Monadic design patterns for the Web

Putting data, query and analytic in the hands of the people who use them

L.G. Meredith

http://svn.biosimilarity.com/src/open/talks/MonadicDesignPatternsForTheWeb.pdf

Overview

- Confessions of a mad (computer) scientist
- An architecture for semantic search
- A path to this architecture
- Informed by the monadic design pattern
- Demo

Confessions of a mad (computer) scientist

- What am i passionate about?
  - A revolutionary approach to search not based on "semantic web" techniques and yet results in genuine semantics-based search capability.
  - The technical details of this idea have taken 5 years to work out

A path to this goal

- (Behavioral and structural queries in physical systems) Find all the pathways, p, in <SignallingPathwayDB> such that when <Reagent> is added in concentration K p eventually reaches states in which we see these <Reagent1>,...,<ReagentN> in concentrations K1,...,KN.

- (Behavioral and structural queries in logical systems) Find all the implementations of message queue in http://svn.myrepo.org such that the queue is FIFO and use the Xerces XML parser for XML-encoded messages

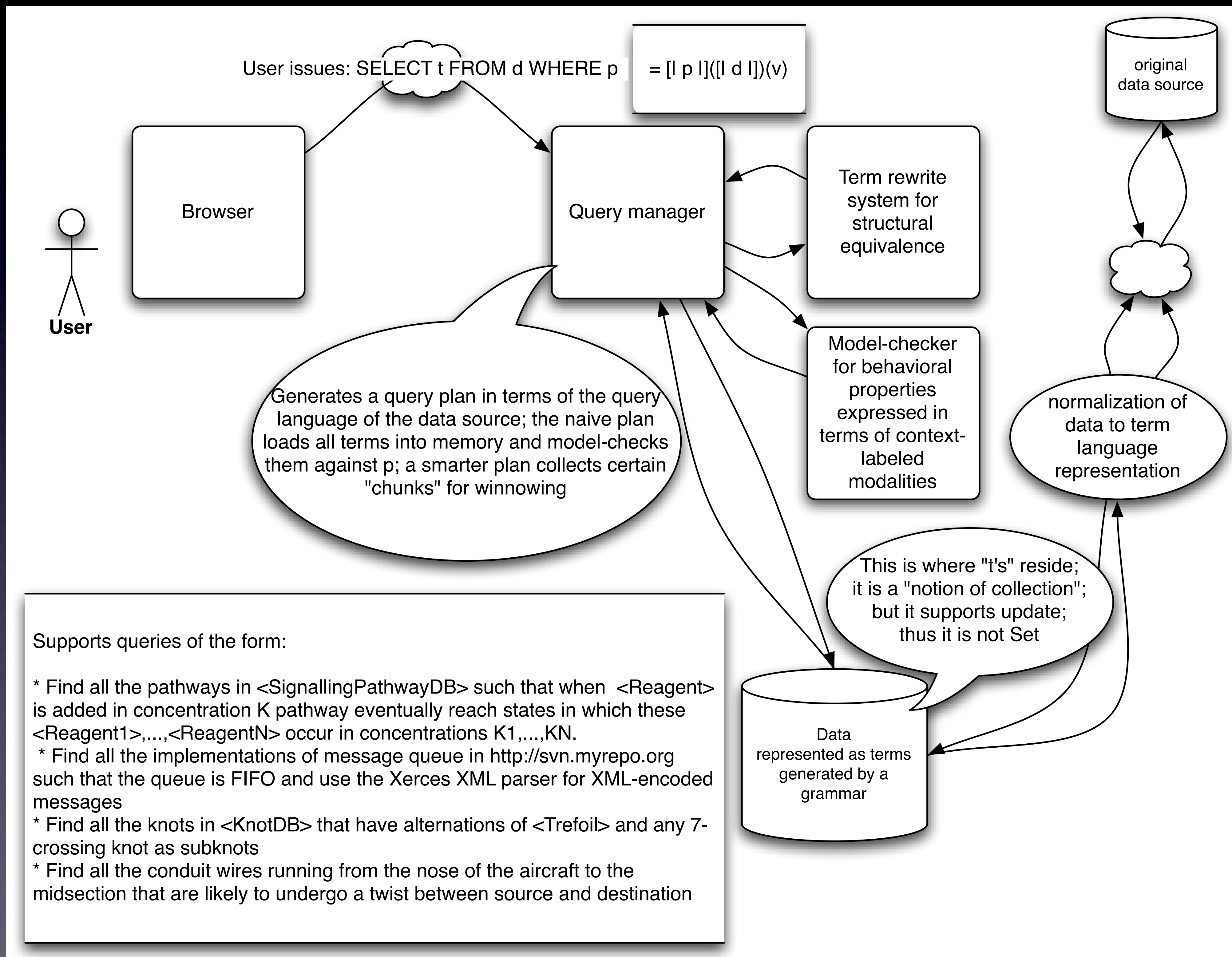
- (Structural queries in physical systems) Find all the knots in <KnotDB> that have alternations of <Trefoil> and any 7-crossing knot as subknots

