

Structured Synchronous Reactive Programming with Céu



LabLua – PUC-Rio
www.lua.inf.puc-rio.br

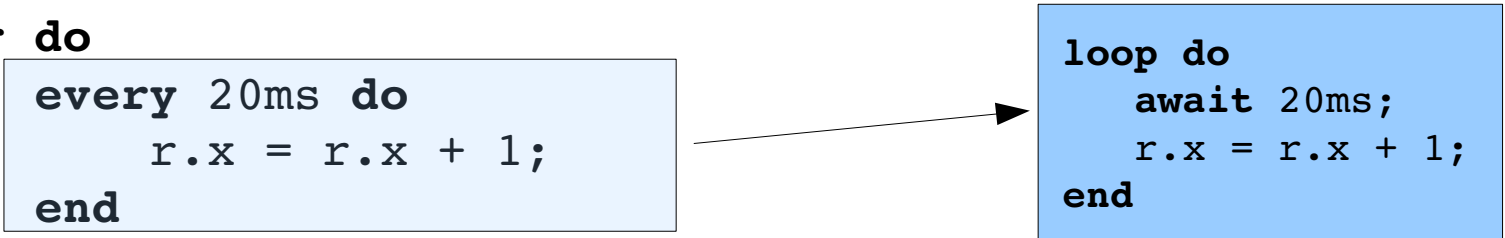
Authors
Francisco Sant'Anna

```
input void SDL_REDRAW; // input event from the environment

<...> // initialize graphical library, renderer, etc.

var _SDL_Rect r = { x=100,y=100,w=20,h=20 };

par do
  every 20ms do
    r.x = r.x + 1;
  end
with
  every SDL_REDRAW do
    _SDL_SetDrawColor(0xFF,0xFF,0xFF,0);
    _SDL_FillRect(&r);
  end
end
```



```
graph LR
    subgraph par_do [par do]
        direction TB
        every["every 20ms do  
r.x = r.x + 1;  
end"]
        loop["loop do  
await 20ms;  
r.x = r.x + 1;  
end"]
        every --> loop
    end
    subgraph with_block [with]
        draw["every SDL_REDRAW do  
_SDL_SetDrawColor(0xFF,0xFF,0xFF,0);  
_SDL_FillRect(&r);  
end"]
    end
    par_do --- with_block
```

```
var SDL_Rect r1 = { 100,100,20,20 };
```

```
var SDL_Rect r2 = { 100,300,20,20 };
```

```
par do
```

```
  every 20ms do  
    r1.x = r1.x + 1;  
  end
```

```
with
```

```
with
```

```
with
```

```
  every SDL_REDRAW do  
    _SDL_SetDrawColor(0xFF,0xFF,0xFF,0);  
    _SDL_FillRect(&r2);  
  end
```

```
end
```

r1

The need for abstractions!

