



VisuAlea, Towards a Scientific Modelling Environment using Visual Programming

Christophe Pradal, Daniel Barbeau, Thomas Cokelaer, Eric Moscardi

► **To cite this version:**

Christophe Pradal, Daniel Barbeau, Thomas Cokelaer, Eric Moscardi. VisuAlea, Towards a Scientific Modelling Environment using Visual Programming. EuroSciPy 2010, 2010, Paris, France. 2010. <hal-00831794>

HAL Id: hal-00831794

<https://hal.inria.fr/hal-00831794>

Submitted on 7 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

VisuAlea, Towards a Scientific Modelling Environment using Visual Programming



Christophe Pradal^{1,2} Daniel Barbeau¹, Thomas Cokelaer¹
Eric Moscardi¹

¹INRIA, ²CIRAD



OpenAlea Goals

OpenAlea is an open source platform for modelling plant development and functioning at different scales.

Sharing knowledge

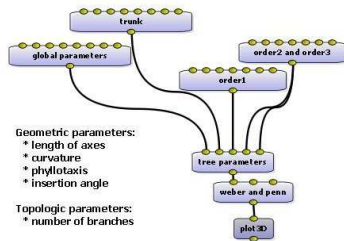
- Reuse software and tools
- Share development between various labs
- Share databases and training efforts

Common software platform

- Integration of existing software & tools
- Rapid development of new models
- Enhance accessibility
- Quality rules

Design choices

- Open Source scientific community
 - Distributed development (sprints)
- Language centric (Python)
 - Common modelling language
 - Glue language
- Component architecture
 - Dynamic composition
 - High-level dataflow approach
- Visual programming (**VisuAlea**)
 - Graphical model representation
 - Automatic GUI generation
- Shared deployment tools
 - Build, packaging, installation, distribution, update



Visual Programming

Visual Programming Environments

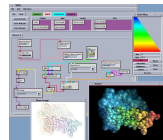
LabView, VTK, Vision, Orange, VisTrails, ...

Advantages

- Interactive creation and modification of flexible workflows
- Visual representation of the structure of a model
- Dynamic composition of software components

Drawbacks

- Less expressive than textual languages (for, while)



Vision (Sanner *et al.*, 2002)



Orange (Demsar *et al.*, 2004)



VisTrails (Freire *et al.*, 2005)

VisuAlea

The image shows the VisuAlea software interface with several components highlighted by red arrows and text:

