

# Using Spec to Build a UI

Benjamin Van Ryseghem, Stephane Ducasse,  
**Johan Fabry**



**Pleiadi**

# What is it for?

---

✕ - □
Halt

UndefinedObject	Dolt
OpalCompiler	evaluate
SmalltalkEditor	evaluateSelectionAndDo:
SmalltalkEditor	evaluateSelection
[...] in PluggableTextMorph	dolt
[...] in PluggableTextMorph	handleEdit:

✕ - □
Live Robot Programming UI

+Var
+Mach
+State
+Trans
+Event
Machines:
Selected Machine:

```

2 (var sensor := [LRPEV3Bridge sensor])
3 (var mright := [LRPEV3Bridge motorA])
4 (var mleft := [LRPEV3Bridge motorD])
5 (var s := [20])
6 (var dist := [50])
7 (machine follower
8   (state moving
9     (running [
10      mright turn: dist atSpeed: s.
11      mleft turn: dist atSpeed: s.
12     ]))
13   (state looking
14     (machine lookalgo
15       (var looktime := [2000])
16       (state lookright
17         (running [mright turn: dist /

```

follower

looking  
lookalgo

---

Variables:

sensor	false
mright	false
mleft	false
s	20
dist	50
looktime	2000

Toggle Animation

Pause Int.
Step Int
Reset Int.
Inspect bot
Select va
Set
Inspect

< >
EyelInspector
▼

- self
- ast
- source
- context

an OpalCompiler

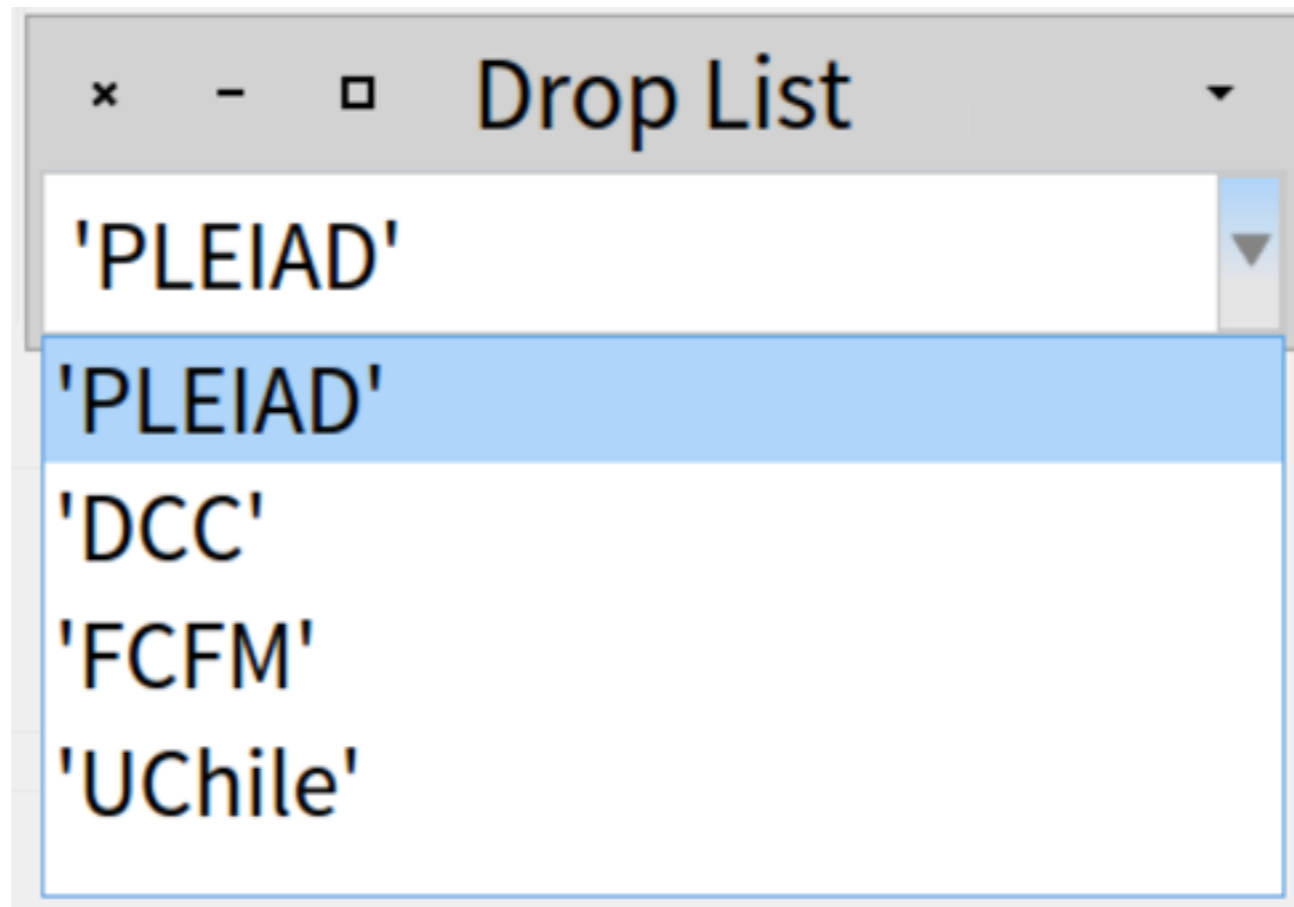
- thisContext
- stackTop
- all temp vars
- value
- selectedSource
- itsSelection

