QuidQuid: using Haskell to Turn the Internet on its Head.

Pasqualino "Titto" Assini
Quid2.Org
tittoassini@ gmail.com

Why the Internet Needs to be Turned on Its Head.

If names are not rectified … people will not know how to move hand or foot. (Confucius, Analects 13:3)

DNS + WWW	As It Is.	As It Should Be.
Information Organisational Principle.	By Publisher. Federation of private information namespaces.	By Meaning. Both private and shared information namespaces.
Information Retrieval.	Users chase information (Publisher-Oriented).	Information flows to interested users (User-Oriented).
Information Structure.	Addresses (not really names) + primitive (MIME) types, flat.	Addresses, names, data types, functions, expressions, types, compositional.
Information Presentation.	Same content, different presentation. Zillions of different interfaces. Mix-up of content and presentation.	Single adaptive interface (one page to rule them all). Separation of content and presentation.

Hacking, hacking, hacking

People have realised these limitations for a long time. In fact, the history of the Internet can be seen as a series of increasing sophisticated patches for the "wrong organisation" and "lack of expressivity" problems:

- Google :: Concept -> IO [Address]
- Wikipedia :: Concept -> IO Article
- Facebook :: PersonName -> IO HomePage
- BitTorrent :: Name -> IO [File]
- CheapFlights.Com ::
  - From -> To -> Time -> IO [Flight]

However, all these applications are either proprietary or limited in scope.

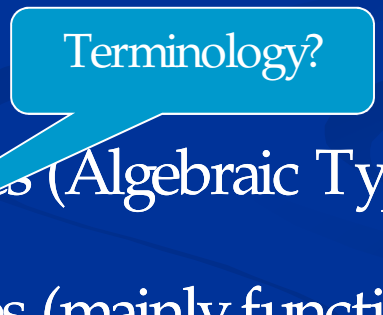
And Technologies to Match

- Semantic Web
- Web Services (RPC)
- Semi-Proprietary Google/Facebook/Yahoo Web Services.
- Orchestration Languages.
- Publish/Subscribe Systems.
- Content-Addressable Systems (Distributed HashTables).
- …. not terribly successful so far.

So What Would Fix It?

Example: perform a web search filtering out inappropriate results:

```
Data.List.filter Search.isKosher $ Search.search "sex and the city"
-> [
[[..Google hits..], [..Bing hits]] as filtered by the Catholic Church
,[[..Google hits..], [..Bing hits]] as filtered by the Free Love Society
]
```



The elements of a solution:

- A way of **defining** typed "closed" values (Algebraic Types, functions).
- A way of **declaring** typed "open" values (mainly functions).
- A way of providing alternative **definitions** of the open values.
- A way of creating more complex terms by functional application.
- A way of evaluating the resulting expressions.

Doing It in Haskell

- Playing the "WWW Trick":
  - Take an existing technology, so far used mainly in a local environment.
  - Simplify it to the bone.
  - Extend it to work in a distributed environment.

WWW = globalise (simplify hyperText) globalNamingSystem netProtocol

QuidQuid is Latin for "Whatever".
Ideas for a better name?

Quid^2 = globalise (simplify haskell) globalNamingSystem netProtocol

Haskell Goes Global

Simplification:

- Simpler Syntax.
- Simpler Type System.

Functions and data in a distributed language will be monadic (values are usually returned by a remote agent), how can we change the syntax to reflect that?

Global Naming System: an evaluation context is a set of uniquely identified (e.g. by a hash-code) and uniquely named modules.

Extensions:

- Non-Determinism (a la Curry) to support multiple distributed implementations of open values.
- Security (Big Big trouble).