- (Geometric queries in physical systems) Find all the conduit wires running from the nose of the aircraft to the midsection that are likely to undergo a twist between source and destination

Confessions of a mad (computer) scientist

- How am i going to build this?
  - Get an active open source community excited by the ideas and let the magic happen!
  - Everybody wins if this can be realized at the right scope and scale

Architecture



A path to this goal

- What is a collection?
- What is an item?
- What is an analytic?
- A machine for mashing things up
- Questions?

What is a collection?

- Back in undergraduate maths we knew what a collection was: it was anything you could write like this

```
{ x in D | C1(x), C2(x), ..., Cn(x) }
```

- Later, when we got jobs in computing it was anything you could write like this

```
select x from D where C1(x)and ...
and Cn(x)
```

What is a collection?

- Then along came Haskell (and Python, and C#, and F#, and Scala, and XQuery, and ...) and the difference between these two ways of thinking about collections began to blur
- [x for x in S if x % 2 == 1] (Python)
- [x | x <- S, odd x] (Haskell)
- ...

What is a collection?

This key language feature came about because some very clever people in computing and maths saw some very deep patterns that unified a lot of different phenomena

What is a collection?

And a few large scale companies and institutions knew a good thing when they saw it

What is a collection?

- The fact is these are just two syntaxes for ***one*** underlying language for denoting collections

```
{ x in D | C1(x), C2(x), ..., Cn(x) }
```

```
select x from D where C1(x) and ... and Cn(x)
```

In scala

```
for( x <- D if C1(x) && ... && Cn( x ) ) yield x
```

- That can be backed in memory, or storage of many different shapes and kinds

What is a collection?

- And that language has already proven itself to be an excellent way to represent data sets
- ... for the last 30 years of enterprise computing
- ... and the last 100 years of mathematics

What is an item?

- Back in undergraduate maths we knew what an item was: it was anything you could put between

  `'{'` and `'}'`

- Later, when we got jobs in computing it was anything you could stick in a table and retrieve with statements like

  ```
  select x from D where C1(x) and ...
  and Cn(x)
  ```

What is an item?

- Then along came the Internet and it became important that individuals and companies that didn't have common access to a single store be able to exchange items
- So.... items were things you could put between `<tag>` and `</tag>`

What is an item?

- But tag soup is pretty unsavory... so several proposals for what an item is have been in play
  - DTD, XSD, RelaxNG, ...
- All of these line up with the notion of algebraic data type as witnessed in Haskell and F# and Scala and OCaml and Erlang and ...

What is an item?

Some 80/20 rules to keep in mind:

- Mutable data types don't make very good message types
- And they don't make very good record types (for storage media)
- As the Telco's figured out a long time ago the intersection between algebraic data types and what's described by an EBNF grammar is more than adequate

What is an analytic?

- Things are looking pretty rosy
  - A collection is anything denoted by an existing store or a select-from-where construct
  - An item is anything described by an EBNF grammar
- There's always some little detail

What is an analytic?

- Sometimes items have parts
  - ‘Menus’ have ‘items’ and ‘submenus’
  - ‘Graphs’ have ‘vertices’ and ‘edges’ and ‘subgraphs’
- ... and sometimes they have *lots* of parts

What is an analytic?

