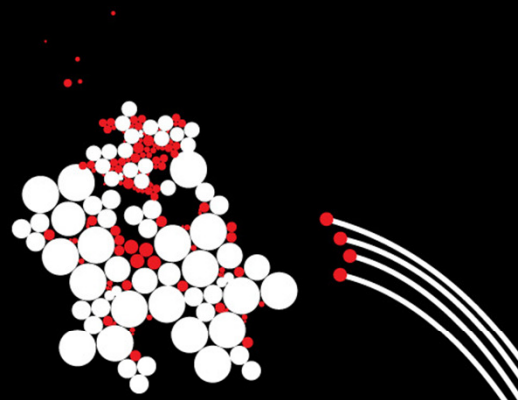
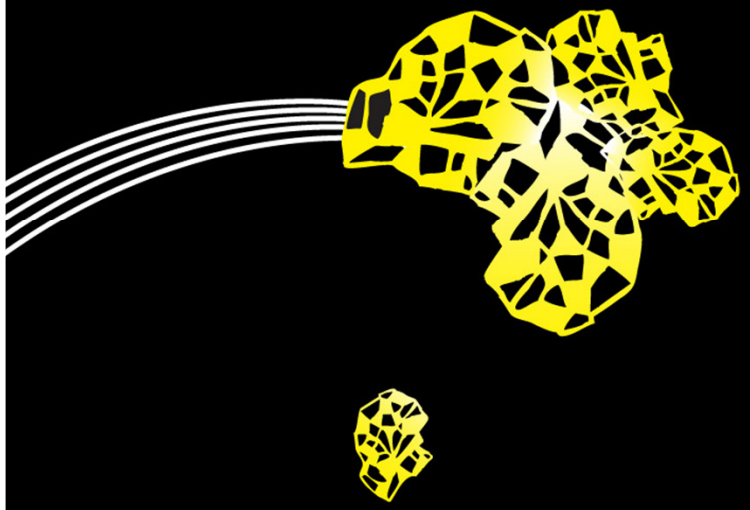
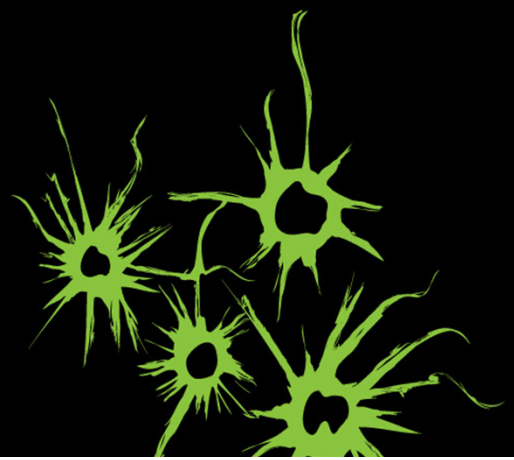
UNIVERSITY OF TWENTE.



Requirements for tailorable and composable control flow





Control flow

- What is control flow
- Examples of control flow constructs
  - if statement
  - Goto
  - Exceptions
  - Continuations
  - Closures
  - Etc..

Motivation

- What do we want?
  - To not be limited by predefined language constructs
  - To be able to implement tailorable and composable control-flow mechanics that can be shared through libraries.

Classical

```
var out = null
try {
  out = File.open("Some file that does not exist")
  out.write("Everything went fine")
} finally {
  if (out != null)
    out.close()
}
```

First class

```
var out = null
tryFinally({
  out = File.open("Some file that does not exist")
  out.write("Everything went fine")
}, {
  if (out != null)
    out.close()
})
```

Differences

- Embedded in language vs function
- Advantages of the first class example
  - More control over the functionality
  - Programmers can create own variants
  - Keeps the language and syntax simpler
  - Create a library with control flow functions
    - Better reusability
    - Have a large choice of mechanics without having to create them yourself.

Existing first-class control-flow representations

- Possible underlying technology
- Programmable control flow constructs

Javascript	eval
C#	Delegates
Java	Exceptions
C	goto
Scala	continuations

JavaScript (eval)

- Eval function
  - Executes a string as source code
  - Binding done at execution time, at execution scope
  - No type or variable checking done by a compiler

C# (Delegates)

- Ability to pass a delegate containing source code
- Where the delegate is created, variables referred to in the delegate need to exist
- Values can be changed after the delegate is created, invoking the delegate will use these updated values
- Scope used by the delegate is always the scope where the delegate was created, regardless of execution scope.

Java (Exceptions)

- Used for error handling
- Changes execution order of program
- One way control flow constructs
  - Can only throw an exception towards a catch block higher in the stack

C (goto)

- Allows the to jump to a label specified in the source code
- Only changes the point of execution
- Limitted to which labels can be jumped to
  - Not possible to jump to labels in other functions

Scala (continuations)

```
var s = "This value is never used"
reset {
  s = "This will be printed first"
  shift { (block: Unit => Unit) =>
  println(s)
  block()
  println(s)
  }
  s = "This will be printed second";
}
```

Scala (continuations)

- `reset` – enclose the continuation
- `Shift` – pass continuation to its body

- Continuation can be invoked multiple times
- Values for variables used in the continuation are updated

Simulate existing control flow constructs

- Simulate `if` statements
- Simulate loops
  - `for` loop
  - `while` loop
- Simulate `break` statement

`if` statement in JavaScript

```
function ifTrueEslse(b, t, f) {
  var tfArray=new Array();
  tfArray[0]=f;  tfArray[1]=t;
  eval(tfArray[Number(b)]);
}
Function main() {
  var sTrue = “alert(‘waar’)”;
  var sFalse = “alert(‘onwaar’)”;
  ifTrueElse(true, sTrue, sFalse);
}
```

`while` loop in C#

```
doWhileLoop(Delegate check, Delegate block,
Delegate inc) {
  if (check()) {
    block(); inc(); doWhileLoop(check, block, inc); }
}
Main () {
  int i = 0;
  Delegate check = new Delegate(i < 10);
  Delegate block = new Delegate(write(i);
  Delegate inc = new Delegate(i++);
  doWhileLoop(check, block, inc);
}
```

`for` loop problems

- A `for` loop can initialize its own variable
- The delegate “code” in the C# example would have to have acces
- We need to be able add or attatch a different scope to the delegates

Comparison

- Can it be used to make first class implementations
  - JavaScript is a first class implementation but is not what we want
  - C# allows us to make a first class implementation
- Scoping
  - JavaScript only uses current execution scope
  - C# can only access variables that existed in the scope the delegate was created, at the time the delegate was created
- Variable binding
  - Both imlementations use reference binding
- Reusability
  - Both implementations are reuseable (can be invoked multiple times with different behavior based on the value of the varialbes)

Evalution

- Not one of the tested mechanics satisfies all our needs
- We need the freedom of passing around the constructs, without it having to worry about where the construct is invoked
- Scope of variables is key
  - Scopes used by JavaScript and C#
- None of the mechanics give us the ability to alter the scope

Conslusions

- What do need to
  - First class, The ability to pass around the control-flow mechanics while not having to have prior knowledge of the execution point.
  - Selectable scopes, The control-flow mechanics should be able to use other scope then the execution scope of the control-flow mechanics .
  - Variable binding, The variables should be bound by reference to allow our control-flow mechanics to change variables.

Questions