



Openalea - visual programming and component based software for plant modeling

Christophe Pradal, Samuel Dufour-Kowalski

► To cite this version:

Christophe Pradal, Samuel Dufour-Kowalski. Openalea - visual programming and component based software for plant modeling. EuroPython 2007, 2007, Vilnius, Lithuania. 2007. hal-00831823

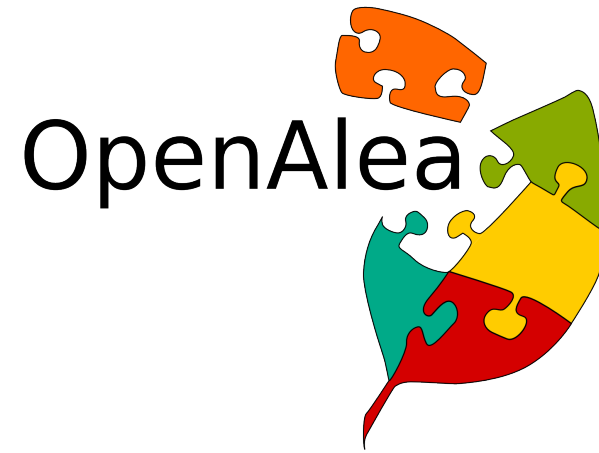
HAL Id: hal-00831823

<https://inria.hal.science/hal-00831823>

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.



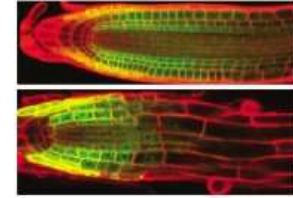
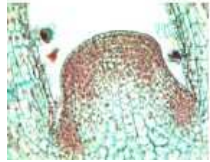
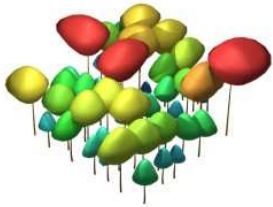
Visual Programming and Component based software for plant modeling

S. Dufour-Kowalski, C. Pradal

Plant Modeling



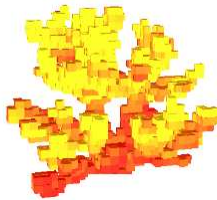
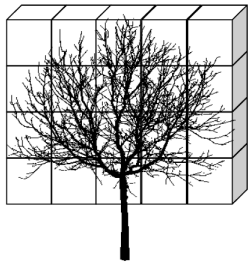
Forestry



Biological objects at
different scales

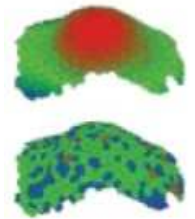
Measures

Mathematics



A pluri-disciplinary
research

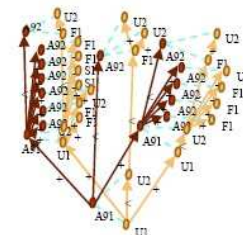
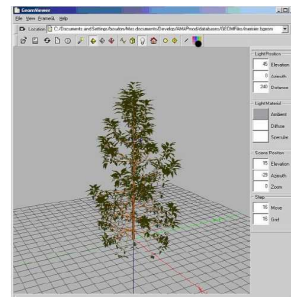
Simulation