- We never write out all the integers, we just say Int

```
lazy val intStrm =
  List( 0 ).toStream +++ ( intStrm map { _ => _ + 1 } )
```

- ... but more importantly the web 2.0 is teaching us that what initially looks like a small thing

```
type TwitterMsg = Char [140] // not legal Scala
```

becomes pretty valuable when it is iterated into a stream

```
abstract class TwitterStream
case class Twitterful( msg : TwitterMsg, strm : TwitterStream )
extends TwitterStream
```

What is an analytic?

- And the streams are bundled

```
case class Tweeter( myStrm : TwitterStream, followers :
[TwitterStream] )
```

- And then we mine them for common patterns

```
def hottopic( tweeters : List[Tweeter] ) =

 tweeters match {

  case ( out, follows ) :: ts =>

   for ( x <- merge( follows )

         if ( follows precedes x out ) )  yield x

  ... }
```

What is an analytic?

- Notice hottopic is *also* a stream
- Many interesting items have parts (components) that are programmatically specified -- we refer to the parts programmatically either because
  - there are too many of them or
  - they arrive on an as-needed basis
  - because we gain a significant information compression advantage in our exchange of these items

What is an analytic?

We are familiar with this idea from simple arithmetic.

- When there are a reasonable number of numbers to add we write out the addition, like so: $a_1 + a_2 + ... + a_n$
- When there are more than a reasonable number of numbers to explicitly write out the addition, we write it out in indexed form, like so: $\sum_{i \in I} a_i$

In this example '+' provides the composition and numbers are both the container and the component (this is the operational definition of what it means for a data type to be compositional: instances can be in the role of container and in the role of component)

What is an analytic?

- Likewise, when an item is compositional in nature (like numbers or menus or graphs) then we need to add *indexed* forms of composition to their *term* language
- Example:

```
case class Menu[A]( items : List[(A,Menu[A])] )
```

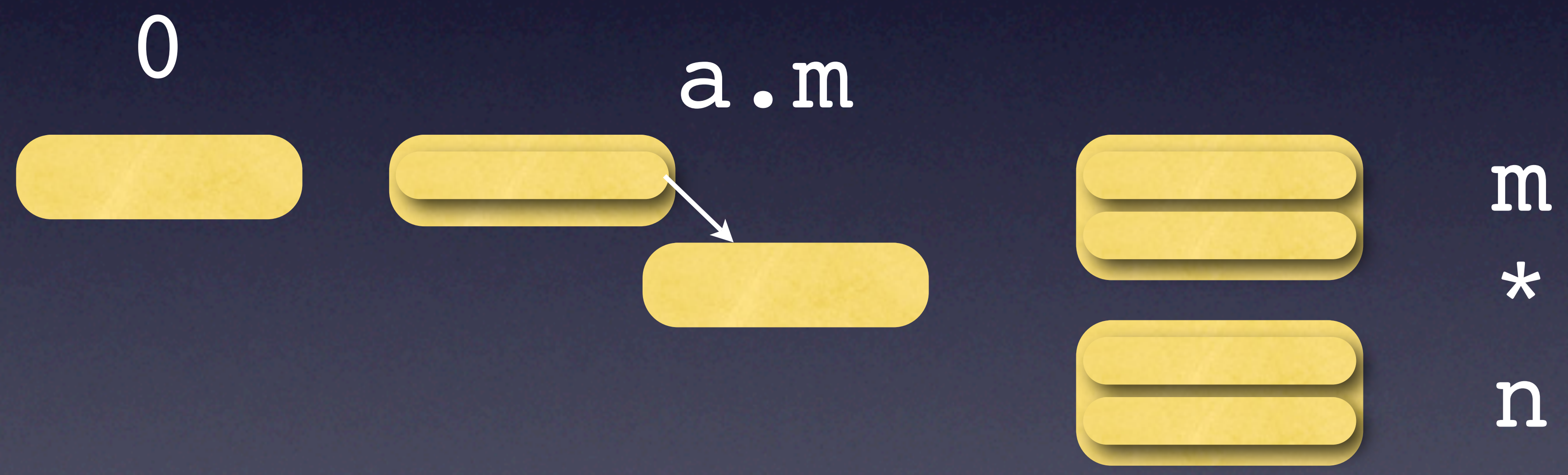
really needs to be

```
abstract class Menu[A]

case class MenuExplicit( items : List[(A , Menu[A])] ) extends Menu[A]

case class MenuProgrammatic(

   mptn : MPtn[A], msrc : MGen[A], mcond : List[MenuCondition[A]]

) extends Menu[A]
```

What is an analytic?

