Computation Abstraction

Going beyond programming language glue, or what we've missed from FP for so long in mainstream

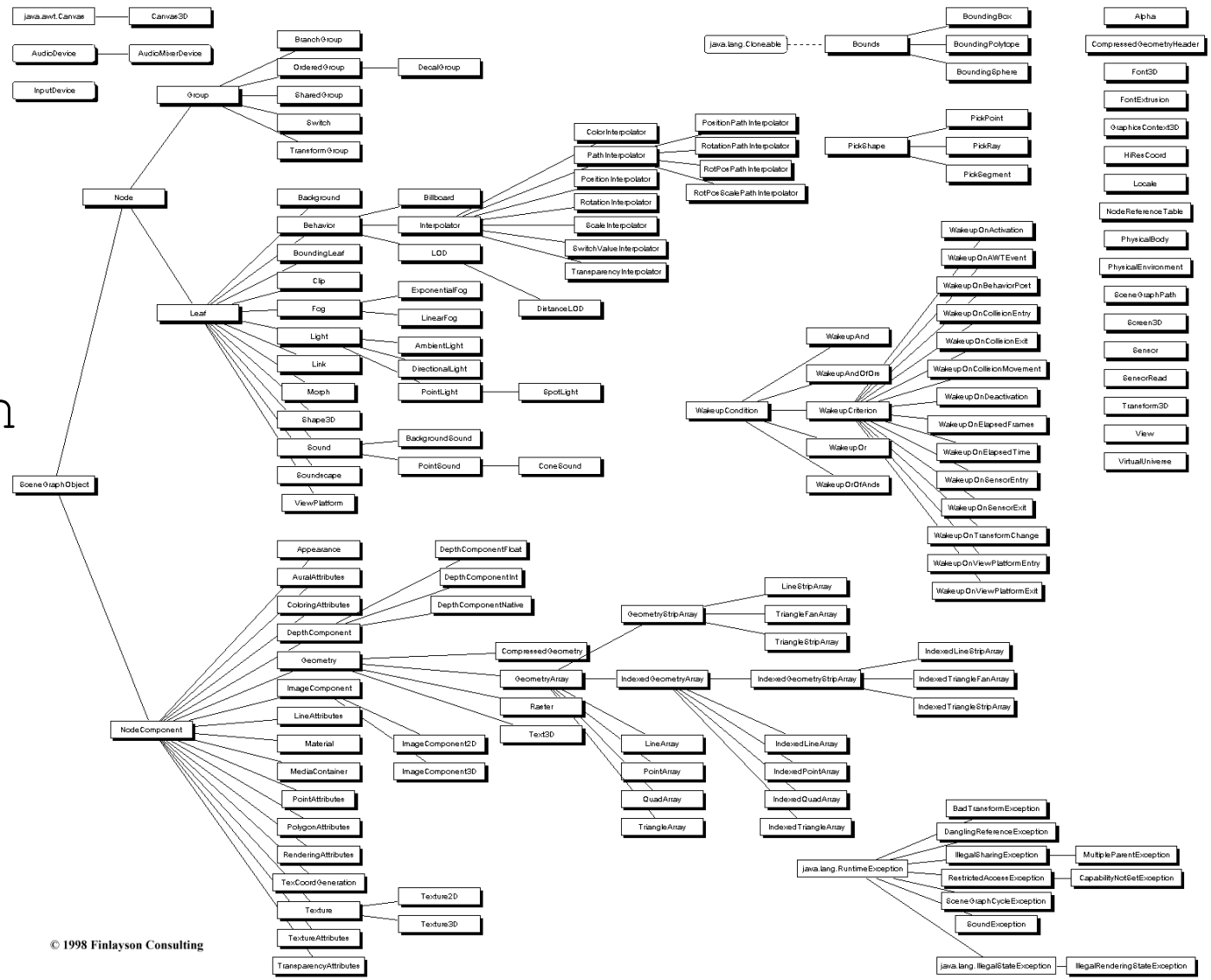
googlewave.com!w+PgcakhgiA
sadache@twitter

Sadek Drobi
Consultant (j2EE,.Net)
Programming Languages Hobbyist
Training: FP (F#, Scala), WOA
Architecture Editor at InfoQ.com
Maybe known for my FP interviews?
http://www.infoq.com/bycategory/co

WWW.SadekDrobi.com
sadache@twitter

What has been abstraction for us in mainstream?

