Scientific workflow for reusing plant/FSPM models

Jérôme Chopard, Christophe Pradal, Daniel Barbeau, Thomas Cokelaer, Christophe Godin

Scientific workflow for reusing plant/FSPM models

J. Chopard [a,b,c], C. Pradal [a], D. Barbeau [a], T. Cokelaer [a] and C. Godin [a]

[a] *VirtualPlants team, INRIA*
[b] *Center of Excellence for Climate Change Woodland and Forest Health, University of Western Australia*
[c] *School of Plant Biology and UWA Institute of Agriculture, The University of Western Australia*
Email: jerome.chopard@uwa.edu.au

Abstract: For many years a large collection of models have been developed to describe plants (e.g., trees, crops). However, few of them can be reused directly to be integrated into more complex models or combined with other plant models to answer scientific questions. This leads to a loss of time spent in re-implementing published models before starting new projects. To tackle this challenge, a platform called OpenAlea has been developed over the last 10 years with the aim of sharing models and tools among the OpenAlea community.

This platform uses the well-known Python programming language to link heterogeneous components/models together. It performs the evaluation of complex models expressed as a dataflow where each node is a different piece of the model. A visual front-end, VisuAlea, allows users to assemble different components together to answer practical questions without writing a single line of code.

In this article, we propose to demonstrate the functioning of the platform with a single model/component. As a first step, this component will be connected to statistical components to demonstrate the ability of the platform to share common analysis and plotting functions. In a second step, the same component, instead of taking fixed parameters, will be attached to external models that will provide estimated values for these parameters, showing the ability of the platform to connect models developed by different teams. The third step will be to connect a sensitivity analysis component to this system. Finally, we will propose a different evaluation of the component as part of a temporal simulation to demonstrate how a simple static model can be reused to construct complex plant growth models.

Keywords: *Dataflow, Python programming language, functional structural plant modelling*

1. INTRODUCTION

Models are increasingly being used in the plant research community to understand plant growth and physiology. Most of these rely on the same components to describe plant functioning. However, few of them can be reused directly either by integrating or combining them with other plant function models to answer scientific questions. Platforms have been proposed in different fields (e.g. biochemistry with the Vision project (Sanner, 2005), visual data mining with the Orange project (Demsar et al., 2004)) with these goals in mind: i) ease the creation of new models by reusing elements of old models and ii) simplify the combination of these elements. In the plant modelling community, two large projects, L-studio (Federl et al., 1999) and GroImp (Hemmerling et al., 2008), focusing on computer graphics, offer an integrated environment to develop new models. These approaches force the user to develop new components in their specific language in order to plug them into the simulation kernel.

The OpenAlea platform (Cokelaer et al. 2009, Pradal et al. 2010, Cokelaer et al. 2010, Fournier et al. 2010) was created to allow users to interact easily with models developed by the community. OpenAlea uses the well-known Python language to link heterogeneous components together and provide a solution to solve the four main problems that usually prevent people using the models from other teams:

- i) Applications are a combination of components designed by modellers to be used by different users. Each component can be rearranged by users to suit the required degree of complexity of a particular problem. Components can be implemented in different languages (e.g. C, C++, Java, R) and reused in the platform by providing a Python interface.

- ii) Common resources are easily shared in the platform and multiple scientific libraries are already available to read, write and plot data in 2D and 3D or to perform statistical analyses.

- iii) Documentation is strongly encouraged among the community. This documentation is part of the code of components and can be read directly or translated into webpages to provide user and developer documentations.

- iv) From the developer point of view, tools to deploy applications on multi-platforms are available. From the user point of view, the complexity of implementation is hidden: a common graphical user interface, VisuAlea, allows components to be combined together in a workflow, which describe the different steps of evaluation, without a single line of code.

In this article we intend to demonstrate how a single component designed for a particular task can be reused directly as part of two types of models: a simple decision model and a functional-structural plant model.

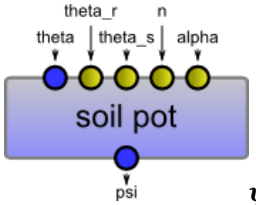
2. PLATFORM DESCRIPTION

2.1. Components definition

An application is a combination of components designed by modellers to be used by users. Each component can be rearranged by users to suit the required degree of complexity of a particular problem. Components can be implemented in different languages (e.g. C, C++, Java, R) and reused in the platform thanks to a Python interface.