Visualization

Analysis

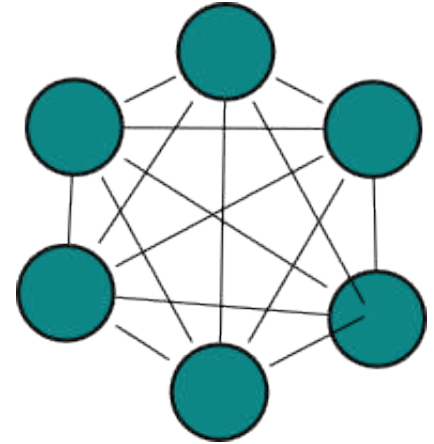
Biophysics



Problems



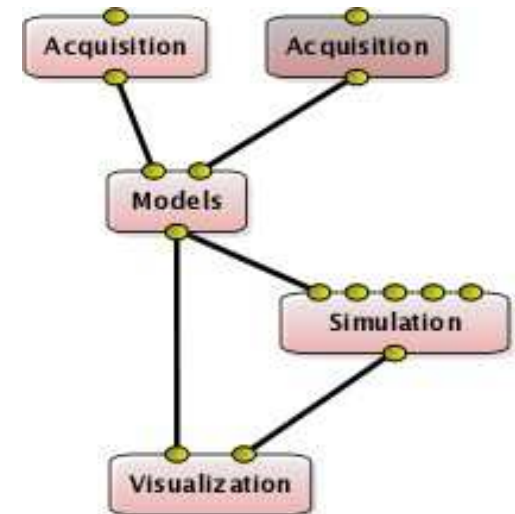
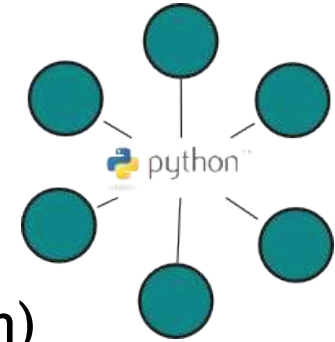
- Ad-hoc solutions for each problem
 - No reuse
- Models are difficult to interconnect
 - N^2 possibilities for N models
- Different types of actors in the community
 - Developers / Modelers / Users



Design principles



- Python as a glue
 - Common modeling language
 - Python simplicity + scientific libraries
 - Integrate existing softwares (C/C++, Fortran)
- Connectable components
 - Autonomous / Reusable
 - High level **data flow** approach
- Visual programming
 - Visual representation of a model
 - Less expressive, more intuitive

