Porting BETA to ROTOR

ROTOR Projects Presentation Day, June 16 2004

by Peter Andersen

The BETA programming language

- Object-oriented programming language
  - Scandinavian school of OO, starting with the Simula languages
  - Simple example:

```
Calculator:
 (# R: @integer;
    set:
     (# V: @integer enter V do V → R #);
    add:
     (# V: @integer enter V do R+V → R exit R #);
 #);
```

A *pattern* named `Calculator`

Static instance variable named `R`

Internal *pattern* named `set` with an input variable `V`

Internal *pattern* named `add` with an input variable `V` and a return value named `R`

BETA example use

```
Calculator:
 (# R: @integer;
    set:
     (# V: @integer enter V do V → R #);
    add:
     (# V: @integer enter V do R+V → R exit R #);
 #);
```

Use of `add` as a method:

```
C: @Calculator;
X: @integer;
5 → C.add → X
```

Use of `add` as a class:

```
C: @Calculator;
X: @integer;
A: ^C.add;
&C.add[] → A[];
5 → A → X
```

Creation of an instance of C.add

Execution of the C.add instance

BETA vs. CLR/CLS

- Class and method unified in *pattern*
- General nesting of patterns, i.e. also of methods
  - Uplevel access to fields of outer patterns
- INNER instead of super
- Enter-Do-Exit semantics
- Genericity in the form of virtual patterns
- Multiple return values
- Active objects in the form of Coroutines
- No constructors, no overloading
- No dynamic exceptions

BETA.Net/Rotor Challenges

- Mapping must be complete and semantically correct
- BETA should be able to uses classes from other languages
- Other languages should be able to use classes generated from BETA source code
- BETA should be able to inherit classes from other languages
- Other languages should be able to inherit from BETA
- The BETA mapping should be ’nice’ when seen from other languages
- In .NET terminology:
  - BETA compliant with Common Language Specification (CLS)
  - BETA should be a *CLS Extender*

The mapping

- Generating bytecode for CLR mostly corresponds to making a BETA source mapping into C# source code
- C# used here for presentation purpose
- But we do generate IL (intermediate language bytecodes) directly into IL files
- IL files assembled with **`ilasm`**

Mapping patterns: nested classes

```
public class Calculator: System.Object {
  public int R;
  public class add: System.Object {
    public int V;
    Calculator origin;
    public add(Calculator outer) { origin = outer; }
    public void Enter(int a) { V = a; }
    public void Do() { origin.R = origin.R + V; }
    public int Exit() { return origin.R; }
  }
  public int call_add(int V){
    add A = new add(this);
    A.Enter(V);
    A.Do();
    return A.Exit();
  }
  …
}
```

CLS does not allow for this to be called just add()

```
Calculator:
 (# R: @integer;
    …
    add:
     (# V: @integer
     enter V
     do R+V → R
     exit R
     #);
  #);
```

Use of `add` as a class:

```
C: @Calculator;

X: @integer;
A: ^C.add;
&C.add[] → A[];
5 → A → X
```

```
Calculator C
   = new Calculator()
int X;
Calculator.add A;
A = new Calculator.add(C);
A.Enter(5);
A.Do()
X = A.Exit();
```

Use of `add` as a method

```
C: @Calculator;

X: @integer;
5 → C.add → X
```

```
Calculator C
   = new Calculator()
int X;
X = C.call_add(5);
```

Interface to external classes etc.

- Pt. in a declarative manner
- Ex:

```
String: ExternalClass
  (# _init_ArrayOfChar: cons (* constructor *)
       (# result: ^String; arg1: [0]@char;
       enter (arg1[]) exit result[]
       #);
     ...
     CompareTo_Object: proc (* overloaded CompareTo *)
       (# result: @int32; arg1: ^Object;
       enter (arg1[]) do 'CompareTo' -> procname; exit result
       #);
     ...
  do '[mscorlib]System.String' -> className;
  #);
```

- Rudimentary support for overloading, constructors etc.
- Offline batch tool `dotnet2beta` implemented using reflection (generates BETA source files); should be part of BETA compiler
- System.String vs. BETA text: Automatic coercion

Not described here…

- **Virtual classes** – corresponding to generics (.NET 2.0 – "Whidbey") – implemented with virtual instantiation methods and a lot of (unnecessary) casting.
- **Coroutines** and **concurrency** – implemented with threads. More on this later…
- **Pattern variables**: Classes and methods as first-class values – implemented with reflection
- **Leave/restart** out of nested method activations – implemented with exceptions
- **Multiple return values** – implemented with extra fields
- Numerous minor details!