Each component is fully characterized by the parameters it takes in inputs and the values it returns on completion of its task (outputs). The actual complexity of the task performed by a component is not bounded and can vary from single mathematical operations (e.g., a '+' node takes two parameters and returns a single value or 'soil potential' node in figure 1) to more integrated/computationally intensive/complex models (an FSPM node may take a single 'age' parameter and return the structure of the plant). It does not have to be deterministic and the same inputs can produce a different output each time the component is evaluated.

While Python does not require typed arguments, interfaces have been proposed to characterize inputs and outputs to simplify the connection between heterogeneous components. The most common (number, date, colour ...) are already registered to increase the standardization between models but the user can easily declare their own interface that they want to expose to the community(e.g. 'tree' to represent the topological structure of a plant or 'grid' to represent a voxelisation of space).



$$\psi = -\frac{1}{\alpha}\sqrt[n]{\left(\frac{\theta_s-\theta_r}{\theta-\theta_r}\right)^{1/m}-1} \text{ with } m = 1-\frac{2}{n}$$

Figure 1. Simple 'soil water potential' component as defined in Smettem et al. (1996).

2.2. Dataflow description – connection between components

In OpenAlea, complex models are built as an assemblage of multiple components performing atomic tasks. This workflow explicit the steps needed to compute the final result from the evaluation of each component. It uses a dataflow to keep track of the connections between each component. When one output of a component A is connected to an input of another component B, the value computed by A is used as an input by B. Hence, the dataflow is an acyclic graph where components play the part of nodes and each edge (line between two nodes) represents a transfer of information between an output of one component and the input of another. Different cases occur depending on how nodes are connected (see figure 2):

- Nodes whose entries are not connected to anything must take default values for these inputs;
- Multiple inputs of nodes N_1, ..., N_n connected from the same output of node A share a reference on the object actually produced by A. This can lead to unwanted side effects since evaluation order is not guaranteed between nodes connected to the same input (i.e. if node N_1 modifies the object, the object seen by node N_i is not the one produced by A but the one already modified).
- If multiple outputs from nodes N_1, …, N_n are connected to the same input of node A, the actual value input in A is the list of the n values ordered according to the spatial position of the nodes (V_i will be before V_j in the list if node N_i is horizontally on the left of node N_j).

From an evaluation point of view, a dataflow is also a component. Its inputs correspond to the list of non-connected inputs and its output corresponds to the list of non-connected outputs. Hence, an assemblage of simple components to perform a task can be reused as a component in a more complex model. This multi-scale approach simplifies the design of complex models by grouping components together and reducing the total number of components at each scale (e.g., a complex model of plant growth can be decomposed into three high-levels components soil/plant/atmosphere. Each high-level component is also an assemblage of smaller scale components. E.g. the 'plant' growth component can incorporate photosynthesis/water transport/carbon storage … and so forth).

The inside of a component is opaque and the actual code is not required for evaluation (see below). That way, a component can easily be replaced by another component that exposes the same inputs/outputs. For instance in the previous example, a first model may be made with the three components soil/plant/atmosphere where each component is a single equation/function. Later on, the user can choose to replace the plant component by a more complex dataflow that exposes the same inputs/outputs but provides a more detailed modeling.

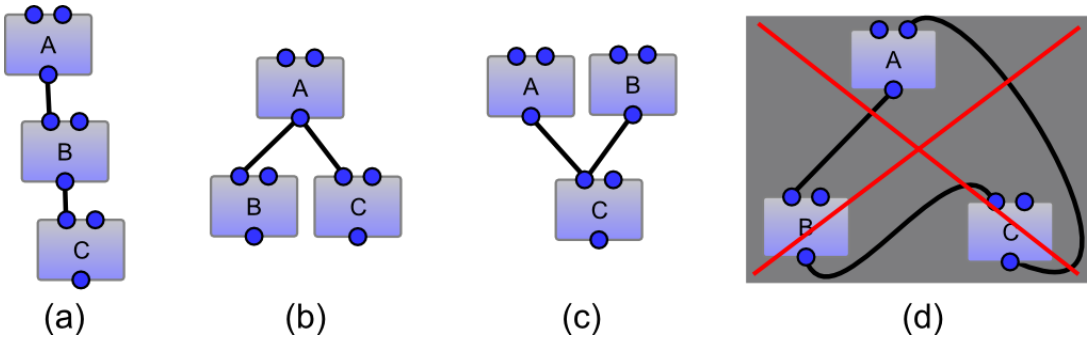


Figure 2. Dataflow examples: a) simple linear dataflow, b) multiple inputs on one output, c) multiple outputs in one input and d) circular connections (not a proper dataflow)