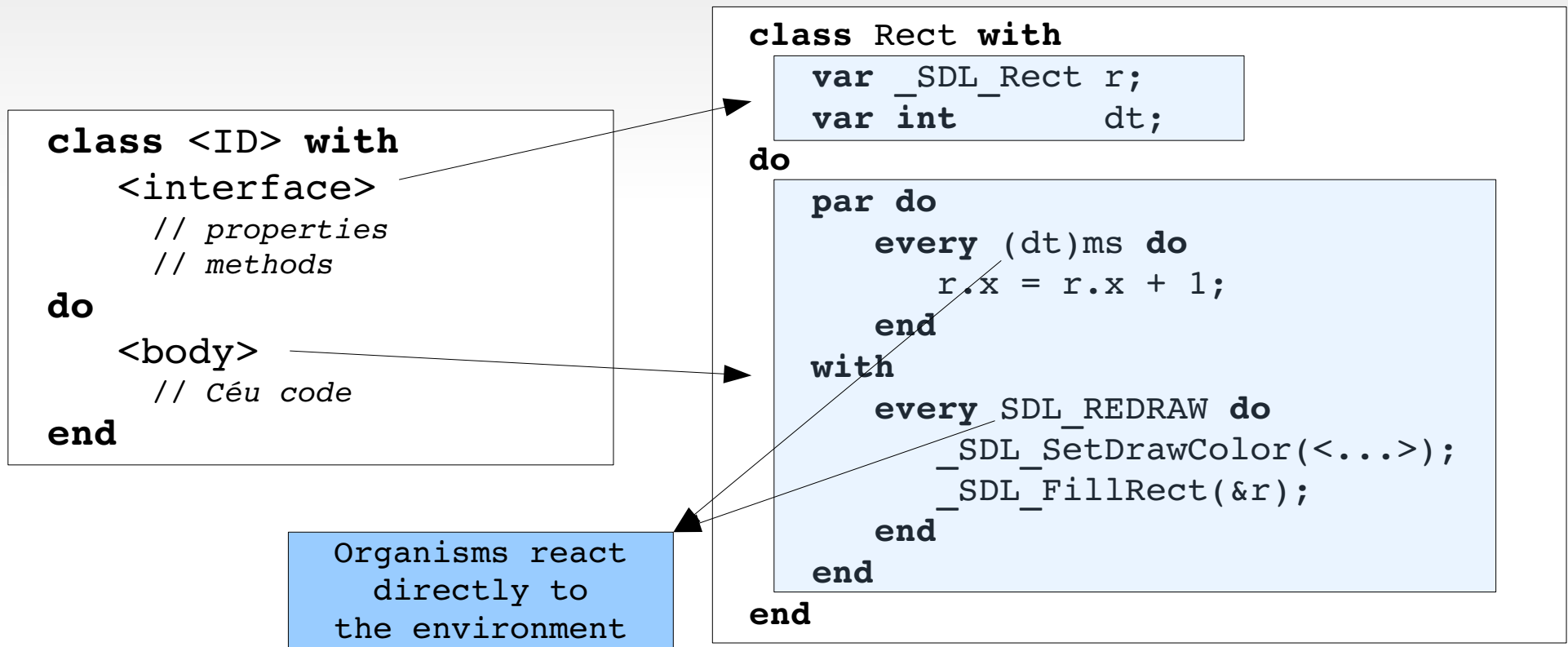
The O.O. way

- Use the observer pattern
 1. Create a Rect class
 2. Create two instances
 3. Add instances as listeners for timers and redrawing events
 - Explicit references
 - No loops and parallel compositions (short-lived methods)
- Memory (heap) management:
 - Memory leaks
 - Dangling pointers
 - Garbage collection

We want something as simple as locals with automatic management!

Céu Organisms

- *Organism ~ (Object + Trails)*



```
class Rect with  
    <interface>  
do  
    <body>  
end
```

```
var Rect r1 with  
    this.r.y = 100;  
    this.dt  = 20;  
end;
```

```
var Rect r2 with  
    this.r.y = 300;  
    this.dt  = 10;  
end;
```

 **await FOREVER;**

A normal variable declaration:

```
var <type> <ID>;
```

(but with a constructor
and **body in parallel**)

[birds - 01]
(bird class)

```
class Bird with
  <...>
  var int speed;    // px/secs
do
  <...>
end
```

Reaction to the
environment is
abstracted inside
the body

```
var Bird b1 with
  this.r.y = 100;
  this.speed = 100;
end;

var Bird b2 with
  this.r.y = 300;
  this.speed = 200;
end;
```

On instantiation,
only the interface
matters

 await FOREVER;

```
class Bird with
  <...>
do
  <...>
end

var int i = 1;
var Bird[5] birds with
  this.r.y = 20 * 4*i;
  this.speed = 100 + 10*i;
  i = i + 1;
end;

await FOREVER;
```

x5


```
class Bird with
    <...>
do
    <...>
end

loop do
    var int i = 1;
    var Bird[5] birds with
        this.r.y    = 20 * 4*i;
        this.speed = 100 + 10*i;
        i = i + 1;
    end;
    await SDL_MOUSEBUTTON;
end
```

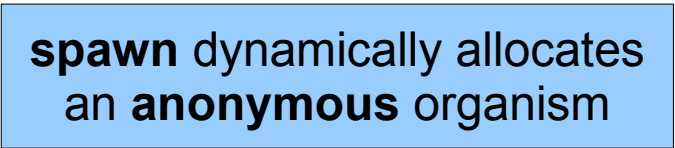
organisms have
lexical scope

organism out of scope:
data reclaimed and body aborted

[birds - 04]
(dynamic instances)

```
class Bird with
  <...>
do
  <...>
end

every 1s do
  spawn Bird with
    this.r.y = 20 + _rand()%HEIGHT;
    this.speed = 100 + _rand()%100;
  end;
end
```



spawn dynamically allocates
an **anonymous** organism

[birds - 04]
(Bird implementation)

```
class Bird with
  <...>
do
  par do
    every SDL_FRAME do
      <animate>
    end
  with
    every SDL_REDRAW do
      <redraw>
    end
  end
end

every 1s do
  spawn Bird with
    this.r.y = 20 + _rand()%HEIGHT;
    this.speed = 100 + _rand()%100;
  end;
end
```