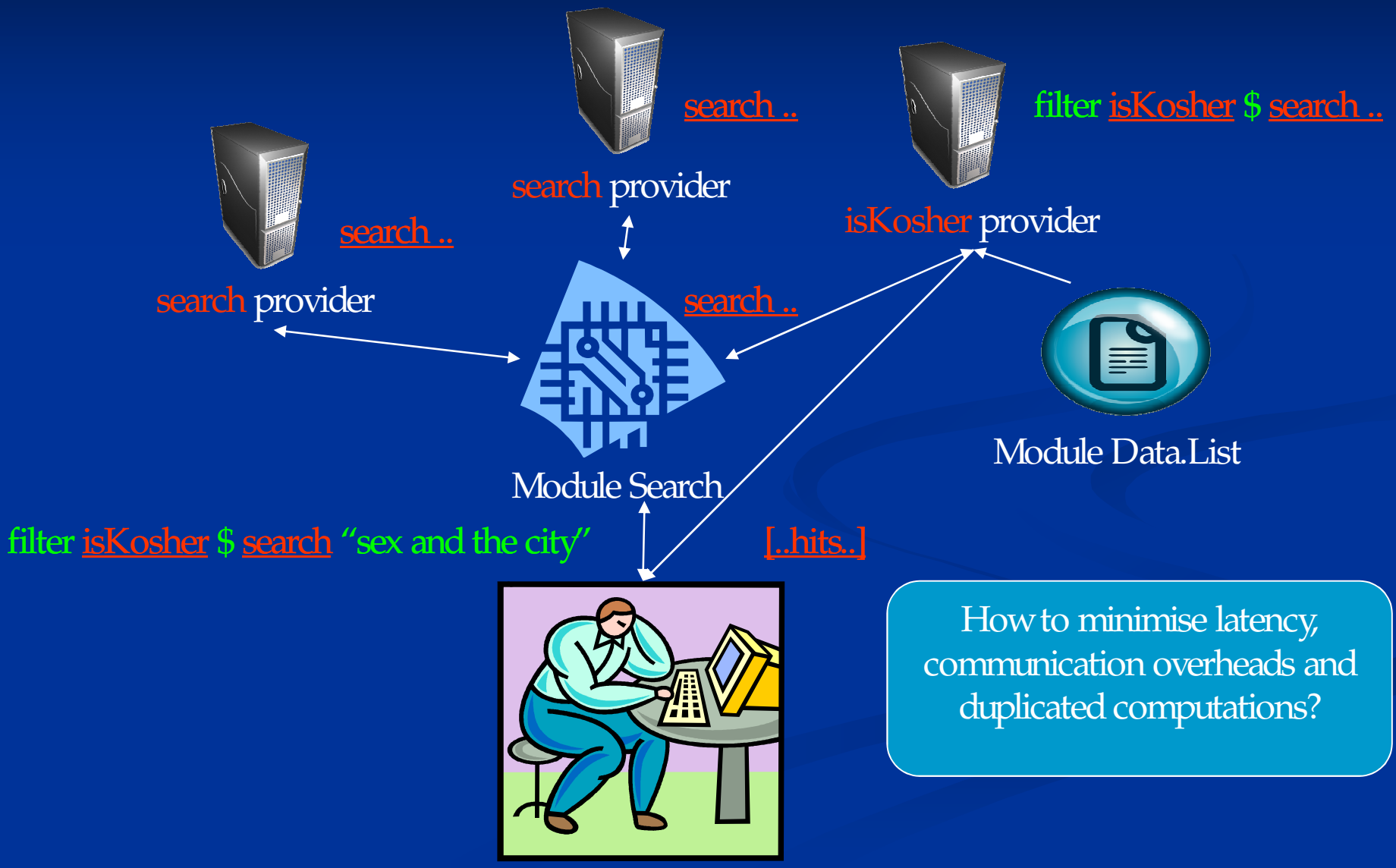
How much security needs to be embedded as a primitive in the system and how much can be defined in it (e.g. isKosher) ?

Quid^2 Modules ≈ Haskell

```
module Search where
import Data.Bool

-- Haskell-Like Data Types
data Hit = Hit { url::URL,title::Title}
type URL = String
type Title = String

-- Declaration without definition
-- indicates a non-deterministic
-- externally defined value.
search :: String -> [Hit]

isKosher :: Hit -> Bool
```

```
module Data.List where

-- Declaration with definition,
-- a plain value.
filter :: (a -> Bool) -> [a] -> [a]
filter pred []    = []
filter pred (x:xs)
  | pred x        = x : filter pred xs
  | otherwise     = filter pred xs
```

Distributed Evaluation.



Haskell Provider/Client

```
main = do
  -- Connect to QuidQuid
  connectVia "http://quid2.org/api"

  -- Register the implementation of one or more functions
  Quid2.Sig.Search.def_search googleSearch

  -- Evaluate an expression in source format (e.g. as typed by an user)
  result :: [[Hit]] <- evaluate "Search.search \"sex and the city\""

  -- Or in code
  result2 <- runQ $ do
    Quid2.Sig.Search.search (return "sex and the city")
```

Monad needs to support non-determinism, laziness, (non-strict evaluation, sharing of results), IO. As the ones in the *explicit-sharing* or *Orc* packages.

A Modest Plan

1. Prototype/Proof of Concept:
    1. Centralised Router.
    2. Haskell API.
    3. JavaScript API.
2. Distributed Development Environment targeted at the Haskell Community:
    1. Distributed Editing, Storing, Compilation and Execution of Quid^2, Haskell, JavaScript, HTML/CSS code.
    2. Web Adaptive Interface: type an expression and the returned value (a module, a document, an Int, a graph, a table, whatever) is displayed by an appropriate viewer.

HELP WANTED!