OpalCompiler>>evaluate

# Reuse is key

---

```
|dl|  
dl := DropListModel new.  
dl items: { 'PLEIAD' . 'DCC' . 'FCFM' . 'UChile' }.  
dl openWithSpec.
```



```
LRPUserInterface new openWithSpec
```

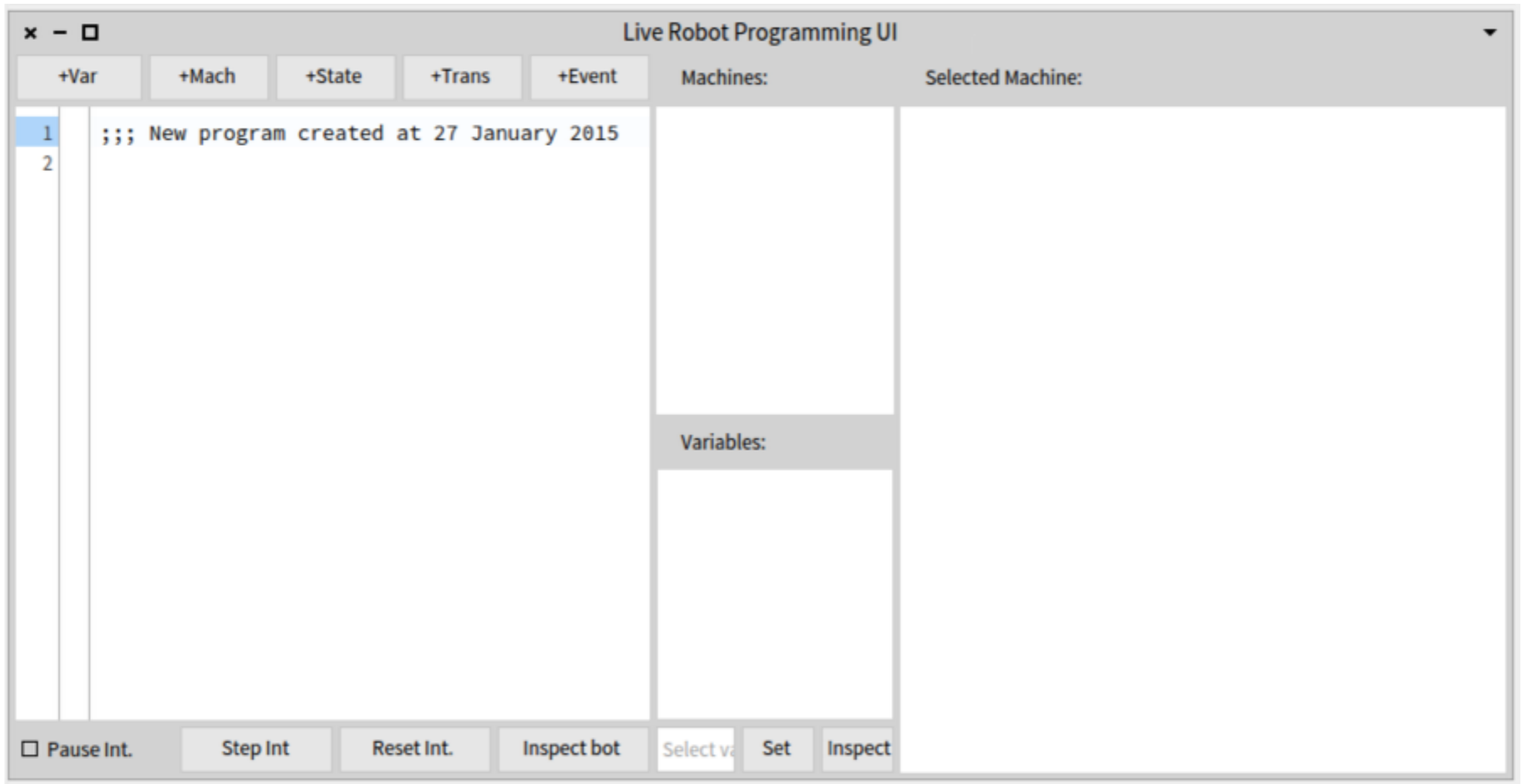
Live Robot Programming UI

+Var +Mach +State +Trans +Event Machines: Selected Machine:

```
1 ;;; New program created at 27 January 2015
2
```

Variables:

Pause Int. Step Int. Reset Int. Inspect bot Select va Set Inspect





Every UI is reusable  
by default

```
|d|  
d := DropListModel new.  
d items: { 'PLEIAD' . 'DCC' . 'FCFM' . 'UChile' }.  
d openWithSpec.
```