*animation
trail*

*redrawing
trail*

[birds - 05]
(animation break)

```
class Bird with
  <...>
do
  par/and do
    every SDL_FRAME do
      <animate>
      if r.x >= WIDTH-DX then
        break;
      end
    end
  with
    every SDL_REDRAW do
      <redraw>
    end
  end
end

every 1s do
  spawn Bird with
    this.r.y = 20 + _rand()%HEIGHT;
    this.speed = 100 + _rand()%100;
  end;
end
```

only this trail
terminates

animation
trail

redrawing
trail

```
class Bird with
  <...>
do
  par/or do
    every SDL_FRAME do
      <animate>
      if r.x >= WIDTH-DX then
        break;
      end
    end
  with
    every SDL_REDRAW do
      <redraw>
    end
  end
end

every 1s do
  spawn Bird with
    this.r.y = 20 + _rand()%HEIGHT;
    this.speed = 100 + _rand()%100;
  end;
end
```

only this trail
terminates

but this trail
is aborted

spawned organisms
are anonymous

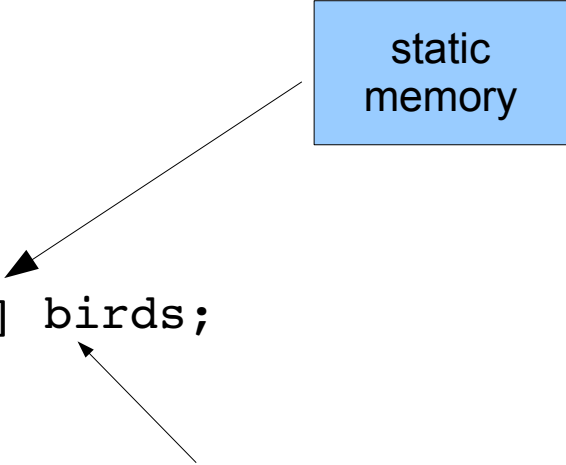
dynamic organisms are
automatically reclaimed
on termination

[birds - 07]
(pools - bounded)

```
class Bird with
    <...>
do
    <...>
end

pool Bird[2] birds;

every 1s do
    spawn Bird in birds with
        this.r.y    = 20 + _rand()%HEIGHT;
        this.speed = 100 + _rand()%100;
    end;
end
```



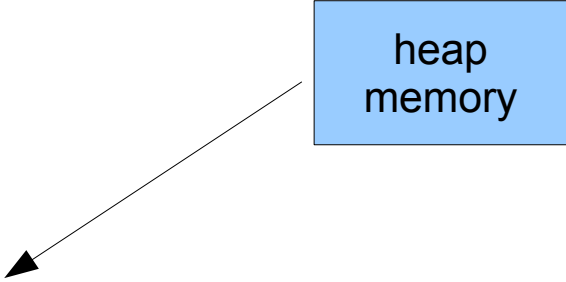
A diagram illustrating memory management. A blue box labeled "static memory" has two arrows pointing to the code. One arrow points to the **pool** Bird[2] birds; line, and the other points to the **in** birds with line in the spawn statement, indicating that the objects in the pool are allocated in static memory.