A more compact term language for menus

```
M a  ::= 0 | a . M a | M a * M a
```



What is an analytic?

- This data structure

```
M a  ::= 0 | a . M a | M a * M a
```

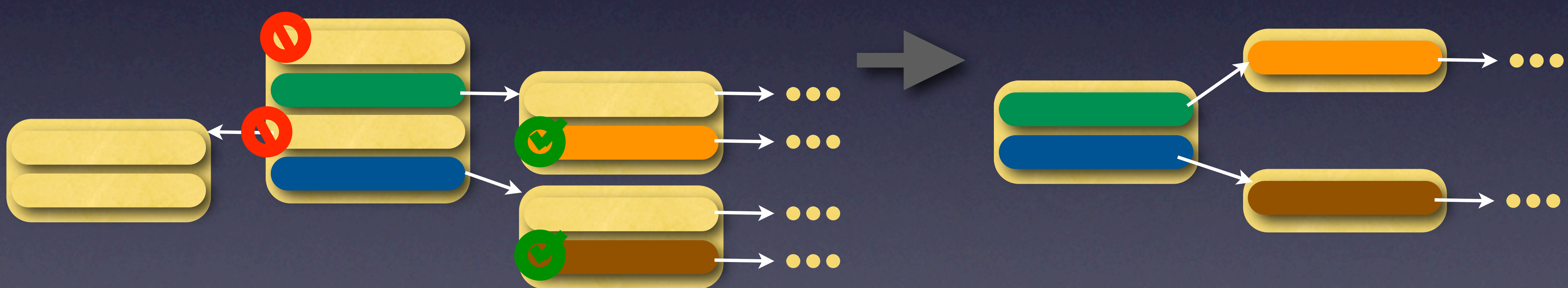
- Becomes

```
QM a ::= 0 | a . QM a | QM a * QM a

           | { MPtn a : (MPtn <- Ma)*, (MSat a)* }

MPtn a ::= M (LitOrVar a)

MSat a ::= MPtn a in MCnd a

MCnd a ::= true | ~ MCnd a | MCnd a or MCnd a

             0 | a . MCnd a | MCnd a * MCnd a
```

What is an analytic?

```
{ x . r : x . l*r <- m, r in true.~0, l in true.~0 }
```

represents a kind of "projection" that picks out submenus satisfying a more global property



A machine for mashing things up

- (QM a) is <u>*calculated*</u> from (M a)
  - Plus a 'notion of collection' -- first line of MCnd grammar says this is something that is a boolean lattice -- hence essentially the powerset of the set of all (M a)'s
  - A grammar giving formulae of multiplicative LL would have implied the notion of collection is powerset of sequences of (M a)'s -- another gives streams
- This means the Q process is applicable to any type with compositional structure -- hence *mashable*

What is an analytic?

- This data structure

```
M a  ::= 0 | a . M a | M a * M a
```

- Becomes

```
QM a ::= 0 | a . QM a | QM a * QM a
         | { MPtn a : (MPtn <- Ma)*, (MSat a)* }
MPtn a ::= M (LitOrVar a)
MSat a ::= MPtn a in MCnd a
MCnd a ::= true | ~ MCnd a | MCnd a or MCnd a
           0 | a . MCnd a | MCnd a * MCnd a
```

Data type + Collection Type

Data type w/ built in indexing

+

Pattern language

+

Logic

Summary

- A collection is anything we can write with a comprehension
  - Effectively given by a monad C
- An item is anything we can represent with EBNF
  - Effectively given by a monad G
- An analytic is a machine for mashing things up
  - Effectively given by a distributive law D : CG -> GC

Summary

- D shows how to express collections of terms as terms of collections
- The existence of such distributive law for reasonable notions of collections and items is the generalization of Codd's theorem

A machine for mashing things up

- Capability set is strictly richer than what can be said with LINQ
  - Menu example already causes exponential blow-up for MS LINQ
  - "Q" procedure extends to *behavioral* predicates these are not expressible in LINQ
- Efficient approach compiles this to a *lazy* query plan