2.3. Evaluation of a dataflow

A dataflow represents the structure of a model. It makes explicit the connections between the different components of the model. However, the actual evaluation of the dataflow to produce the result of the model is not constrained. Its evaluation strongly depends on the semantics associated with components. It is important to disaggregate models into components with the appropriate granularity. That way, components may be reused in other projects to answer different scientific questions.

The evaluation offered by the platform is a two pass process. In the first pass, the dataflow is walked from the bottom leaves (nodes without outputs or whose outputs are not connected) to the top nodes (nodes without inputs or whose inputs are not connected). The first pass orders the evaluation of all the components to ensure that a component is evaluated only when all the components connected to its inputs have been

evaluated. The second pass actually performs the evaluation of each node. For each node, the values of its inputs are retrieved from the outputs of the nodes connected to it. Then, the component performs its task and the result is stored in the output of the node.

This simple raw evaluation can be improved by introducing the concept of laziness in the network. With this approach, in the second pass of the algorithm, a node is evaluated only if it is tagged as non-lazy or if the values of its inputs have changed since the last evaluation. This approach saves resources by computing only the parts of the dataflow that need to be recomputed between two evaluations and provide a framework to perform temporal simulations through scheduling. If a node A implements a task and is tagged non-lazy once every two evaluations of the dataflow whereas a node B implements a task permanently non-lazy, continuous evaluations of the dataflow will result in task B being performed twice as much as task A (see figure 3).

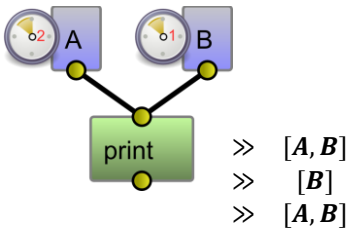


Figure 3. Scheduling, node A is tagged non-lazy every 2 evaluations whereas node B is tagged non lazy every evaluation. The result of continuous evaluation of the dataflow is displayed on the right.

2.4. VisuAlea

VisuAlea provides a visual programming environment allowing users to dynamically combine existing and independent pieces of software into a customizable dataflow. This environment offers several interesting points for managing scientific workflows. First, users have access to the program state at run-time and thus can introspect, modify, document and debug their workflows interactively. Second, users can build graphically complex workflows by interactively assembling OpenAlea components. Finally, the visualisation of the workflow's structure eases the communication between different users.

The definition of types for inputs and outputs allows the system to automatically propose a graphical user interface for each component. Alternatively, the developer that creates a new component can also design a more suitable GUI and attach it to the node, overriding the default behaviour.

