Dynamic @ Work

S. Ducasse

www.pharo.org

Topics

- TDD
- Classes as objects
- Stack as objects
- Black magic

Transparent Object Migration

Define a class Box

Create an instance of Box

Open an inspector

Change class Box

Instance gets migrated automatically

On the fly recompilation

When a method is not found in the debugger,

ask for the creation of a method on the fly

the system compiles on the spot a special method, then reexecutes the method

it raises a shouldBeImplemented exception

then you can edit the method in the debugger

then proceed and the program continues to run

Classes are first class objects

- Structure (instance format)
- Inheritance tree
- Methods

Accessing structural information

```
Dictionary instVarNames

Dictionary allInstVarNames #('tally' 'array')
```

Dictionary subclasses

{IdentityDictionary. WeakKeyDictionary.
WeakValueDictionary. PluggableDictionary.
LiteralDictionary. MethodDictionary. KeyedTree}

Dictionary allSubclasses

```
 a Set(MethodDictionary KeyedTree SystemDictionary
IdentityDictionary WeakIdentityKeyDictionary
LiteralDictionary WeakKeyToCollectionDictionary
WeakKeyDictionary WeakValueDictionary
PluggableDictionary)
```

Instances and pointers

Dictionary allInstances size

1294 :)

pointersTo

To get all the pointers to a given object :)

anObject pointersTo

returns all the pointers pointing to this object

Execution Stack as an Object

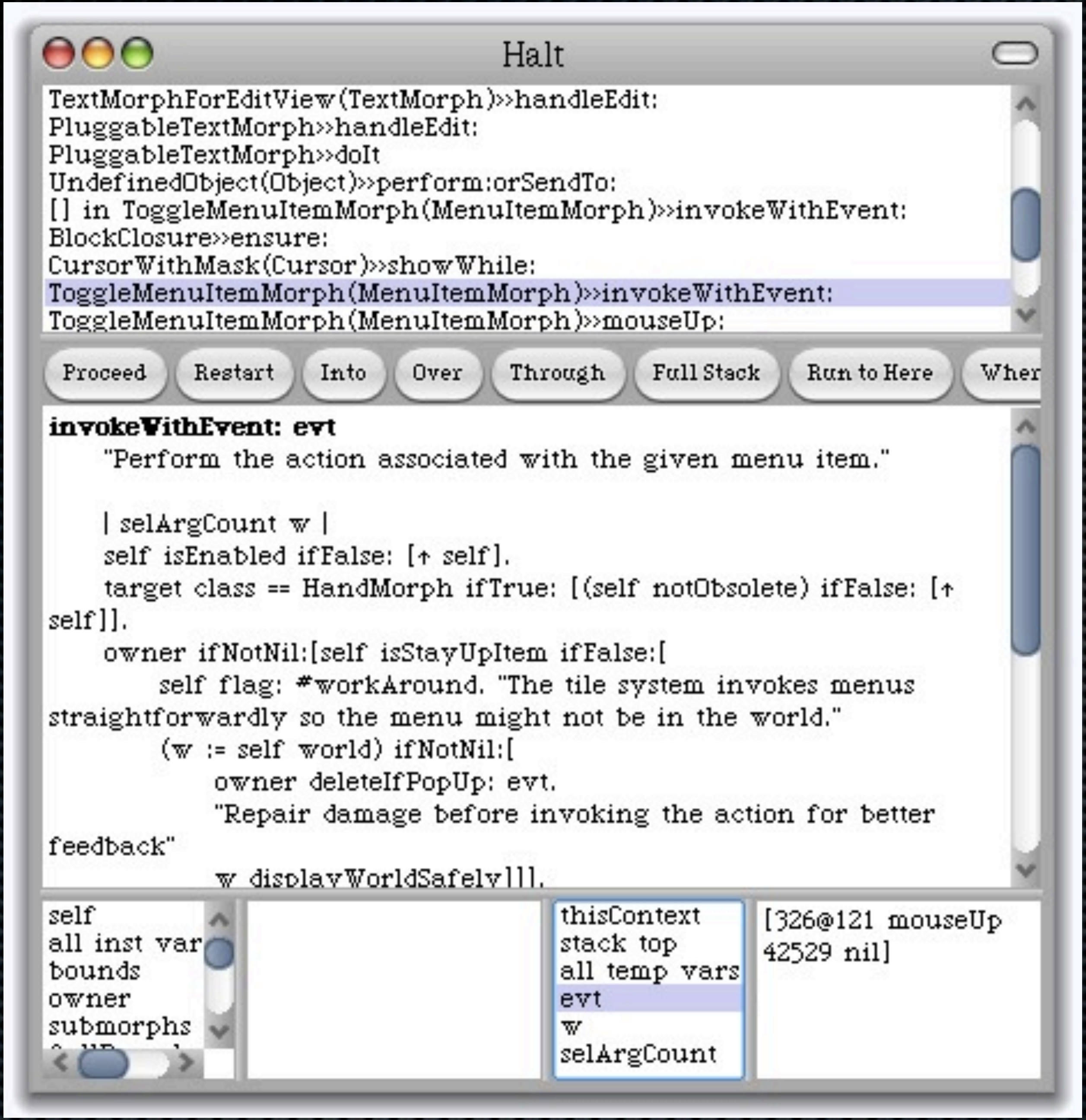
- To be able to define exceptions from within the language
- Debugger support!
- Advanced debugging
- Continuation