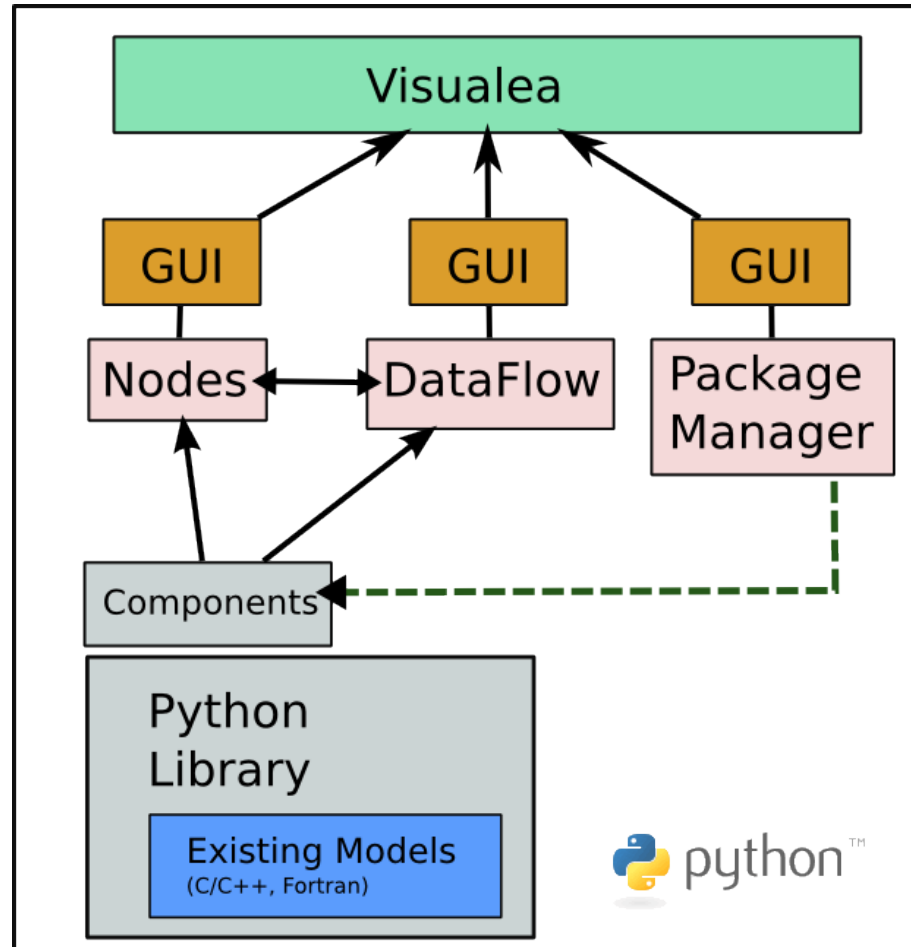


Related Work



- **Vision/Viper** (Sanner & al. 02)
 - Visual programming for bioinformatics
- **Orange** (Demsar & al., 04)
 - Visual programming for data mining
- **Enthougt – TraitsUI**
 - Automatic widget generation based on traits

Architecture



Visualea (PyQt4)



Package Manager

Data Pool

Code Editor

Dataflow

Widgets

Interpreter

Pydrop model (Bussière and al. FSPM 07)

The screenshot displays the Visualea software interface, which is a PyQt4-based application. The interface is divided into several panels:

- Package Manager:** Located on the left, it shows a list of packages and their categories. A yellow tooltip is visible over the 'Linear' package, stating: "Name: LRtoPlot, Category: Stat, Description: generate plotable object from linear regression".
- Data Pool:** Below the Package Manager, it lists available data objects: 'b3d (openalea.pydrop.drop_digitrec)' and 'plant (openalea.pydrop.drop_util.plan)'. An arrow points from the 'Data Pool' label to this section.
- Code Editor:** At the bottom left, it shows the Python code for the 'DropTri' widget. The code includes imports from 'openalea.pydrop.drop_sdk1' and 'openalea.pydrop.drop_util', and defines a 'def __call__(self, inputs):' function that handles plant creation and triangulation. An arrow points from the 'Code Editor' label to this section.
- Dataflow:** The central workspace shows a dataflow graph with nodes: 'ReadB3D', 'DropTri', 'DropPrint', and 'DropDisplay'. Arrows indicate the flow of data between these components. An arrow points from the 'Dataflow' label to this section.
- Widgets:** On the right, there are two widget windows: 'ReadB3D' (with a file selection field) and 'DropPrint' (with sliders for 'resolution_mm', 'rain_mm', 'stemflow_k', and 'stemflow_size'). An arrow points from the 'Widgets' label to these windows.
- Interpreter:** Below the code editor, a window shows the output of the Python interpreter, including the shell command 'python' and the execution of 'datapool['plant']'. An arrow points from the 'Interpreter' label to this window.
- DropDisplay:** At the bottom right, a 3D visualization of a plant model is shown on a grid. An arrow points from the 'DropDisplay' label to this window.

Concepts



-
- Node
 - Component Widget
 - Dataflow
 - Composite Node
 - Package Manager

Node / Component



Node

- A python callable
- Inputs/Outputs Ports (automatic or specified)

```
def linearmodel(x=0., a=0., b=0):  
    ''' return a*x+b '''  
    return a*x+b
```