The screenshot shows a software development environment with a class hierarchy on the left and a list of items on the right. The class hierarchy includes ContainerModel, DiffModel, DropListModel (highlighted), ImageModel, LabelModel, ListModel, and IconListModel. The list of items includes -- all --, initialization, private, protocol (highlighted), protocol-events, getList, iconHolder, iconHolder:, items: (highlighted), listItems, listSize, and resetSelection.

# Let's build a UI

---

Live Robot Programming UI

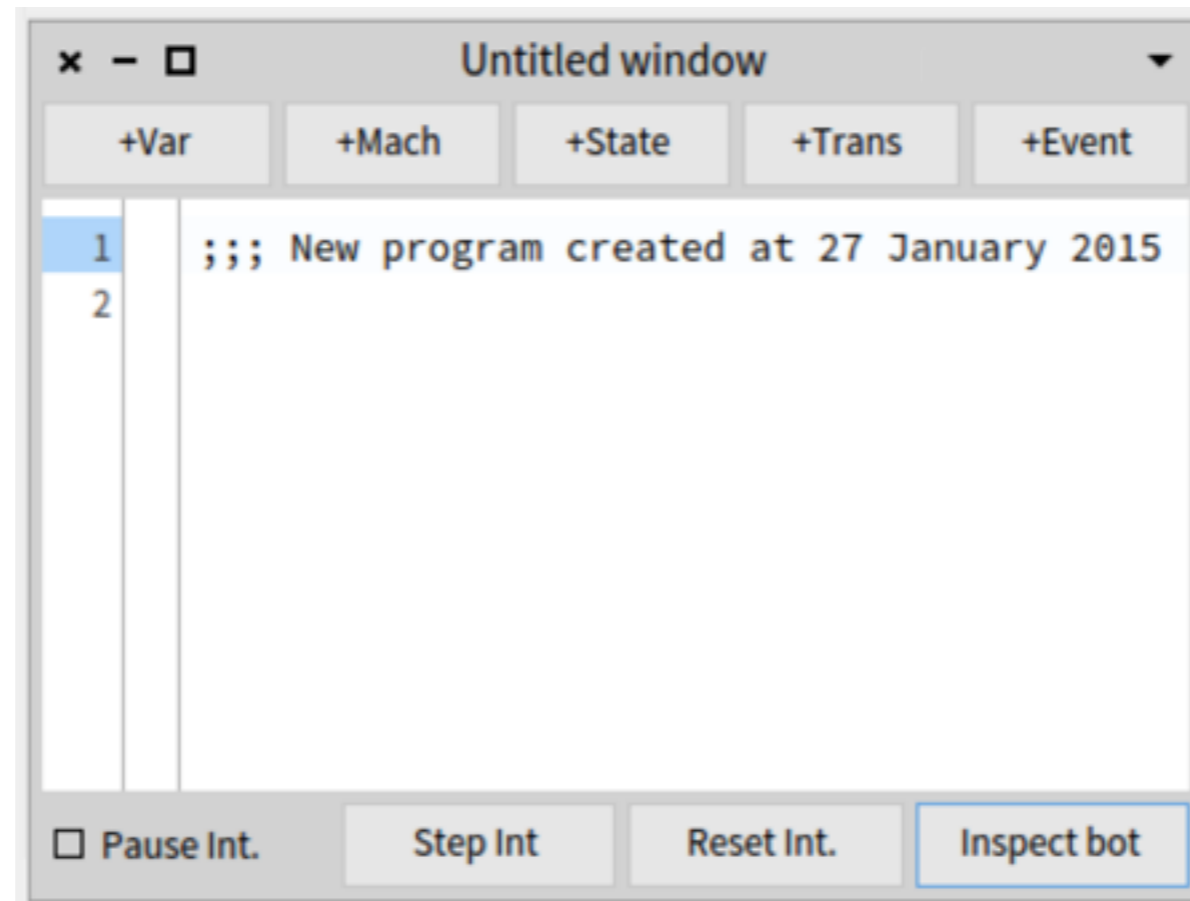
+Var +Mach +State +Trans +Event Machines: Selected Machine:

```
1 ;;; New program created at 27 January 2015
2
```