A machine for mashing things up

- (QM a) semantics is independent of the means by which the constraints are solved
- Oleg Kiselyov's LogicT monad transformer is a good example -- he shows that for a certain 'notion of collection' one can give a 'natural' backtracking solution
- Nothing prevents one from using a special purpose solver
    - This is one of the many reasons why having a specification of the logic that is *as tight as possible* is a good thing -- the solver can often be chosen from the constraint language

A path to this goal

- (Behavioral and structural queries in physical systems) Find all the pathways, p, in <SignallingPathwayDB> such that when <Reagent> is added in concentration K p eventually reaches states in which we see these <Reagent1>,...,<ReagentN> in concentrations K1,...,KN.
    - Following Priami/Regev/Shapiro, et al, encode biological processes as processes in Milner's π-calculus
    - Term language monad -- Milner's π-calculus
    - Collection monad -- Nominal sets
    - Distributive law -- semantics of Caires' logic of spatial and behavioral observations
    - SELECT process FROM xmldb WHERE formula-in-logic-of-spatial-and-behavioral observations

A path to this goal

- (Structural queries in physical systems) Find all the knots in <KnotDB> that have alternations of <Trefoil> and any 7-crossing knot as subknots
  - Following Meredith and Snyder knots as processes in Milner's π-calculus
  - Term language monad -- Milner's π-calculus
    - Alternate term language -- Conway's knotation
  - Collection monad -- Nominal sets
  - Distributive law -- semantics of Caires' logic of spatial and behavioral observations
  - SELECT process FROM xmldb WHERE formula-in-logic-of-spatial-and-behavioral observations