Hierarchical and Structural ~~O~~Abstr~~u~~action



What do we know about Computation Abstraction in mainstream?

This page is intentionally left blank.

What do we know about Computation Abstraction in mainstream?

Did anyone mention Behavioral GOF Design Patterns?

What is Computation Abstraction?

googlewave.com!w+PgcakhgiA
sadache@twitter

It is all about Glue!

googlewave.com!w+PgcakhgiA
sadache@twitter

```
IDictionary<string,int> zipCodes= new Dictionary<string,int>{
            {"Paris",75}
        };
IDictionary<int, int> population = new Dictionary<int, int>{
            {75,100}
        };

int GetInterstingNumber(string cityName){
        var myCityCode= zipCodes[cityName];
       return ( population[myCityCode] *100 ) / TOTAL_POPULATION ;
}

void PrintIt(string[] args){
      Console.WriteLine("Paris has "+getInterstingNumber("Paris")+
                                "% of Population");
}
```

```
IDictionary<string,int> zipCodes= new Dictionary<string,int>{
            {"Paris",75}
        };
IDictionary<int, int> population = new Dictionary<int, int>{
            {75,100}
        };

int GetInterstingNumber(string cityName){
        var myCityCode= zipCodes[cityName];
       return ( population[myCityCode] * 100) / TOTAL_POPULATION ;
}

void PrintIt(string[] args){
      Console.WriteLine("Nancy has "+getInterstingNumber("Nancy")+
                                "% of Population");
}
```

googlewave.com!w+PgcakhgiA

sadache@twitter

Welcome in the REAL WORLD

googlewave.com!w+PgcakhgiA
sadache@twitter

```
IDictionary<string,int> zipCodes= new Dictionary<string,int>{
        {"Paris",75}
    };
IDictionary<int, int> population = new Dictionary<int, int>{
        {75,100}
    };

int GetInterstingNumber(string cityName){
    var myCityCode= zipCodes[cityName];
    return ( population[myCityCode] * 100 ) / TOTAL_POPULATION ;
}

void PrintIt(string[] args){
    Console.WriteLine("Nancy has "+getInterstingNumber("Nancy")+
                      "% of Populatin");
}
```

googlewave.com!w+PgcakhgiA

sadache@twitter

```
static int? GetInterstingNumber(string cityName){

    int? myCityCode=null;
    try
      {
        myCityCode = zipCodes[cityName];
      }
    catch(KeyNotFoundException e)
      {
        myCityCode = null;
      }
    try
      {
        return (population[myCityCode.Value] * 100 / TOTAL_POPULATION);
      }
    catch (KeyNotFoundException e){ return null;}

    catch(/* .Value can produce an*/ InvalidOperationException e)
          {
              return null;
          }
}
```

googlewave.com!w+PgcakhgiA

sadache@twitter

```
static int? GetInterstingNumber(string cityName){

    int? myCityCode=null;
    try
      {
        myCityCode = zipCodes[cityName];
      }
    catch(KeyNotFoundException e)
      {
        myCityCode = null;
      }
    try
      {
        return (population[myCityCode.Value] * 100 / TOTAL_POPULATION);
      }
    catch (KeyNotFoundException e){ return null;}

    catch(/* .Value can produce an*/ InvalidOperationException e)
          {
              return null;
          }
}
```

googlewave.com!w+PgcakhgiA

sadache@twitter

How does this default glue look like?

googlewave.com!w+PgcakhgiA
sadache@twitter

How does this only possible glue look like?