Variables:

Pause Int. Step Int. Reset Int. Inspect bot Select va Set Inspect

The image shows a software window titled "Live Robot Programming UI". At the top, there are five buttons: "+Var", "+Mach", "+State", "+Trans", and "+Event". To the right of these buttons are two labels: "Machines:" and "Selected Machine:". Below these is a large text area containing two lines of code: line 1 is ";;; New program created at 27 January 2015" and line 2 is empty. To the right of the code area is a panel labeled "Variables:". At the bottom of the window is a control bar with several buttons: " Pause Int.", "Step Int.", "Reset Int.", "Inspect bot", "Select va", "Set", and "Inspect".



ComposableModel subclass: #LRPPProgramEditor

instanceVariableNames: 'pv pm ps pt pe text pause step reset inspBot  
parser lastmodel interpreter codeChangeBlock'  
classVariableNames: ''  
category: 'LiveRobotics-UI'

# Three things about widgets

- Widgets are selfish
- Widgets are cooperative
- Widgets know their place in society



# Selfish

## initializeWidgets

```
inspBot := self newButton.  
inspBot label: 'Inspect bot'.  
inspBot action: [ self interpreter robotValue inspect ].  
  
text := RubScrolledTextMorph new  
    vResizing: #spaceFill; hResizing: #spaceFill;  
    beWrapped; autoAccept: true;  
    updateTextWith: self newStartText;  
    withLineNumbers; withTextSegmentIcons;  
    asSpecAdapter.
```

---

# Cooperative

## initializePresenter

```
step action: [pause state: true. self interpreter stepRun].  
reset action: [|res|  
  res := self parser parse: text widget text styleOn: text widget.  
  res isNil ifTrue:[res := #()].  
  pause state ifTrue: [ pause state: false ].  
  self interpreter restartInterpreter: res.  
  self lastmodel: res.  
  self codeChangeBlock value: res].
```

# Their place

(... which is very classy ...)

## defaultSpec

```
<spec: #default>
```

```
^ SpecLayout composed
```

```
  newColumn: [:col]
```

```
    col newRow: [ :row |row add: #pv; add: #pm; add: #ps; add: #pt; add: #pe]  
      height: (self toolbarHeight);
```