Status

- 95% of BETA language implemented
  - Leave/restart across component border missing
  - Coroutines and leave/restart not ideally implemented
- Some things need to be more ‘nice’
- Not yet 100% CLS compliant
  - E.g. custom attributes and consumption of value types
- Optimizations needed
  - Large number of classes generated due to the generality of BETA

Major missing details

- Value types consumption
- Enumeration consumption
- Throw and handling of CLR exceptions
- Consumption of static fields
- Support for custom attributes
  - Maybe proc, class etc. as attributes?
- Leave/restart over coroutine border
- Support for multiple interfaces
- 64 bit arithmetic
- Boot-strapped compiler (needs some of above)
- Implementation of BETA class libraries
- Direct compiler support for reading external classes
- Visual Studio .NET language extension

Plans for ROTOR

1. ✔ Simple hello-world and complete compiler test suite
   - Hello-world and most of compiler test suite up-and-running; `clix` script generation added
2. Implement (some of) above mentioned missing details
3. Bootstrapping the BETA compiler to ROTOR and .NET
   - Currently ongoing
4. Develop a GUI framework on top of ROTOR and .NET.
   - System.Windows.Forms and System.Drawing not available on ROTOR
   - BETA traditional GUI library *Lidskjalv* and new OpenGL based GUI library *Octopus* considered
5. Investigate support for Simula/BETA-style coroutines
   - Modify ROTOR bytecodes/jitter/GC/class libraries?

Coroutines in C#

- Imagine:

```
abstract class Coroutine {  // Similar to Thread
  ...
  public void call() { ... }
  protected void suspend() { ... }
  abstract void Do();   // Similar to Run()
}
SpecificCoroutine: Coroutine{ … }
Coroutine S = new SpecificCoroutine();
```

- Do() is action part of coroutine
- S.call() will invoke Do()
- suspend() in S.Do() (or methods called from S.Do()) will return to the point of S.call() and resume execution after S.call()

Example: Adder

- Produces sequence start + start, (start+1)+(start+1) …
- By using (infinite) recursion
- Suspends after each computation

```
class Adder: Coroutine {
    public int res;
    int start;
    public Adder(int s) {
      start = s;
    }
    void compute(int V){
            res = V+V;
            suspend();
            compute(V+1);
    }
    public override void Do() {
            compute(start);
    }
}
```

Example: Multiplier

- Produces sequence start * start, (start+1) * (start+1) …
- By using (infinite) recursion
- Suspends after each computation

```
class Multiplier: Coroutine {
    public int res;
    int start;
    public Multiplier(int s) {
      start = s;
    }
    void compute(int V){
            res = V*V;
            suspend();
            compute(V+1);
    }
    public override void Do() {
            compute(start);
    }
}
```