- Errors are represented through an Exception System that by default cascades them up the call stack
- Exceptions short circuit (interrupt execution until they are "catched")
- Nulls by default produce errors that cascade up as exceptions
- These defaults can't be overridden but can be interrupted using some language syntax

What's wrong with our only possible glue?

googlewave.com!w+PgcakhgiA
sadache@twitter

Mainly two types of problems:
one **P**RACTICAL and
one **C**onceptual

googlewave.com!w+PgcakhgiA
sadache@twitter

Practical Problem:

- **Noise that disperses the main algorithm** declaration making readability a challenge

```
static int? GetInterstingNumber(string cityName){

    const int TOTAL_POPULATION=123;
    int? myCityCode=null;
    try
     {
        myCityCode = zipCodes[cityName];
     }
    catch(KeyNotFoundException e)
     {
        myCityCode = null;
     }
    try
     {
        return (population[myCityCode.Value] * 100 / TOTAL_POPULATION);
     }
    catch (KeyNotFoundException e){ return null;}

    catch(/* .Value can produce an*/ InvalidOperationException e)
          {
             return null;
          }
}
```

The special case handling is duplicated in two different places

googlewave.com!w+PgcakhgiA
sadache@twitter

Conceptual Problem:

```
static int? GetInterstingNumber(string cityName){

    const int TOTAL_POPULATION=123;
    int? myCityCode=null;
    try
     {
        myCityCode = zipCodes[cityName];
     }
    catch(KeyNotFoundException e)
     {
        myCityCode = null;
     }
    try
     {
        return (population[myCityCode.Value] * 100 / TOTAL_POPULATION);
     }
    catch (KeyNotFoundException e){ return null;}

    catch(/* .Value can produce an*/ InvalidOperationException e)
          {
             return null;
          }
}
```

I can't abstract it and say for instance: *for any null you encounter in the algorithm stop and return null as a final answer.*

… and since I can't abstract the glue logic **I can't reuse it** in another algorithm definition across my application/**domain**.

googlewave.com!w+PgcakhgiA
sadache@twitter

Some languages continue kindly trying
add more ways to approach these issues
but they still share the same problems
with the main glue

"Nancy has" + getInterstingNumber("Nancy")??"NA"+"% of Population"

googlewave.com!w+PgcakhgiA
sadache@twitter

So how do we abstract “glue” or “computations”?

or as some call it “overriding the semicolon ;”

googlewave.com!w+PgcakhgiA

sadache@twitter

We need computation abstraction tools!

Functional Languages contain naturally our needed tools (functions, functors, monads,…)

The good news is that some currently mainstream (C#) and other potentially mainstream (Scala, F#) programming languages have them in someway too!

Abstracting the glue with nulls (None) in F# would be:

```
let getInterestingNumber
   (cities:Map<string,int>) (population:Map<int,int>)
   (cityName:string) :int Option=
     maybe{ let! zipCode= cities.TryFind cityName
            let! cityPopulation= population.TryFind zipCode
            return cityPopulation * 100 / TOTAL_POPULATION }
```

googlewave.com!w+PgcakhgiA
sadache@twitter

The glue implementation with null (None) propagation looks like:

```
module Maybe=
    let succeed x = Some(x)
    let fail = None
    let bind p rest =
        match p with
            | None -> fail
            | Some r -> rest r
    let delay f = f()

    type MaybeBuilder() =
        member b.Return(x) = succeed x
        member b.Bind(p, rest) = bind p rest
        member b.Delay(f) = delay f
        member b.Let(p,rest) = rest p

    let maybe = MaybeBuilder()
```

googlewave.com!w+PgcakhgiA
sadache@twitter

What about a glue implementation that collects errors and goes on?

```
let plusOneWithError ints= (Error "First Error",List.map ((+)1) ints)
let plus2WithAnotherError ints=(Error "2nd Error",List.map ((+)2) ints)
let twiceWithNoError ints= (NoError,List.map (( *)2) ints )

let answer= collectingErrors { let! l1= plusOneWithError [1;2;3]
                               let! l2= plusTwoWithAnotherError l1
                               let! l3= twiceWithNoError l2
                               return l3 }
```

```
val final : Error * int list =
  (ErrorList [Error "First Error"; Error "Second Error"], [8; 10; 12])
```

googlewave.com!w+PgcakhgiA
sadache@twitter

In Scala you can Abstract Computation too:

```
for{ i <- Some(1)
     val j = i +1 } yield j )
```

Evaluates to Some(2)

```
def map[B](f: A => B): Option[B]=
                o match{case None => None
                                case Some(a) => Some f(a)}


def flatMap[B](f: A => Option[B]): Option[B]=
                          o match{case None => None
                                        case Some(a) => f(a)}
```

What about C#, a current mainstream language?