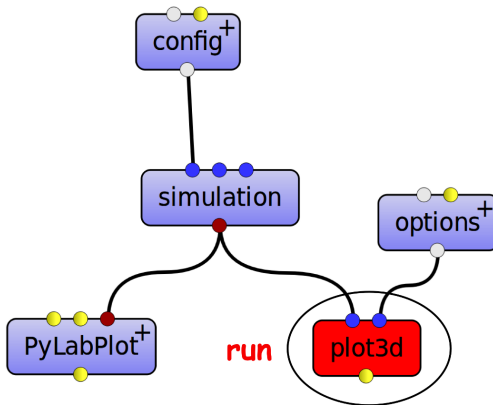
- Package Manager:** Located on the left, it shows a list of installed packages such as 'demo', 'gimp', 'lumpynt', 'm2s3pc', 'exam_note', 'tutorial', 'TestMaterialOut1', 'TestMaterialOut2', 'TestCreateViewCurl', 'TestDirectExe1', 'TestDirectExe2', 'TestDirectExe3', 'TestDirectExe4', 'TestDirectExe5', 'TestDirectExe6', 'TestDirectExe7', 'TestDirectExe8', 'TestDirectExe9', 'TestDirectExe10', 'TestDirectExe11', 'TestDirectExe12', 'TestDirectExe13', 'TestDirectExe14', 'TestDirectExe15', 'TestDirectExe16', 'TestDirectExe17', 'TestDirectExe18', 'TestDirectExe19', 'TestDirectExe20', 'TestDirectExe21', 'TestDirectExe22', 'TestDirectExe23', 'TestDirectExe24', 'TestDirectExe25', 'TestDirectExe26', 'TestDirectExe27', 'TestDirectExe28', 'TestDirectExe29', 'TestDirectExe30', 'TestDirectExe31', 'TestDirectExe32', 'TestDirectExe33', 'TestDirectExe34', 'TestDirectExe35', 'TestDirectExe36', 'TestDirectExe37', 'TestDirectExe38', 'TestDirectExe39', 'TestDirectExe40', 'TestDirectExe41', 'TestDirectExe42', 'TestDirectExe43', 'TestDirectExe44', 'TestDirectExe45', 'TestDirectExe46', 'TestDirectExe47', 'TestDirectExe48', 'TestDirectExe49', 'TestDirectExe50', 'TestDirectExe51', 'TestDirectExe52', 'TestDirectExe53', 'TestDirectExe54', 'TestDirectExe55', 'TestDirectExe56', 'TestDirectExe57', 'TestDirectExe58', 'TestDirectExe59', 'TestDirectExe60', 'TestDirectExe61', 'TestDirectExe62', 'TestDirectExe63', 'TestDirectExe64', 'TestDirectExe65', 'TestDirectExe66', 'TestDirectExe67', 'TestDirectExe68', 'TestDirectExe69', 'TestDirectExe70', 'TestDirectExe71', 'TestDirectExe72', 'TestDirectExe73', 'TestDirectExe74', 'TestDirectExe75', 'TestDirectExe76', 'TestDirectExe77', 'TestDirectExe78', 'TestDirectExe79', 'TestDirectExe80', 'TestDirectExe81', 'TestDirectExe82', 'TestDirectExe83', 'TestDirectExe84', 'TestDirectExe85', 'TestDirectExe86', 'TestDirectExe87', 'TestDirectExe88', 'TestDirectExe89', 'TestDirectExe90', 'TestDirectExe91', 'TestDirectExe92', 'TestDirectExe93', 'TestDirectExe94', 'TestDirectExe95', 'TestDirectExe96', 'TestDirectExe97', 'TestDirectExe98', 'TestDirectExe99', 'TestDirectExe100'.
- Widgets:** A central window titled 'Curve2D' displays a 2D plot with a grid and a curve. Below it, a 'Material' window shows properties like 'name', 'ambient', 'emission', 'shininess', and 'specular'.
- DataPool:** A window at the bottom left shows a list of data sources: 'scene1', 'scene2', 'scene3', 'scene4', 'scene5', 'scene6', 'scene7', 'scene8', 'scene9', 'scene10', 'scene11', 'scene12', 'scene13', 'scene14', 'scene15', 'scene16', 'scene17', 'scene18', 'scene19', 'scene20', 'scene21', 'scene22', 'scene23', 'scene24', 'scene25', 'scene26', 'scene27', 'scene28', 'scene29', 'scene30', 'scene31', 'scene32', 'scene33', 'scene34', 'scene35', 'scene36', 'scene37', 'scene38', 'scene39', 'scene40', 'scene41', 'scene42', 'scene43', 'scene44', 'scene45', 'scene46', 'scene47', 'scene48', 'scene49', 'scene50', 'scene51', 'scene52', 'scene53', 'scene54', 'scene55', 'scene56', 'scene57', 'scene58', 'scene59', 'scene60', 'scene61', 'scene62', 'scene63', 'scene64', 'scene65', 'scene66', 'scene67', 'scene68', 'scene69', 'scene70', 'scene71', 'scene72', 'scene73', 'scene74', 'scene75', 'scene76', 'scene77', 'scene78', 'scene79', 'scene80', 'scene81', 'scene82', 'scene83', 'scene84', 'scene85', 'scene86', 'scene87', 'scene88', 'scene89', 'scene90', 'scene91', 'scene92', 'scene93', 'scene94', 'scene95', 'scene96', 'scene97', 'scene98', 'scene99', 'scene100'.
- Dataflow:** A complex graph of interconnected nodes and arrows representing the data flow between different components.
- Component:** A specific node in the dataflow graph, labeled 'LSystem modeling'.
- Python Interpreter:** A window at the bottom right showing a Python code editor with the following code:


```

      >>> scene = datapool['scene1']
      >>> from openalea.core.dio import function
      >>> node = package['multiple_tutorial']['test_function']
      >>> function(node)
      >>>
      
```

