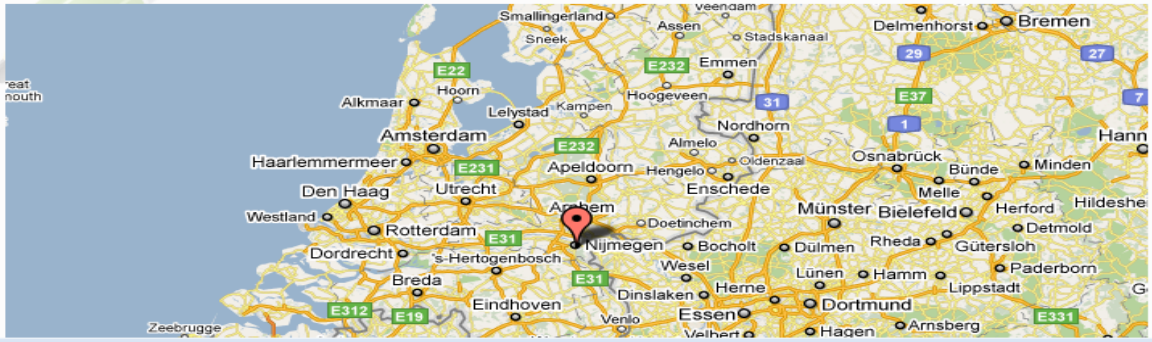
Subject: Google Maps Example (2) **Managed by:** Root User **Deadline:** No deadline

Task Actions



Overview Map

Overview Map



Klaar

"Real" Prototype Applications using iTasks

Simple workflow:

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

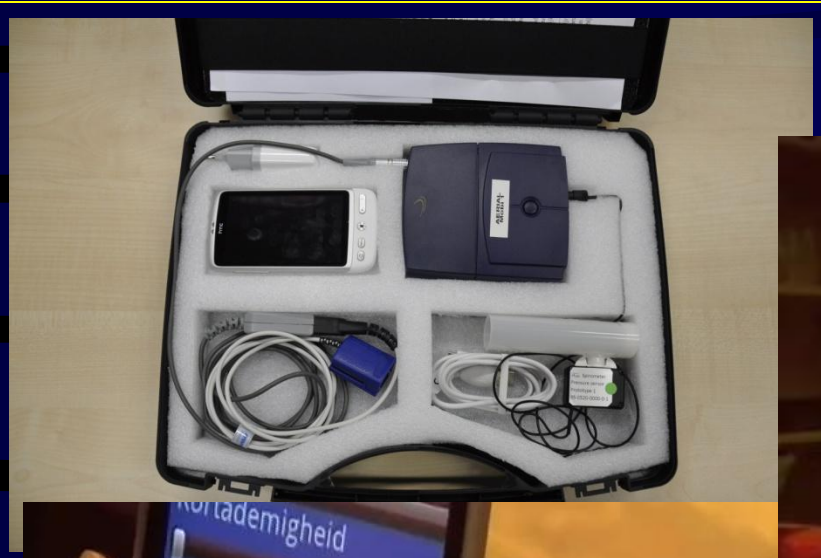
Real real-life workflow:

- Crisis Management:

  Capturing the Netherlands Coast Guard's Search And Rescue Workflow

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

Home Healthcare project

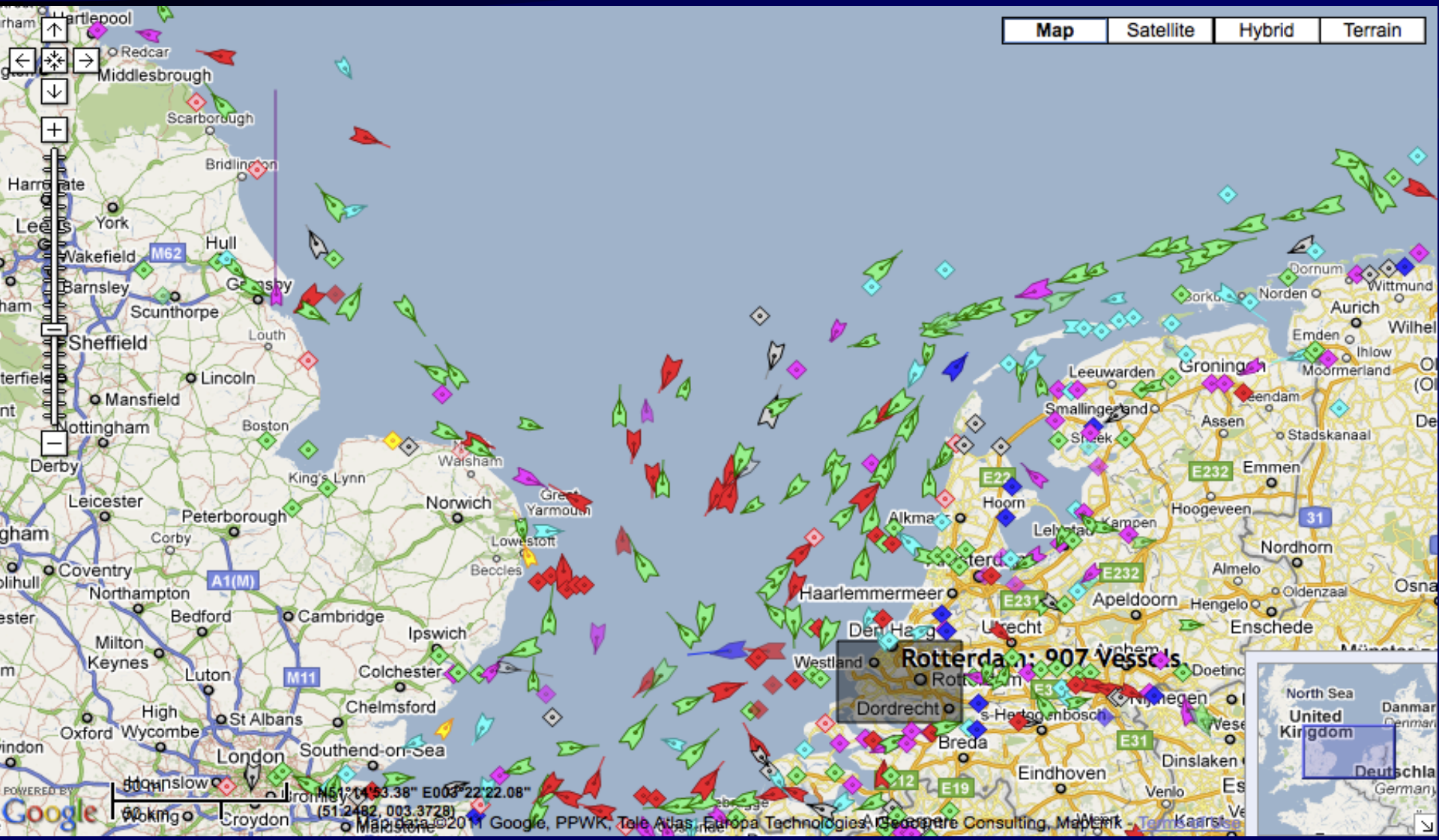


A questionnaire is answered by touchscreen

Measurements are sent wirelessly to the phone

Coast Guard Search And Rescue

Coast Guard Search And Rescue



What did we learn ?

- Coordination panels should not be build-in but become user-definable tasks as well
  - E.g. the iTask main system panel
- Sharing of information between tasks needed to monitor developments
  - Also needed for many to many communication
- Forms are not enough: need to be able to specificy GUI's (windows, menus, …)
- One cannot foresee everything: we have to be able to change running workflows

Currently designing + implementing version 3.0

- All this functionality should be offered by the new AFT
- Yet: we expect to ask only a very few Swiss Army-Knife combinators

What have we done so far ?

Small extensions to Clean:

- *Added (Generic) context restriction in types*
- *Allow overloaded and generic functions in dynamics*
- *Allow generic functions to be overloaded in generic functions*

basic tasks:

- updateInformation :: d (View i v o) [Action i] (Shared i o) → Task (Event, Maybe i)
  | iTask i & iTask v & iTask o & descr d

combinators for combining tasks

Common usage:

- parallel :: d (Merge a ps b) [CTask a ps] [Task a] → Task b
  | iTask a & iTask ps & iTask b & descr d

Exceptional usage:

- workflow / task process handling
- exception handling
- *change handling*

More Future work

- Improve Practical Applicability
  - Robustness ? Performance ? Scaling ? Security ? Software evolution ?
  - Embedding with existing databases
    - ORM specification used to map RDB <-> Clean data types
  - Distributed Servers
  - Add iTasks running on the client, now in JavaScript
  - How to offer dynamic change to the end user ?
  - Reasoning ? Proving ? Testing ?