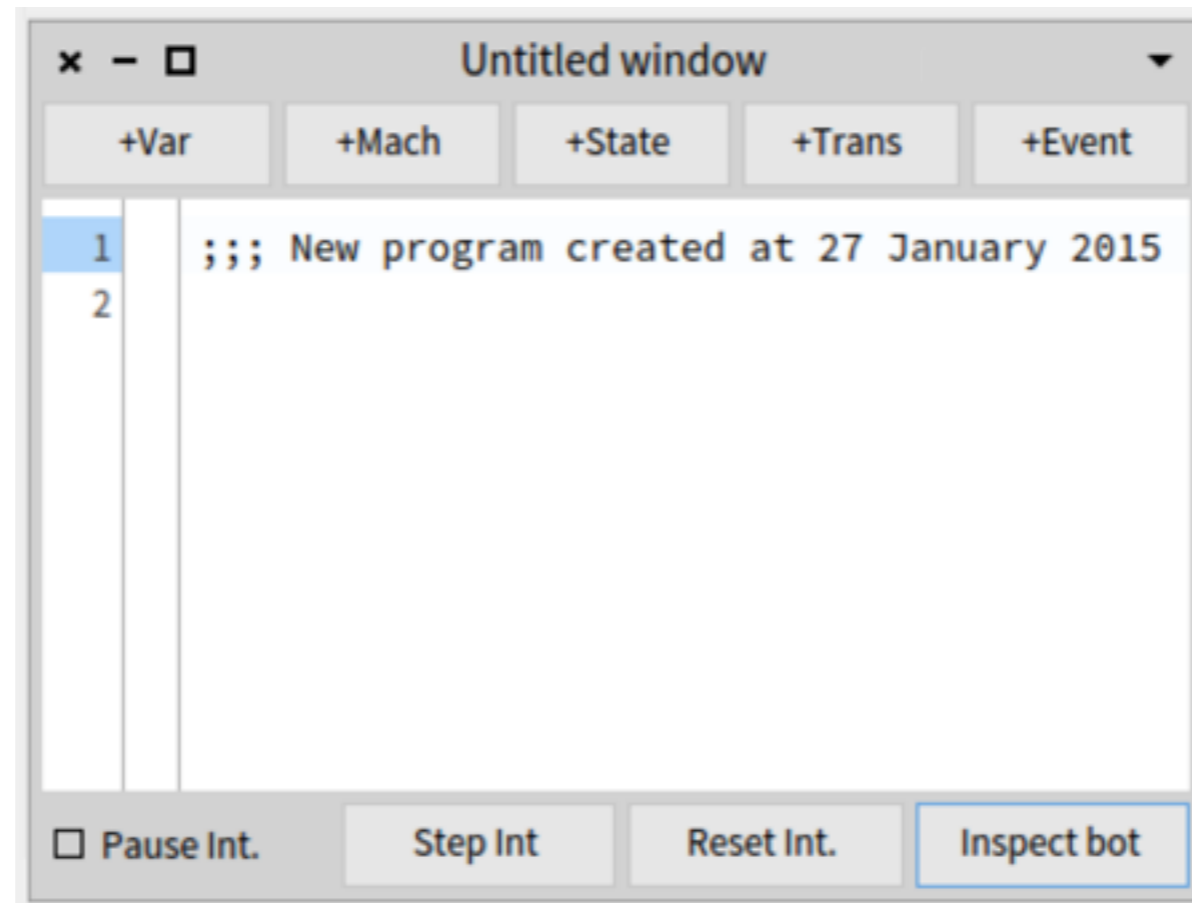
```
    add: #text ;
```

```
    newRow: [ :row |row add: #pause; add: #step; add: #reset; add: #inspBot]  
      height: (self toolbarHeight)]
```