[birds - 08]
(pools - unbounded)

```
class Bird with
    <...>
do
    <...>
end

pool Bird[] birds;

every 1s do
    spawn Bird in birds with
        this.r.y = 20 + _rand()%HEIGHT;
        this.speed = 100 + _rand()%100;
    end;
end
```



A blue rectangular box labeled "heap memory" is positioned above the line `pool Bird[] birds;`. A black arrow points from the bottom-left corner of this box to the `birds` variable in the pool declaration.

```
class Bird with
  <...>
do
  <...>
end

loop do
  par/or do
    pool Birds[] rs;
    every 1s do
      spawn Bird in birds with
        this.r.y = 20 + _rand()%HEIGHT;
        this.speed = 100 + _rand()%100;
      end;
    end
  with
    await SDL_MOUSEBUTTON;
  end
end
```

pools of organisms also
have lexical scope

static or heap
memory

pool out of scope:
data and body of
all organisms reclaimed

Pointers

- Pointers to organisms are sometimes required:

- **Static:** `&&' operator

```
var T t;
```

```
<use &&t>
```

- **Dynamic:** pool iterators

```
pool T ts;
```

```
loop (T&&) t in ts do
```

```
    <use t>
```

```
end
```

- All pointers are temporary references

[birds-10] (events & iterators)

```
class Bird with
  <...>
  event void collided;
do
  par/or do
    <animate>
  with
    await this.collided;
  with
    every SDL_REDRAW do
      <redraw>
    end
  end
end

par do
  <bird creation>
with
  every SDL_FRAME do
    loop (Bird&&)b1 in birds do
      loop (Bird&&)b2 in birds do
        if <&&b1:r vs &&b2:r> then
          emit b1:collided;
          emit b2:collided;
        end
      end
    end
  end
end
end
```

Interfaces support
internal events

Await to abort the body

For every frame

Iterate over the birds

Emits on collision
detection

[birds - 11]
(falling)

```
class Bird with
  <...>
  event void collided;
do
  par/or do
    <...>
  with
    await this.collided;
    every SDL_FRAME do
      <animate>
      if r.y >= HEIGHT-DY then
        break;
      end
    end
  with
    <...>
  end
end
End

<...>
```

```
class Bird with
  <...>
do
  var bool visible = true;
  par/or do
    <...>
  with
    await this.collided;
    <...>
    par/or do
      await 1s;
      with
        every 100ms do
          visible = not visible;
        end
      end
    end
  with
    every SDL_REDRAW do
      if visible then
        _SDL_RenderCopy(img);
      end
    end
  end
end
end
<...>
```

*During 1 second,
toggle the “visible” state
every 100ms.*



Pointers to organisms

- Alive (valid) within a whole reaction
- Dead (invalid) across reactions

```
var T&& ptr = ...;  
ptr:x = 1;  
await 1s;  
_printf("x = %d\n", ptr:x);
```

*possibly a
dangling pointer*

- Pointers can be tracked across reactions

```
var T&& ptr = ...;  
ptr:x = 1;  
watching ptr do  
    await 1s;  
    _printf("x = %d\n", ptr:x);  
end
```

```
par/or do  
    await ptr._killed;  
with  
    <...>  
end
```

```
class Bird with
  <...>
do
  <...>
end

par do
  <bird creation>
with
  loop do
    var _SDL_MouseEvent&& mse = await SDL_MOUSEBUTTONDOWN;
    var Bird&& ptr = null;
    loop (Bird&&)b in birds do
      if <mse vs &&b:r> then
        ptr = b;
        break;
      end
    end
    if ptr != null then
      watching ptr do
        every SDL_REDRAW do
          _SDL_DrawLine(WIDTH/2,HEIGHT,
                        ptr:r.x,ptr:r.y);
        end
      end
    end
  end
end
end
```

*Iterator that
checks if a bird
was clicked*

*Watches the bird
while drawing
a line*

game.ceu

- sprites
- salto (vy)
- aceleração (ax,ay)
- **objetos dinâmicos**
- **colisão**
- ???
- Mini-SDL
 - Apresentações 10/05
 - ~250 LoC/Céu
 - **pode ser em C!**

Compilando / Executando

```
$ su - arduino    # (Password: ArdDev16)

$ git clone https://github.com/fsantanna/reativos
# OU
$ cd ~/reativos/
$ git pull

$ cd ~
$ tar xvf /home/extra/fsantanna/ceu.tgz
$ exec bash
$ ceu

# game.ceu
$ cd ~/ceu/ceu-sdl
$ make CEUFILE=<repo-reativos>/reativos/code/sdl/game.ceu
$ <repo-reativos>/code/sdl/game.exe

# birds.ceu
$ cd ~/ceu/ceu-sdl
$ make CEUFILE=<repo-reativos>/reativos/code/sdl/birds.ceu
$ cd <repo-reativos>/code/sdl/ && ./birds.exe
```