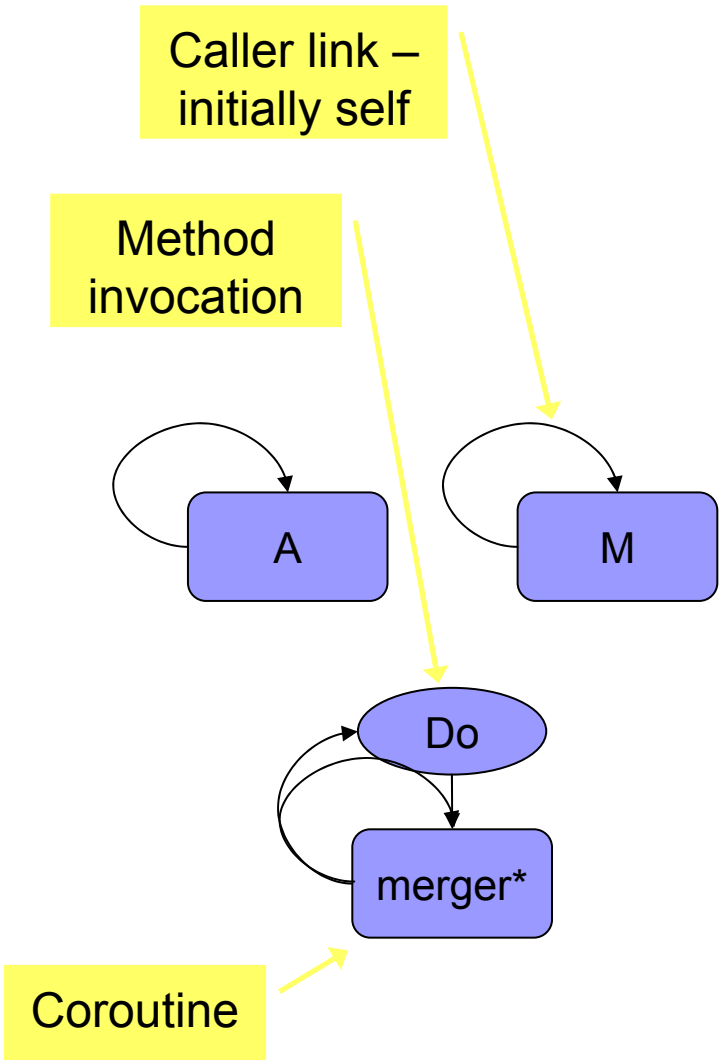
Merger

- Merge sequences produced by Adder instance and Multiplier instance
- Sort in ascending order
- First 6 values

```
class Merger: Coroutine {
    Adder A = new Adder(3);
    Multiplier M = new Multiplier(2);
    public override void Do() {
       A.call(); M.call();
       for (int i=0; i<6; i++){
          if (A.res < M.res) {
             Console.WriteLine("A: " + A.res);
             A.call();
          } else {
             Console.WriteLine("M: " + M.res);
             M.call();
          }
       }
    }
    public static void Main(String[] args) {
        (new Merger()).call()
    }
}
```

Adder | Multiplier | Merger

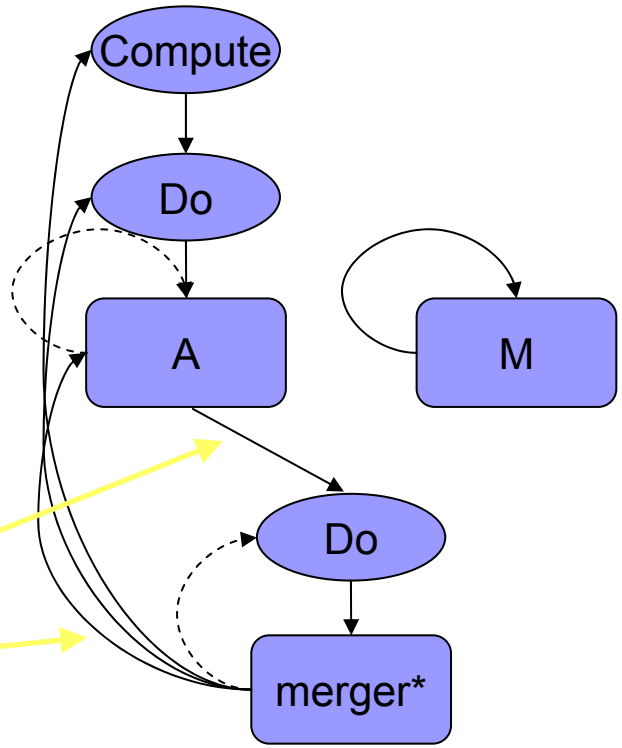
```
class Merger: Coroutine {
    Adder A = new Adder(3);
    Multiplier M = new Multiplier(2);
    public override void Do() {
      A.call(); M.call();
        for (int i=0; i<6; i++){
            if (A.res < M.res) {
                Console.WriteLine("A: " + A.res);
                A.call();
            } else {
                Console.WriteLine("M: " + M.res);
                M.call();
            }
        }
    }
    public static void Main(String[] args) {
      (new Merger()).call()
    }
}
```



Adder | Multiplier | Merger

```
class Adder: Coroutine {
    public int res;
    int start;
    public Adder(int s) {
      start = s;
    }
    void compute(int V){
        → res = V+V;
        → suspend();
          compute(V+1);
    }
→   public override void Do() {
        → compute(start);
    }
}
```