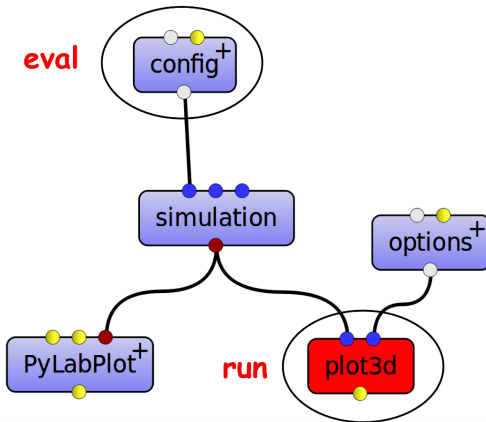
Dataflow Evaluation

Demand driven evaluation



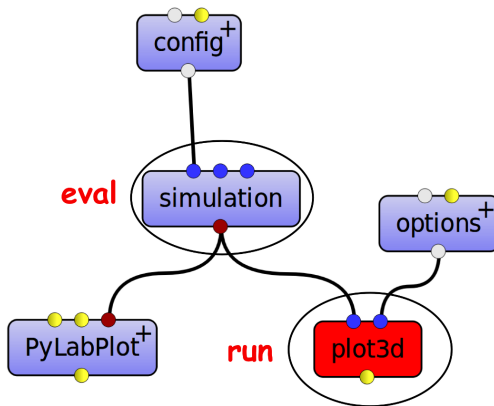
Dataflow Evaluation

Demand driven evaluation



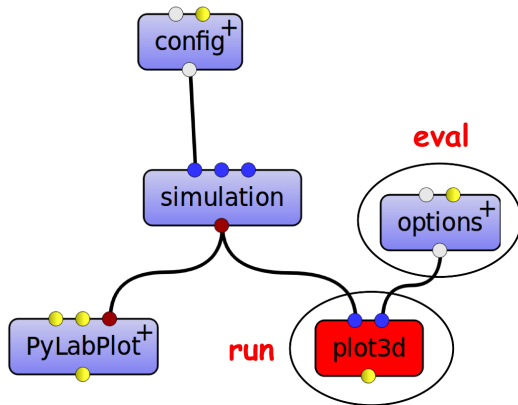
Dataflow Evaluation

Demand driven evaluation



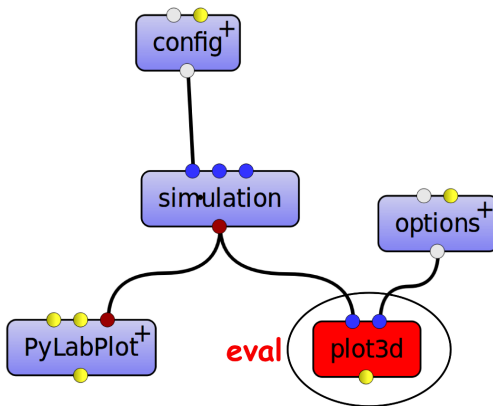
Dataflow Evaluation

Demand driven evaluation



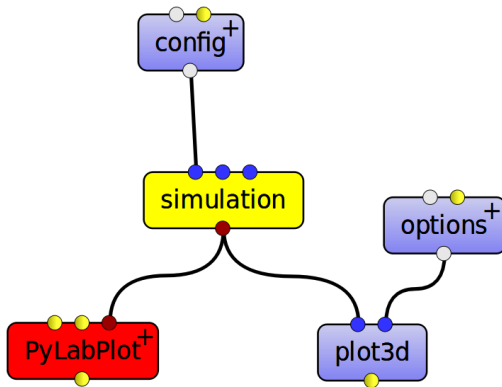
Dataflow Evaluation

Demand driven evaluation



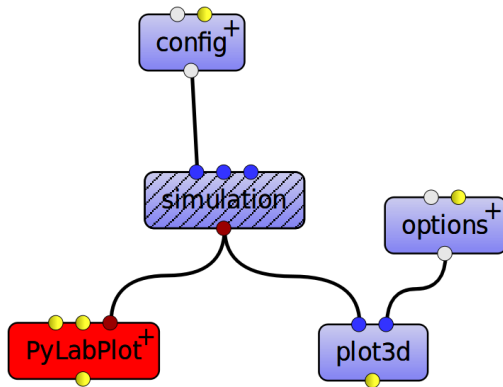
Dataflow Evaluation

Lazy node: re-evaluated only when one of its inputs has changed