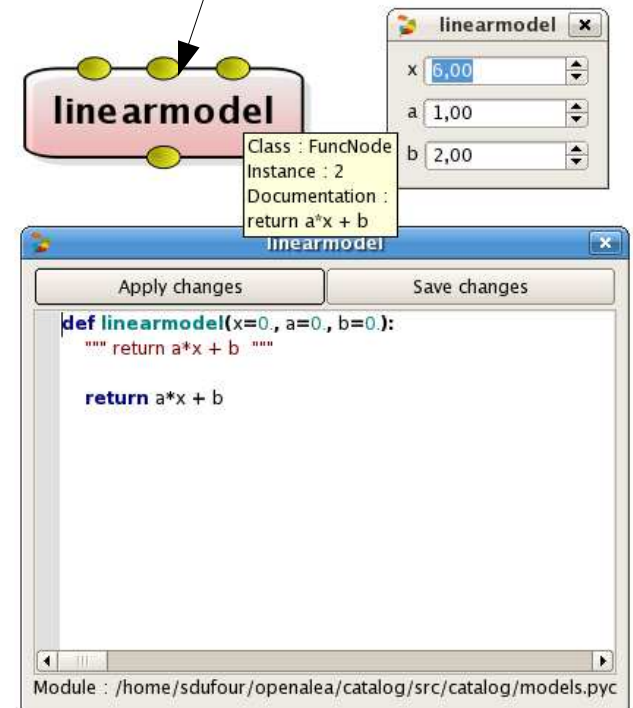
NodeFactory

- Component meta-information
- Lazy loading of modules
- Responsible to instantiate node

```
NodeFactory( name='linearmodel',  
             description='ax+b',  
             category='models',  
             nodemodule='simplemodel',  
             nodeclass='linearmodel',)
```

Port

- name = 'a'
- interface = IFloat

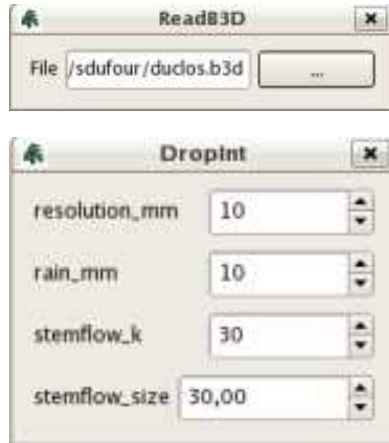


Component Widgets



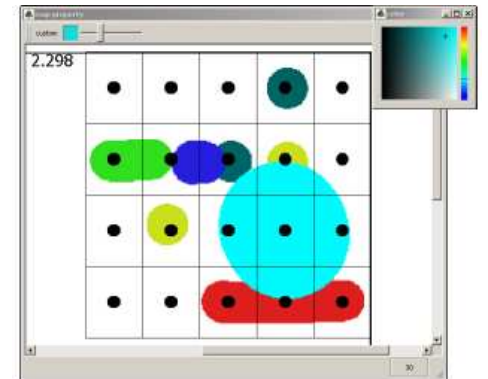
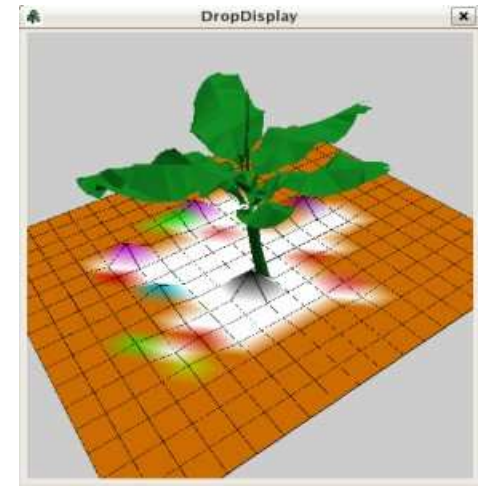
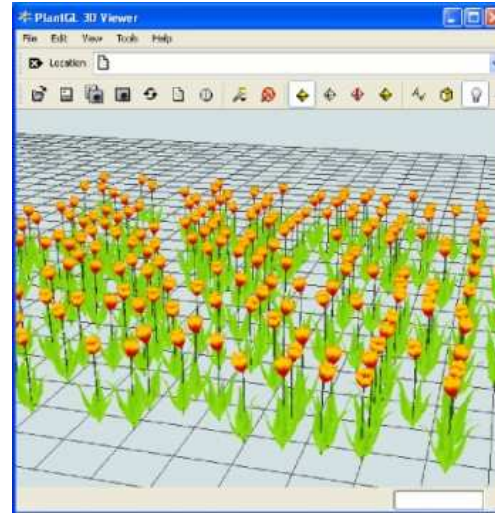
Automatically generated

Use Input port Interfaces :
IInt, IFloat, IString, IFileName,
IColor, IList, IDict...