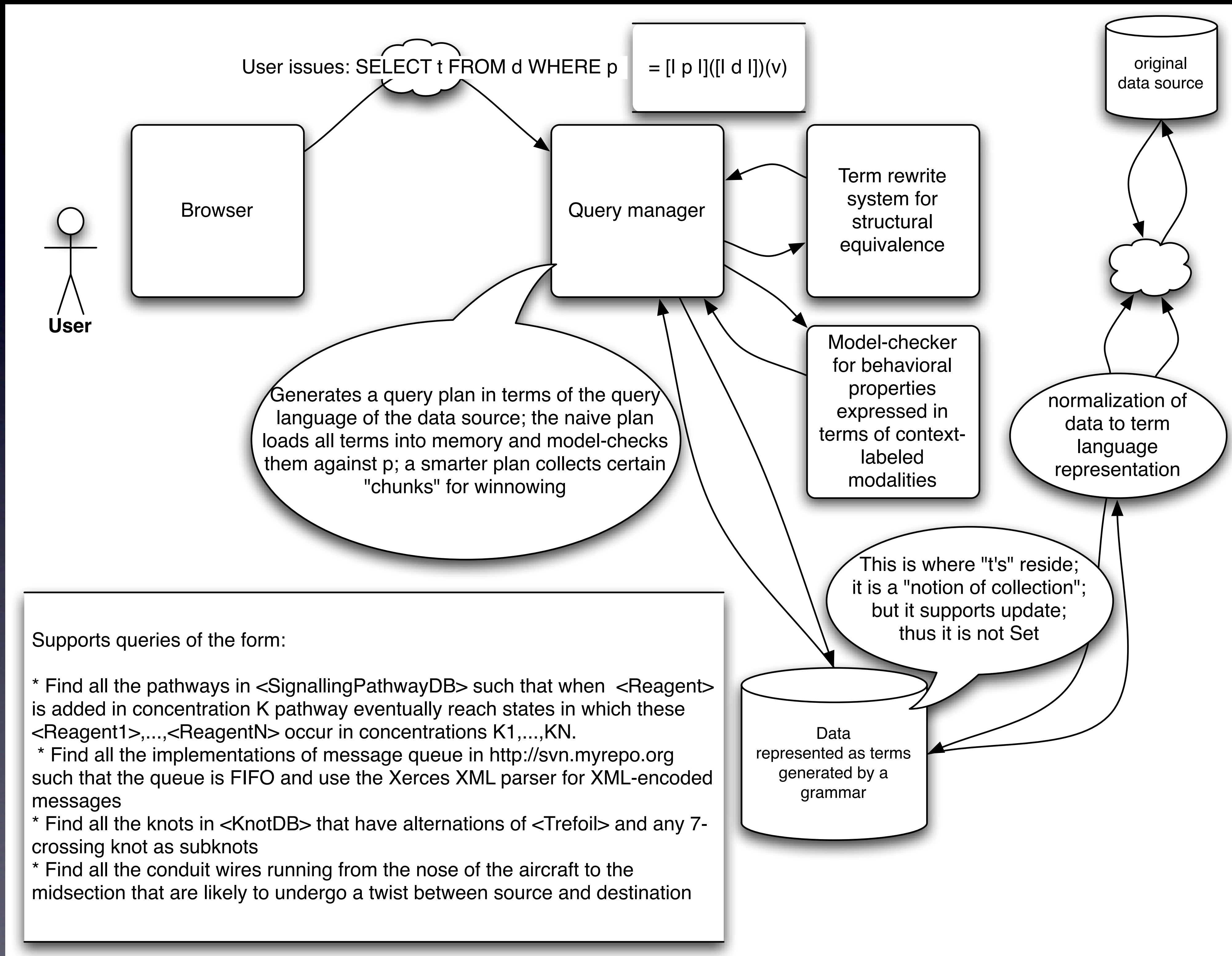
A path to this goal

- (Behavioral and structural queries in logical systems) Find all the implementations of message queue in http://svn.myrepo.org such that the queue is FIFO and use the Xerces XML parser for XML-encoded messages
    - Following Berger, Honda and Yoshida encode Java programs as strategies in games
    - Term language monad -- Games
    - Collection monad -- Mutable sets
    - Distributive law -- semantics of Berger, Honda, Yoshida Hoare-logic
    - SELECT strategy FROM xmldb WHERE formula-in-BHY-logic

A path to this goal

- (Geometric queries in physical systems) Find all the conduit wires running from the nose of the aircraft to the midsection that are likely to undergo a twist between source and destination
    - Encode geometric information in Clifford algebra
    - Term language monad -- Clifford algebra
    - Collection monad -- Mutable sets
    - Distributive law -- exhibited by Meredith and Beckman
    - SELECT ensemble FROM xmldb WHERE formula-in-Geometric-logic

Architecture



References

- Comprehending monads, Phil Wadler
- Backtracking, interleaving and terminating monad transformers, Oleg Kiselyov, Chung-chie Shan, Daniel P Friedman
- Behavioral and spatial observations in a logic for the π-calculus, Luis Caires
- Domain theory in logical form, Samson Abramsky

Haskell/Scala cheat sheet

Haskell	Scala

`do var1 <- e1`
`   var2 <- e2`
`   e`
	`for( var1 <- e1;`
	`     var2 <- e2;`
	`     rslt <- e )`
	`yield rslt`

`do var1 <- e1`
`   var2 <- e2`
`   return e`
	`for( var1 <- e1;`
	`     var2 <- e2 )`
	`yield e`

`do var1 <- e1 >> e2`
`   return e`
	`for( var1 <- e1;`
	`     var2 <- e2 )`
	`yield e`

Haskell/Scala cheat sheet

Haskell	Scala

`do {e;stmts} =`
`  e >> do {stmts}`
`do {p <- e; stmts} =`
`  let ok p = do {stmts}`
`      ok _ = fail "..."`
`    in e >>= ok`
`do {let decls; stmts} =`
`  let decls in do {stmts}`
	`for( var <- e )yield rslt =`
	`  e map { x => rslt }`
	
	`for( var <- e if c ) yield rslt =`
	`  e filter { x => c } map { x => rslt }`
	
	`for ( v1 <- e1; v2 <- e2 ) yield rslt =`
	`  e1 flatMap { x => e2 map { x => rslt } }`

Haskell/Scala cheat sheet

Haskell	Scala

`class Monad m where`
`  -- chain`
`  (>>=) :: m a -> (a -> m b) -> m b`
`  -- inject`
`  return :: a -> m a`
	`trait Monad {`
	`  type T`
	`  def map(f: Monad => T): T`
	`  def flatMap(f: Monad => T): T`
	`  def filter(f: Monad => Boolean): Monad`
	`}`

Category theory
$M : C \to C$ $unit : Id \to M$ $mult : M^2 \to M$

Questions?