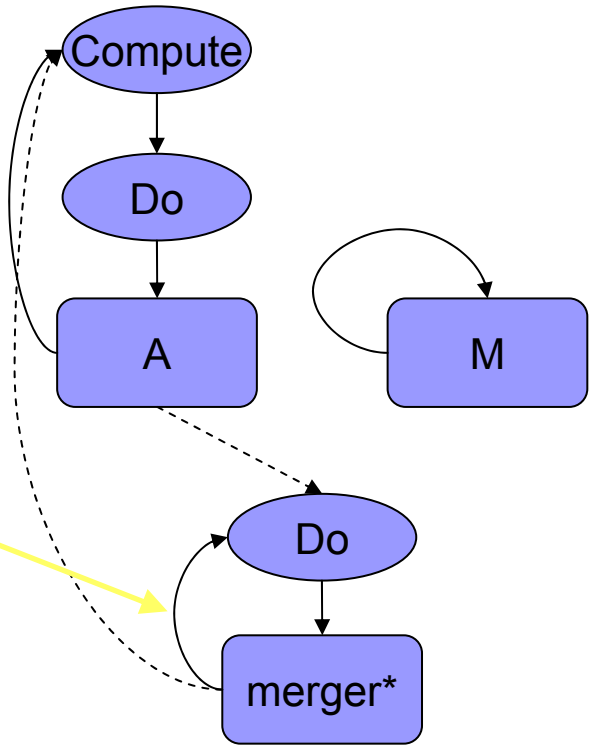
Call() is basically just a swap of two pointers



Adder | Multiplier | Merger

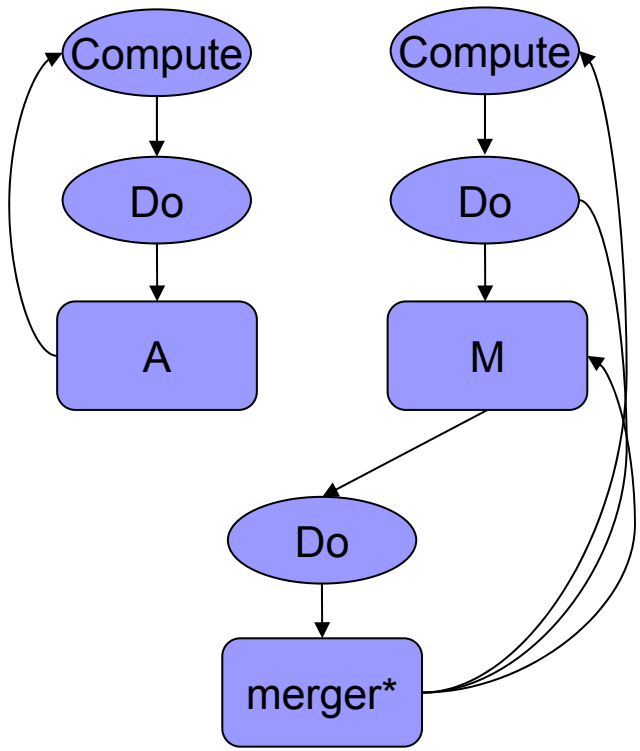
```
class Merger: Coroutine {
    Adder A = new Adder(3);
    Multiplier M = new Multiplier(2);
    public override void Do() {
       A.call(); → M.call();
       for (int i=0; i<6; i++){
          if (A.res < M.res) {
             Console.WriteLine("A: " + A.res);
             A.call();
          } else {
             Console.WriteLine("M: " + M.res);
             M.call();
          }
       }
    }
    public static void Main(String[] args) {
        (new Merger()).call()
    }
}
```

suspend() is also basically just a swap of two pointers



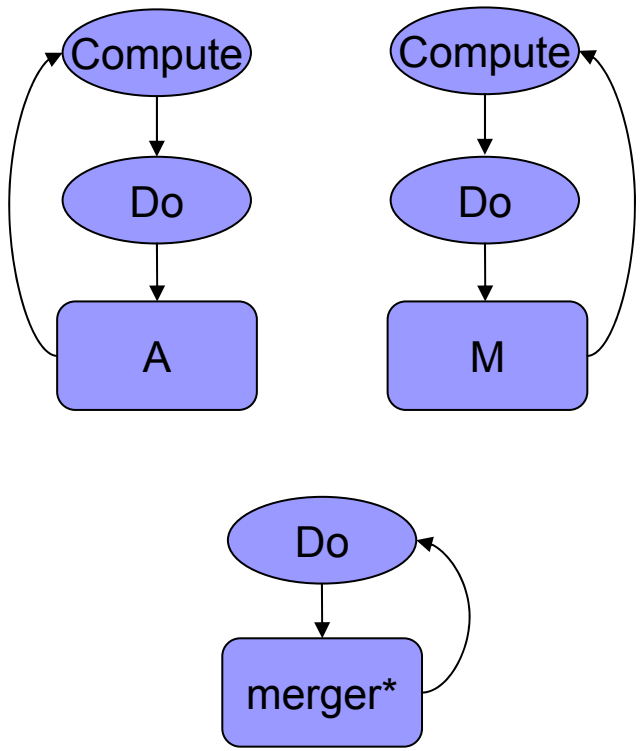
Adder | Multiplier | Merger

```
class Multiplier: Coroutine {
    public int res;
    int start;
    public Multiplier(int s) {
      start = s;
    }
    void compute(int V){
        → res = V*V;
        →suspend();
          compute(V+1);
    }
→public override void Do() {
        → compute(start);
    }
}
```