OR

Particular Widgets
(Viewer, Plots...)



Dataflow



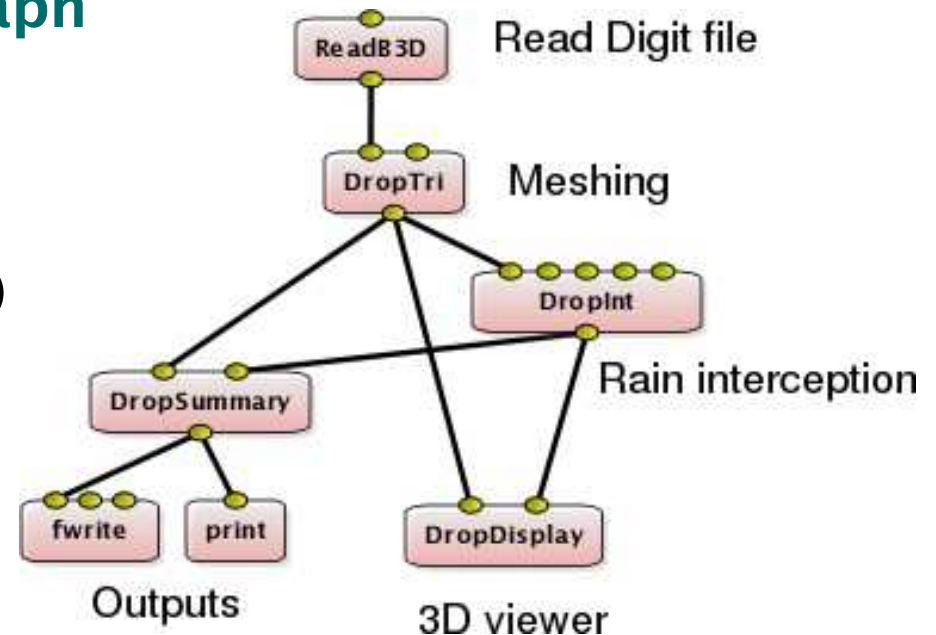
Connect Nodes in a directed graph

Evaluation algorithm is modular

- **Functional** (deterministic)
- **With tokens** (non deterministic)

Optional features

- **multi-inputs** (list creation)
- **Priority management**
- **Lazy evaluation**
- **Object copy**
- ...



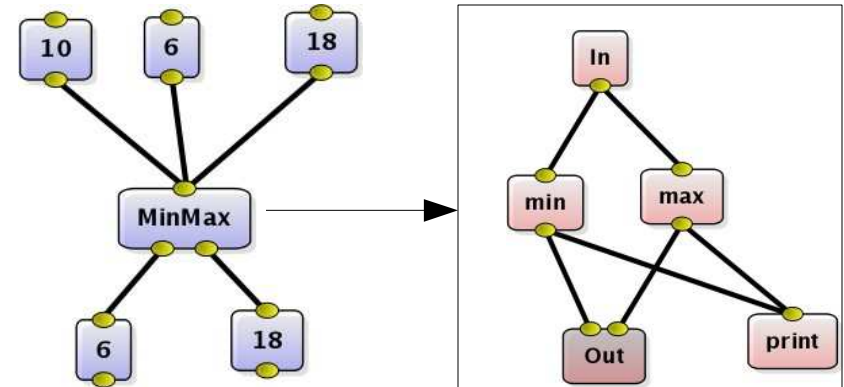
User documentation with graphical text annotations