thisContext

returns an object that represents the method activation

can walk the stack

put self halt in the code to see it and walk.



```
TextMorphForEditView(TextMorph)>>handleEdit:
PluggableTextMorph>>handleEdit:
PluggableTextMorph>>doIt
UndefinedObject(Object)>>perform:orSendTo:
[] in ToggleMenuItemMorph(MenuItemMorph)>>invokeWithEvent:
BlockClosure>>ensure:
CursorWithMask(Cursor)>>showWhile:
ToggleMenuItemMorph(MenuItemMorph)>>invokeWithEvent:
ToggleMenuItemMorph(MenuItemMorph)>>mouseUp:
```

```
invokeWithEvent: evt
    "Perform the action associated with the given menu item."

    | selArgCount w |
    self isEnabled ifFalse: [↑ self].
    target class == HandMorph ifTrue: [(self notObsolete) ifFalse: [↑
self]].
    owner ifNotNil:[self isStayUpItem ifFalse:[
        self flag: #workAround. "The tile system invokes menus
straightforwardly so the menu might not be in the world."
        (w := self world) ifNotNil:[
            owner deleteIfPopUp: evt.
            "Repair damage before invoking the action for better
feedback"
            w displayWorldSafely]]].
```

```
self
all inst var
bounds
owner
submorphs
```

```
thisContext
stack top
all temp vars
evt
w
selArgCount
```

```
[326@121 mouseUp
42529 nil]
```

Powerful breakpoints?

Would be so good if we could say:

“Stop method bar ***only if it is*** invoked from method testBar” i.e.

```
bar

    …

    self haltIf: #testBar….

    …
```

And we have the following behavior...

```
foo
    self bar
```

Executing foo does ***not*** stop

while executing testBar ***should stop***

haltIf: in 6 lines

```
Object>>haltIf: aSelector

        | cntxt |
        cntxt := thisContext.
        [cntxt sender isNil] whileFalse: [
                cntxt := cntxt sender.
                (cntxt selector = aSelector)
                                ifTrue: [ Halt signal]]
```

Basis of Seaside

Powerful dynamic web framework for dynamic web applications

www.seaside.st

book.seaside.st

Black magic... pointer swapping

anObject become: anotherObject

All the pointers pointing to anObject points now to anotherObject and the inverse atomatically

```
| pt1 pt2 pt3 |
pt1 := 0@0.
pt2 := pt1.
pt3 := 100@100.
pt1 become: pt3.
self assert: pt2 = (100@100).
self assert: pt3 = (0@0).
self assert: pt1 = (100@100).
```

Changing the class of an object

Class>>adoptInstance: anInstance

"Change the class of anInstance to me.returns the class rather than the modified instance”

Obviously different from become:

```
| behavior model |
behavior := Behavior new.
behavior superclass: Model.
behavior setFormat: Model format.
model := Model new.
model primitiveChangeClassTo: behavior new.
behavior compile: 'thisIsATest  ^ 2'.
self assert: model thisIsATest = 2.
self should: [Model new thisIsATest]
	raise: MessageNotUnderstood.
```

Simple model

- Powerful reflective system but
- we will revisit it
  - Mirrors
  - Layered
  - AST node level annotation