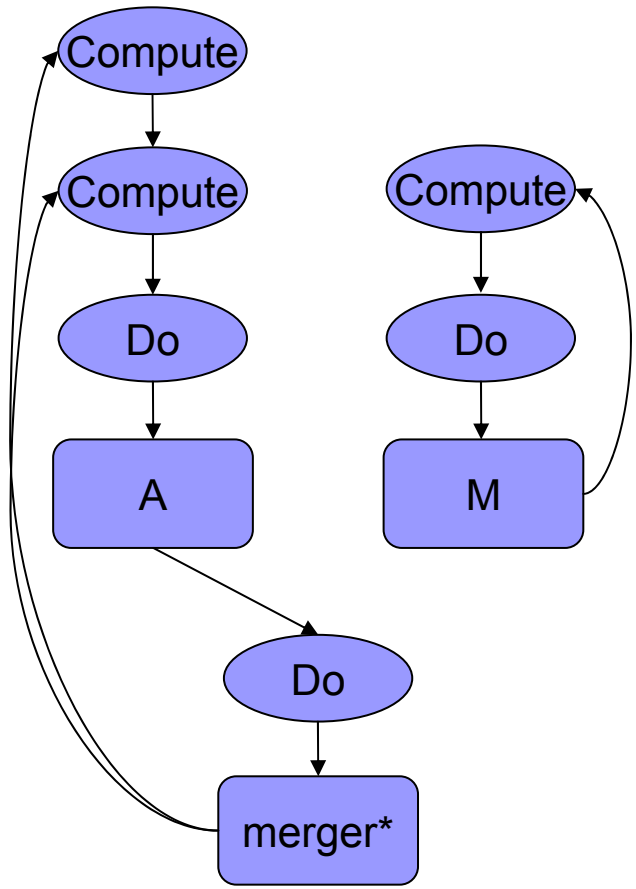
Adder | Multiplier | Merger

```
class Merger: Coroutine {
    Adder A = new Adder(3);
    Multiplier M = new Multiplier(2);
    public override void Do() {
       A.call(); M.call();
    for (int i=0; i<6; i++){
      if (A.res < M.res) {
        Console.WriteLine("A: " + A.res);
        A.call();
          } else {
             Console.WriteLine("M: " + M.res);
             M.call();
          }
       }
    }
    public static void Main(String[] args) {
        (new Merger()).call()
    }
}
```



Adder | Multiplier | Merger

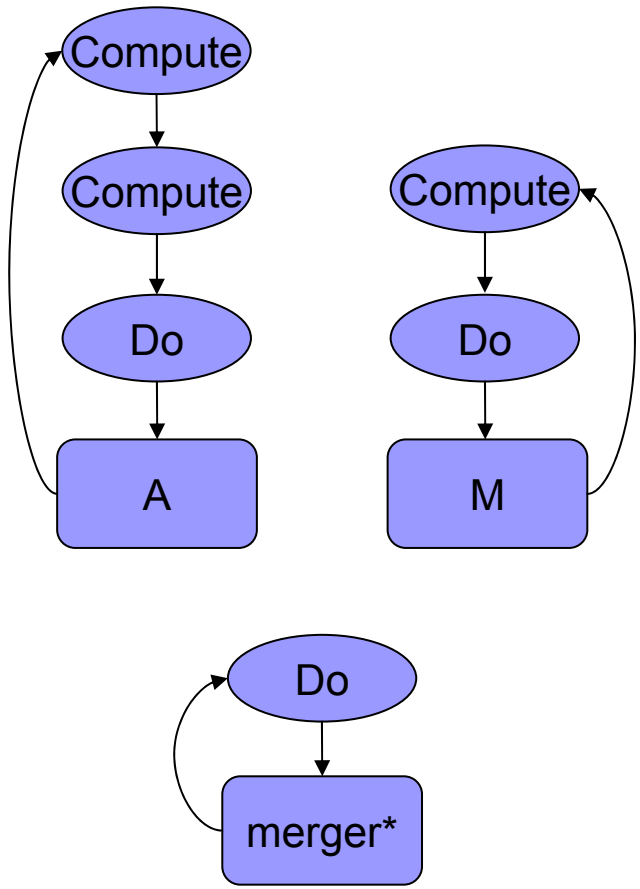
```
class Adder: Coroutine {
    public int res;
    int start;
    public Adder(int s) {
      start = s;
    }
    void compute(int V){
        → res = V+V;
        → suspend();
        → compute(V+1);
    }
    public override void Do() {
            compute(start);
    }
}
```