Composite Nodes



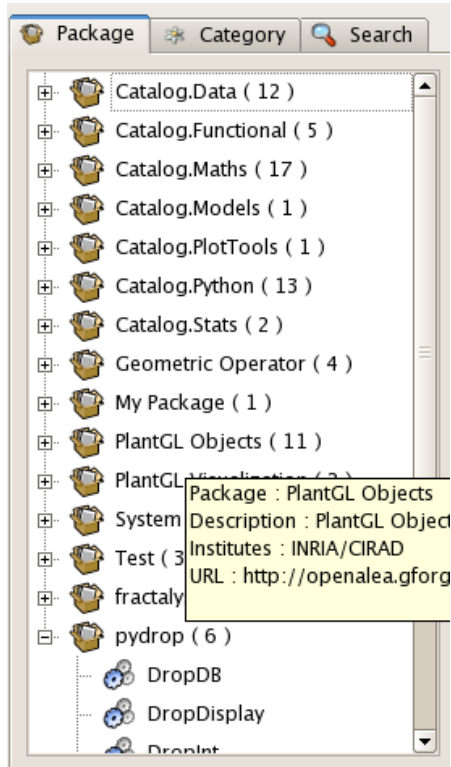
Composite/Macro Nodes encapsulate a dataflow.

- Manage complexity
- Exported as Python code.
- Composite Node
 - Composite Widget



```
CompositeNodeFactory(  
    name='MinMax',  
    description='Return min and max of a list',  
    category='Maths', doc='...',  
    inputs=[{'interface': ISeq, 'name': 'IN1'}],  
    outputs=[{'interface': None, 'name': 'OUT1'},  
              {'interface': None, 'name': 'OUT2'}],  
    elt_factory={ 2: ('catalog.math', 'max'),  
                  3: ('catalog.math', 'min'),  
                  4: ('catalog.python', 'print') },  
    elt_connections={0: (3, 0, 4, 0), 1: ('__in__', 0, 2, 0),  
                     2: (2, 0, '__out__', 1), 3: ('__in__', 0, 3, 0),  
                     4: (2, 0, 4, 0), 5: (3, 0, '__out__', 0)}, )
```

Package Manager



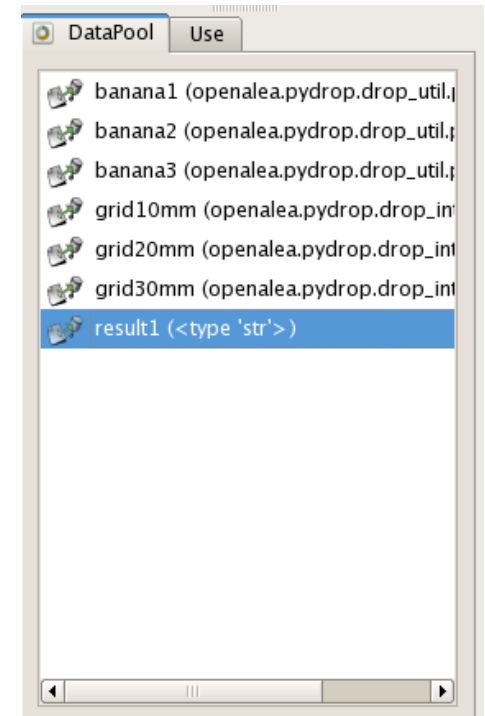
- Nodes are grouped by
 - **Package** (authors, license...),
 - **Category**
- Load packages on demand.
- Research and execute `*wralea.py` files found on the system. (*Future use of setuptools entry_points*).
- Search nodes by name, description, category, ...
- Instantiation with Drag-and-Drop.

All python packages can declare OpenAlea components.

Data Management



- Data Pool
 - Container of global instances
 - Drag and Drop operations
- Data conversion between nodes
 - Based on interfaces/adapters
- Data Persistence
 - Pickling