**That's all, folks!**

# Reuse!

---



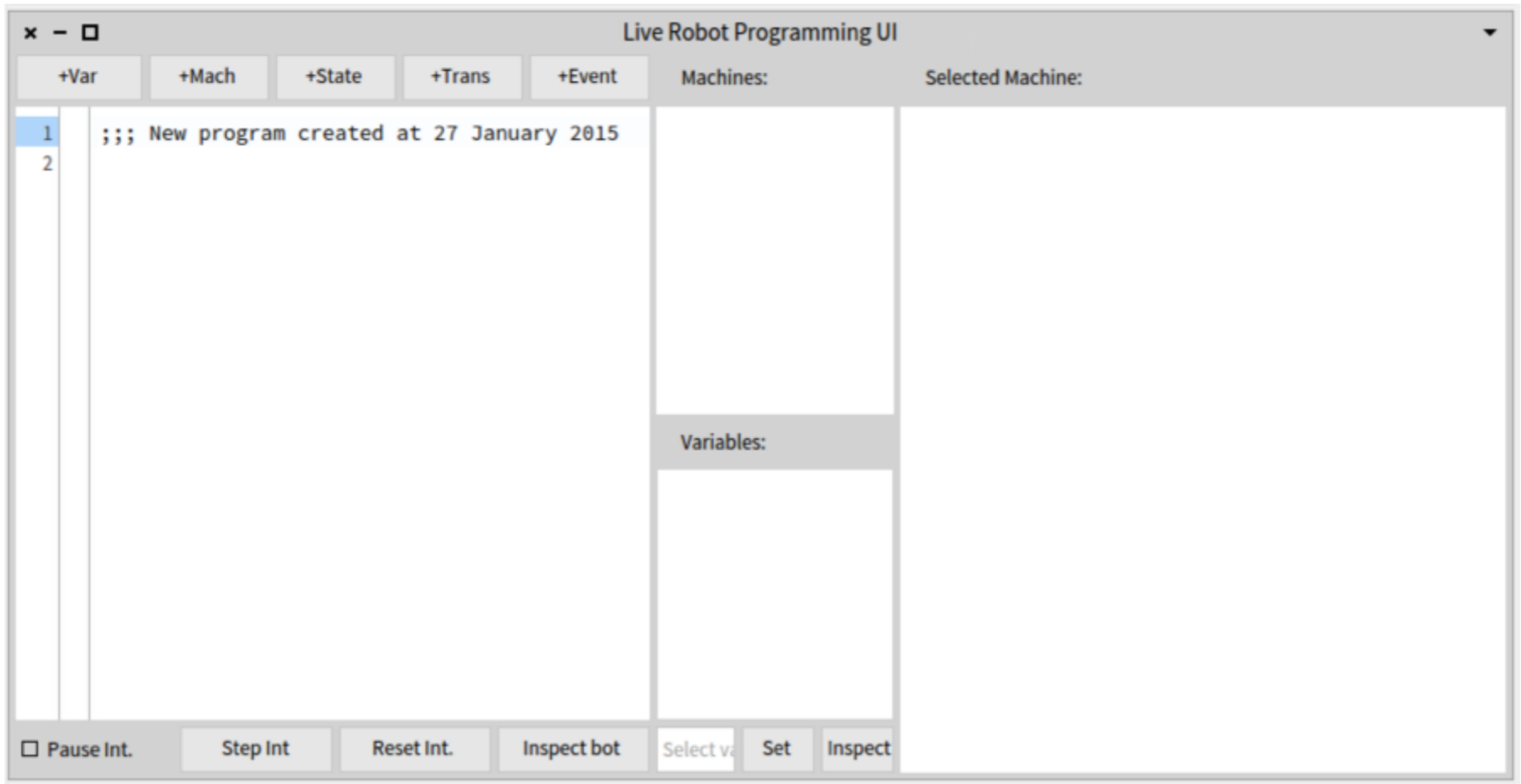
Live Robot Programming UI

+Var +Mach +State +Trans +Event Machines: Selected Machine:

```
1 ;;; New program created at 27 January 2015
2
```

Variables:

Pause Int. Step Int. Reset Int. Inspect bot Select va Set Inspect



LRPDummyInterpreter	-- all --	codeChangeBlock
LRPErrorHandlerUI	accessing	codeChangeBlock:
LRPGrowlErrorHandler	actions	interpreter
LRPProgramEditor	♦ initialization	interpreter:
LRPProgramVisualiz	protocol	



```
ComposableModel subclass: #LRPUIterface  
instanceVariableNames: 'codePane visPane'  
classVariableNames: ''  
category: 'LiveRobotics-UI'
```

# Selfish

## **initializeWidgets**

```
codePane := self instantiate: LRPPProgramEditor.  
visPane := self instantiate: LRPPProgramVisualization.  
visPane code: codePane.
```

# Cooperative

# Their place

## defaultSpec

```
<spec: #default>
```

```
^ SpecLayout composed
```

```
  newColumn: [:tcol]
```

```
    tcol newRow: [:trow]
```

```
      trow add: #codePane width: 400;
```

```
        add: #visPane ]].
```

**That's all, folks!**

# Remember

---

Reuse is key

```
|d|  
d := DropListModel new.  
d items: { 'PLEIAD' . 'DCC' . 'FCFM' . 'UChile' }.  
d openWithSpec.
```

The screenshot shows a GUI inspector window with a tree view on the left and a detailed view on the right. The tree view lists several model classes: ContainerModel, DiffModel, DropListModel (highlighted), ImageModel, LabelModel, ListModel, and IconListModel. The detailed view for DropListModel shows the following structure:

- all --
- initialization
- private
- protocol (highlighted)
- protocol-events

The right pane shows the state of the selected protocol:

- getList
- iconHolder
- iconHolder:
- items: (highlighted)
- listItems
- listSize
- resetSelection



- Widgets are selfish
  - ➔ `initializeWidgets`
- Widgets are cooperative
  - ➔ `initializePresenter`
- Widgets know their place in society
  - ➔ `<spec: #default>`

**Use the source, dude!**