Adder | Multiplier | Merger

```
class Merger: Coroutine {
    Adder A = new Adder(3);
    Multiplier M = new Multiplier(2);
    public override void Do() {
       A.call(); M.call();
       for (int i=0; i<6; i++){
          if (A.res < M.res) {
             Console.WriteLine("A: " + A.res);
             A.call();
          } else {
             Console.WriteLine("M: " + M.res);
             M.call();
          }
       }
    }
    public static void Main(String[] args) {
       (new Merger()).call()
    }
}
```

… and so on



Implementation of class Coroutine

- **`class Coroutine`** implemented by means of another **`class Component`**:

```
public abstract class Coroutine {
    internal Component  _comp;
    public Coroutine(){
       _comp = new Component(this);
    }
    public void call() {  _comp.swap();  }
    protected void suspend() {  _comp.swap();  }
    public abstract void Do();
}
```

call() and suspend() implemented using a single swap method

Implementation of class Component

- **`class Component`** implemented by means of **`System.Threading.Thread`** and **`System.Threading.Monitor`**

```
public class Component {
    public static Component current;
    private Component caller; // == this when suspended
    private Coroutine body;
    private System.Threading.Thread myThread; // notice private
    public Component(Coroutine b)
        { ... Constructor: allocate myThread starting in run; set up caller etc. }
    private void run()
        { ... Thread entry point: call body.Do() and then terminate myThread ... }
    public void swap()
        { ... Main call() / suspend() handling; next slide ... }
}
```

Implementation of `Component.swap()`

- Used asymmetrically:
  - Call: this == callee; this.caller == this
  - Suspend: this == current; this.caller to be resumed

```
public void swap()
{
    lock (this){
        Component old_current = current;
        current = caller;
        caller = old_current;
        if (!myThread.IsAlive) {
            myThread.Start();
        } else {
            System.Threading.Monitor.Pulse(this);
        }
        System.Threading.Monitor.Wait(this);
    }
}
```

Currently executing Component/Coroutine

Swap pointers

Start or resume new current

Suspend old current

Implementation of Components in BETA.Net

- Any pattern in BETA may be used as a coroutine
- Implemented as shown for C#, with **`class BetaObject`** used instead of **`class Coroutine`**
- **`class BetaObject`** is common superclass for all BETA objects
- Same **`class Component`** used

Comparison with C# 2.0 `yield`

- Coming C# 2.0 has new feature called yield return
- Used for implementing enumerator pattern
- May be considered ”poor mans coroutine”
- Can only ”save” one stack frame

Iterators

(slide borrowed from Erik Meijer)

- Method that incremer
  returns a sequence o
  - yield return and yield
  - Must return IEnumera

```
public class List
{
   public IEnumerator GetE
      for (int i = 0; i < count; i++) {
         yield return elements[i];
      }
   }
}
```

```
public IEnumerator GetEnumerator() {
   return new __Enumerator(this);
}

private class __Enumerator: IEnumerator
{
   object current;
   int state;

   public bool MoveNext() {
      switch (state) {
         case 0: …
         case 1: …
         case 2: …
         …
      }
   }

   public object Current {
      get { return current; }
   }
}
```

Coroutine support in .NET/ROTOR?

- Thread synchronization seems clumsy and inefficient (benchmarking pending, though)
- Direct user defined scheduling desirable
  - P/Invoke of WIN32 Fibers?
  - ROTOR extension with e.g. **`coswap`** bytecode?
    - Addition of bytecode straight-forward
    - Managing of thread stacks?
      - No idea how hard that would be in ROTOR
  - ➔Our coming research!!

Contacts:

Questions?

- Peter Andersen (that's me)

  mailto:datpete@daimi.au.dk
- Ole Lehrmann Madsen

  mailto:olm@daimi.au.dk
- Download:

  http://www.daimi.au.dk/~beta/ooli