Python scripting



- Visual programming for high level modeling
 - Do not replace python scripting
- Low-Level interaction
 - Interpreter : Access directly to python object
 - On the fly *code edition* and *node creation*
 - Completion & introspection (QScintilla)

```
The shell running Python 2.4.3 (#1, Oct 23 2006, 14:19:47)
[GCC 4.1.1 20060525 (Red Hat 4.1.1-1)] on linux2.
Type "copyright", "credits" or "license" for more information on Python.
session = Session instance.
pmanager = PackageManager instance.
datapool = DataPool instance.

>>> datapool["result1"].
capitalize
center
count
decode
encode
```

```
fwrite
Apply changes Save changes

def py_fwrite(x="", filename="", mode="w"):
    """ Write to a file """

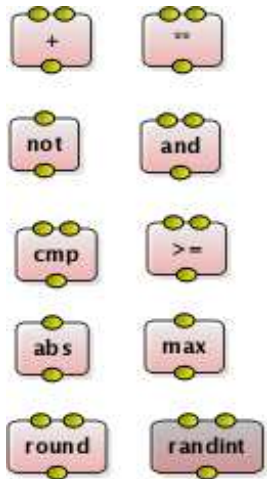
    f = open(filename, mode)
    f.write(x)
    f.close()

Module : /home/sdufour/openalea/catalog/src/catalog/python.pyc
```

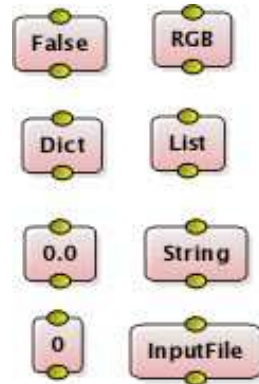
OpenAlea.Catalog



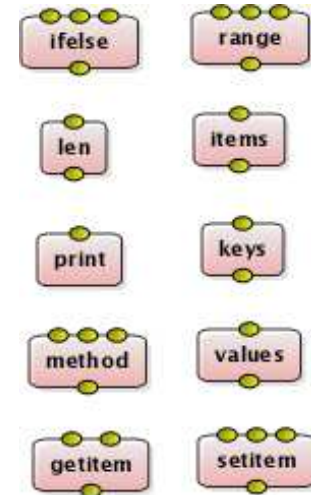
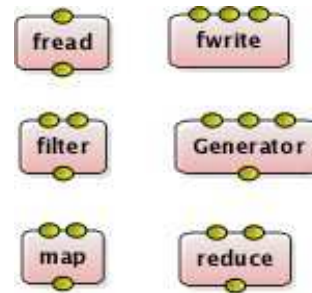
Math



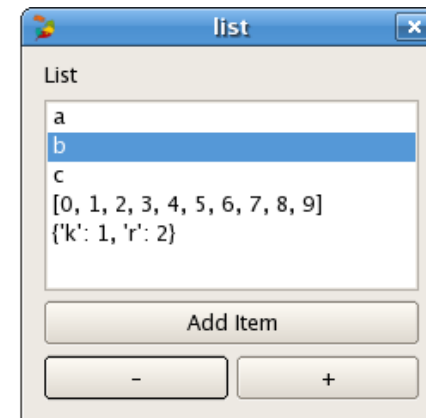
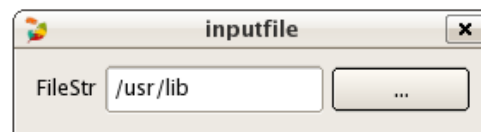
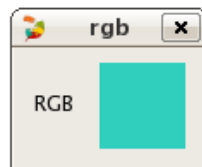
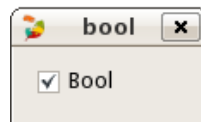
Type



Function



Widget



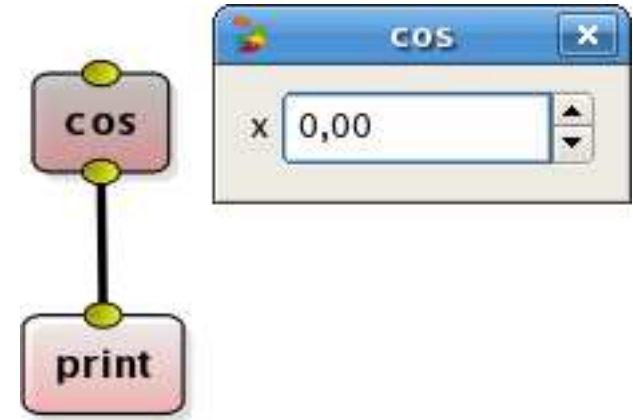
How to wrap a module



```
math.py :  
    def cos(x:float): return float
```

wrlea.py

```
from openalea.core import *  
  
def register_package(pkgmanager):  
  
    pkg = Package('my_pkg', {'version':'0.0.1',  
                             'license':'Python'}, )  
  
    nf = Factory(  
        name="cos",  
        inputs=[{'name':'x', 'interface':IFloat},],  
        outputs=[{'name':'y', 'interface':IFloat},],  
        nodemodule="math",  
        nodeclass="cos",)  
  
    pkg.add_factory(nf)
```