Dataflow Evaluation

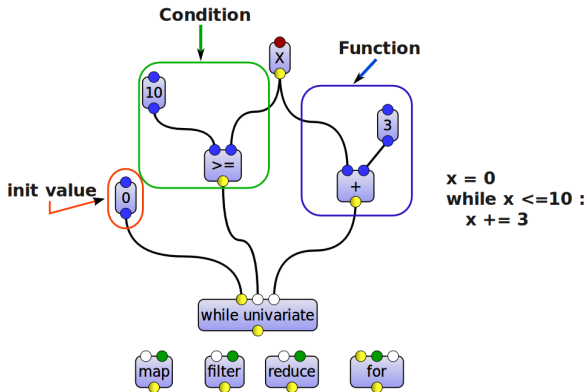
Block node: do not propagate the evaluation



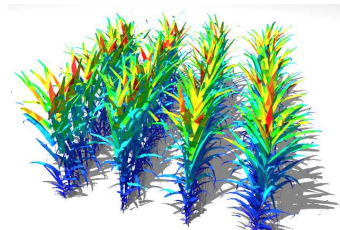
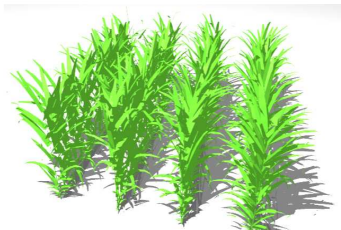
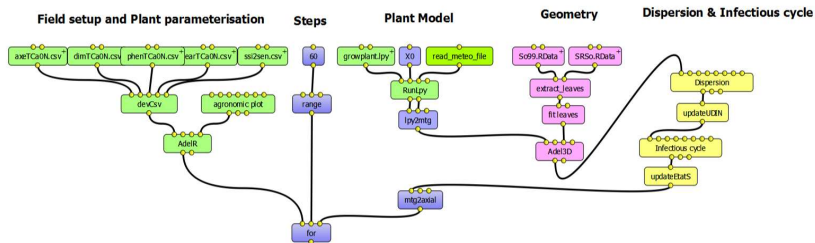
Dataflow Evaluation

Dataflow = no side effects + no cycle.

X node: transform a sub-dataflow into a **lambda** function



Example: simulation of plant/disease interaction



GraphEditor

Need for a **reusable python library** to **view and edit** (m)any **different graph types**, with support for **PyQt4**.