Heard of LinQ of course!

```
IEnumerable<double> result=from i in Enumerable.Range(0,100)
                           select 1.0/i;
```

there is an error of division by zero there!

```
IEnumerable<double> result=from i in Enumerable.Range(0,100)
   .IgnoringErrors()
                           select 1.0/i;
```

Non problem. Since we abstracted computation, we can use a more tolerant implementation of the glue!

googlewave.com!w+PgcakhgiA
sadache@twitter

What about C#, a current mainstream language?

Implementing asynchronous programming glue using Computation Abstraction in the Reactive Programming Framework:

```
// Create mouse drag
 var mouseDrag = from md in this.GetMouseDown()
                 from mm in this.GetMouseMove()
                                    .Until(this.GetMouseUp())
                 select mm;
// Subscribe
 var handler = mouseDrag.Subscribe( e =>

  PublishMouseDrag(e.EventArgs.Location));
```

googlewave.com!w+PgcakhgiA
sadache@twitter

In Scala Error propagating glue without exceptions:

```
def throwError(i:Int):ThrowsError[Int]=
                        Error("I don't like " +i+ "!");

for{i <- throwError(2)
    val j = i + 1}  yield j )
```

Evaluates to Error(I don't like 2!)

googlewave.com!w+PgcakhgiA
sadache@twitter

```
case class ThrowsError[A] (e:Either[Error,A]){

        def map[B](f: A => B): ThrowsError[B]= ThrowsError(e.right.map(f))
        def flatMap[B](f: A => ThrowsError[B]): ThrowsError[B]=
                                        ThrowsError(e.right.flatMap(a=>f(a).e))
  }
```

Asynchronous Worflows in F#

```
let asynctask =
     async { let req = WebRequest.Create(url)
             let! response = req.GetResponseAsync()
             let stream =
response.GetResponseStream()              let
streamreader =
               new System.IO.StreamReader(stream)
                  return streamreader.ReadToEnd() }
```

googlewave.com!w+PgcakhgiA
sadache@twitter

What did I show so far?

Defined glue for:

- Nulls
- Errors that propagate (better exceptions)
- Custom glue that doesn’t short circuit (collectErrors)
- Events and asynchronous programming
- A lot is already available (Lists, Streams, Channels, State, Enviorement…)
- Easily add your own (implementing map, bind(flatMap))
- Combine them!

googlewave.com!w+PgcakhgiA
sadache@twitter

Reuse and combine

Monad Transformers

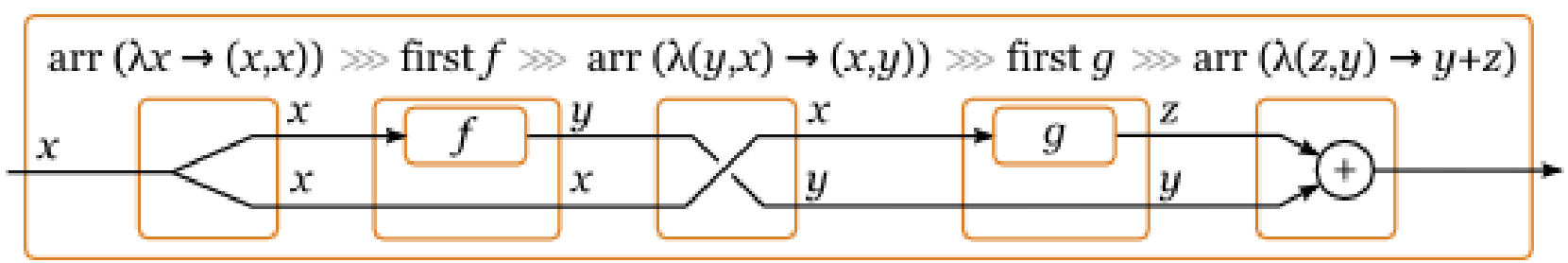
googlewave.com!w+PgcakhgiA
sadache@twitter

Showed Computation Abstraction

Functors, Applicative Functors, Monads, Comands

Still there are others!

Like Arrows for more control over your computation:



googlewave.com!w+PgcakhgiA

sadache@twitter

Release Your Algorithms from plumping code

Let them Express themselves!

Q?

googlewave.com!w+PgcakhgiA
sadache@twitter