Deployment : Building



- OpenAlea.SConsX
 - Simplify the build of complex multi-platform packages
 - Hide the complexity of the build system
 - Set options/flags for different tools and compilers
 - Add knowledge about existing tools (system dependent)

```
ALEAConfig(name, ['boost_python', 'alea', 'qt4', 'opengl'])  
src = ALEAGlob('*.cpp')  
inc = ALEAGlob('*.h')
```

```
ALEAInclude('mylib', inc)  
ALEALibrary('mylib', src)  
ALEAWrapper('mywrapper', src)  
ALEAProgram('myprog', src)
```

Deployment : Installing



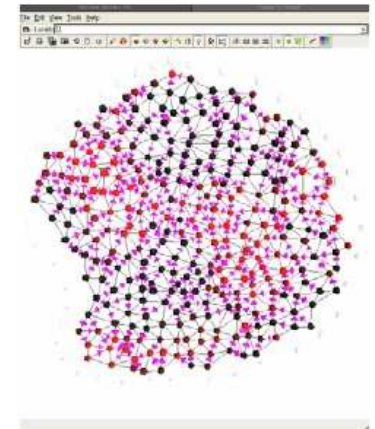
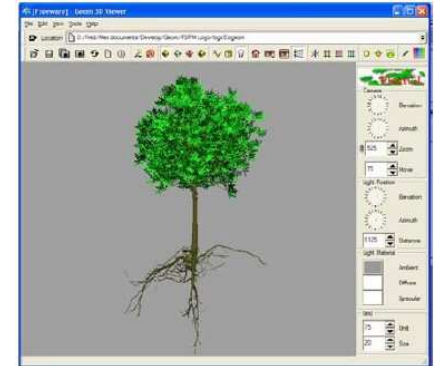
- OpenAlea.DistX
 - Install shared dynamic libraries, data and application (shortcuts, environment variables)
 - Use SConsX as a build system
 - Extend Distutils (Setuptools migration ?)

```
setup(  
    name=name,  
    version=version,  
    ...  
    scons_scripts=['SConstruct'],  
    scons_parameters=["build", "build_prefix="+build_prefix],  
    external_data={pj('test', name) : 'test',  
                   pj('lib'): pj(build_prefix, 'lib'), ... },  
    set_win_var=['PATH='+ ... ],  
    set_lsb_var=['LD_LIBRARY_PATH='+ ... ],  
    win_shortcuts=[...], freedesk_shortcuts=[...]  
    ... )
```

Components



- Analysis of plant architecture
 - VPlants (Godin, Guédon and al.)
- Geometric library and 3D viewer
 - PlantGL (Boudon, Pradal and al)
- Merristem simulation (Barbier de Reuille and al.)
- Radiation absorption
 - RATP (Sinoquet and al.)
 - Fractalysis (Da Silva and al.)
- Rain interception : PyDrop (Bussi re, Dufour, and al.)
- ...



Perspectives



- Future work to address:
 - Parallelization of execution
 - Simulation issues (what is the best approach ?)
 - Installation & dependencies management with shared lib
 - Node creation wizard
 - New models and tools integration
- Application to other domains
 - Computer graphics
 - ... ?

Conclusions



-
- OpenAlea is an **open source** project.
 - It aims to share softwares inside and outside the plant modeling community.
 - Improve accessibility for biologists (python and visual programming).
 - OpenAlea = set of libraries and components.
 - OpenAlea modules are being integrated.

OpenAlea on the web

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