Concepts

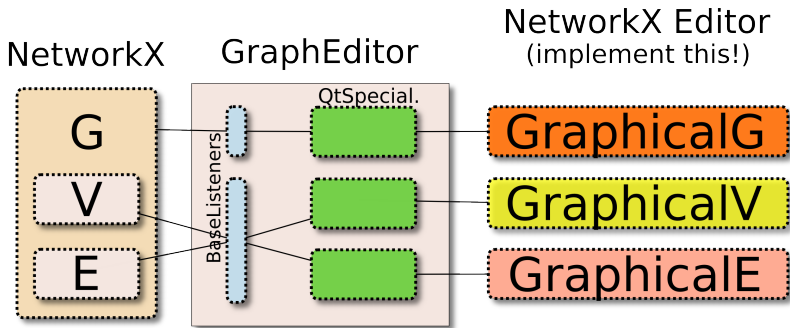


Trees, networks, dataflows (etc ...) boil down to $G = \{ V, E \}$ so
 $GraphicalG = \{ GraphicalV, GraphicalE \}$

GraphEditor

- Simplifies the implementation of custom graph editors
- Both aspect and interaction are customizable
- Has a PyQt4 implementation of the basic API

Example: Building an editor for NetworkX



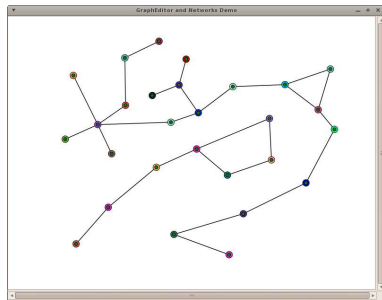
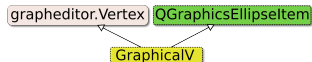
The user implements a strategy to view the data

Example: Building an editor for NetworkX

Implement a simple vertex representation

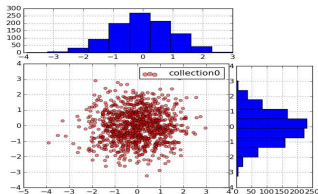
```
class GraphicalVertex(Vertex, QGraphicsEllipseItem):  
    def __init__(self, vertex, graph):  
        QGraphicsEllipseItem.__init__(self, 0, 0,  
                                       20, 20, None)  
        Vertex.__init__(self, vertex, graph,  
                        defaultCenterConnector=True)  
        self.initialise_from_model()
```

```
def initialise_from_model(self):  
    ''' Read the properties stored in the NetworkX  
    graph that can be useful for the view. '''  
  
    # Define the position of the vertex in the view  
    self.setPos(self.node()['position'])  
    # Define the color of the vertex in the view  
    color = self.node()['color']  
    self.setBrush(QBrush(color))
```

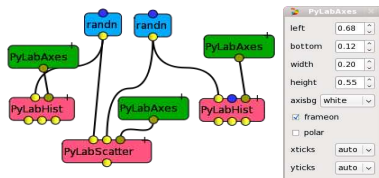


Libraries integration

- In VisuAlea, wrapping/integrating existing libraries into a GUI is made simple.
- PyLab/Matplotlib example: most of PyLab functionalities are available showing the feasibility of integrating complex standard libraries into VisuAlea.



PyLab output figure from the dataflow above.



Dataflow that combines scatter and histogram nodes applied on binormal random distribution using PyLab and Numpy functionalities.

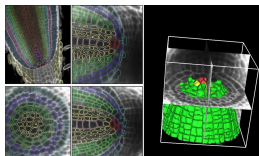
- Main advantage: existing options are now accessible as widgets.
- Numpy and Scipy components are integrated on demand.

Conclusions

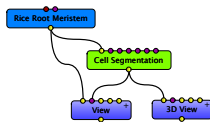
- OpenAlea provides a visual programming environment called VisuAlea
- VisuAlea allows to manage various scientific models in a GUI
 - Foster components/widgets reuse between labs
 - Ease communication
- Recent improvements:
 - Feedback loops using functional programming
 - Graph Editor
 - Many new packages from co-developers: (Biophysics models, image processing, ...)

Perspectives

- Integration of image processing algorithms and visualization tools
 - Registration
 - Fusion
 - Automated cell segmentation
 - Lineage computation
- Parallelization
- Reproducible dataflow simulation



Cells Segmentation and visualization in a rice root meristem (Fernandez *et al.*, Nature Methods, 2010)



Dataflow using a segmentation algorithm and visualization tools

Thank you!

<http://openalea.gforge.inria.fr>