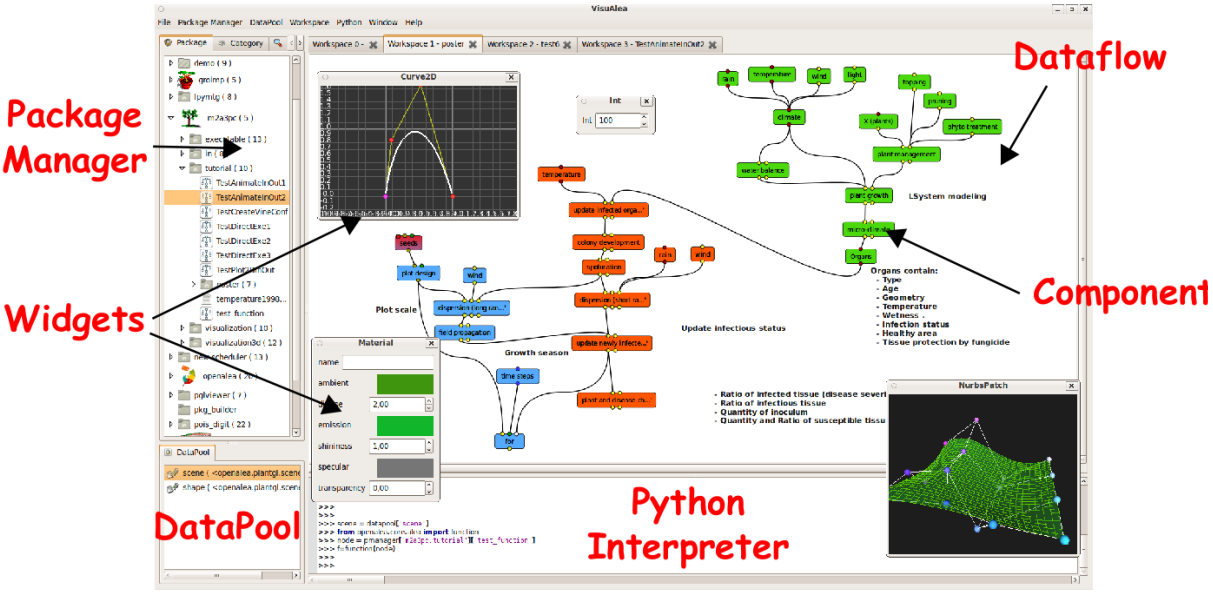


Figure 4. Screenshot of the main VisuAlea user interface used to connect components together and evaluate the scientific workflow. The package manager lists all the components available. The datapool keeps track of state variables. Widgets are attached to the dataflow to display results or query user inputs. A Python interpreter provides a way to introspect the current workflow.

3. EXAMPLE OF USE (AND REUSE)

In this section we will make use of a single component presented above (figure 1) that embeds a very simple mathematical function relating the potential of water in the soil to its water content as defined in Smettem et al. (1996). Hence, the five inputs of the component are:

- a number (theta) that describes volumetric water content in the soil;
- four numbers (theta_s, theta_r, n and alpha) that describe the structure of soil pores.

The last four parameters that describe the structure of the soil are not expected to vary through time. The unique output is a number that represents the water potential for this specific soil and water content.

3.1. Simple straight evaluation

Evaluating an atomic dataflow with only this node will make use of the default parameters stored in the node to estimate the value of water potential. This value is readily accessible by connecting a 'print' node to the output of the node.

A more interesting behavior is obtained by connecting a 'range' node to the first input of our component. Since this node produces a new value in the given range each evaluation, we can readily compute the set of values taken by the water potential in the soil over the range of expected conditions. These values, stacked together to produce an array, are easily displayed using one of the standard graphical libraries.

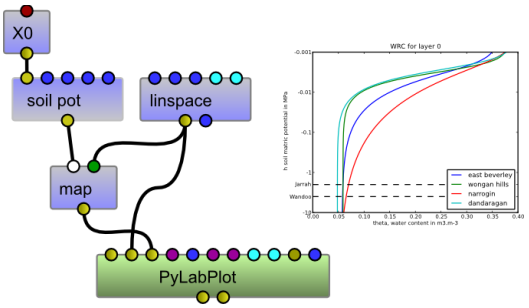


Figure 5. This workflow evaluates the soil water potential (soil pot node) over a range of linearly spaced parameters values (linspace node), stacks the values together (map node) and plots the resulting curve (PyLabPlot node). Inputs and outputs are color coded according to the type of interface they declare. Hence, most of the time an input and an output connected together will have the same color.

This node can also be part of a more complex decision model that determines a water management decision. A first model provides the soil water content throughout time whereas the decision model is a simple switch that decides to water the plants as soon as the soil water potential drops below a given threshold. Hence, our small component is used as an interface to match the output of the earth model with the input of the decision model.

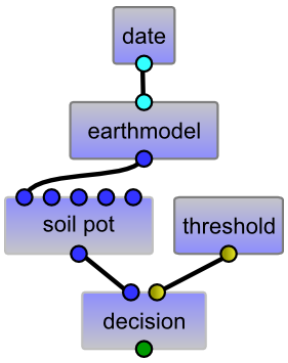


Figure 6. 'Soil pot' node is used as an interface to adapt the output of the earth model to the input of the decision model

3.2. Reuse in a dynamic FSPM

The scheduling evaluation of the dataflow is another great opportunity to reuse this simple component in a more complex FSPM of plant water use. Traditionally that kind of simulation implies a loop, where the state of the plant at time t is used to compute the state of the plant at time t+1. Water content in the soil and the atmospheric conditions drive water uptake by the plant which in turn modifies water content in the soil. However, since a dataflow is acyclic, we cannot connect the output of a node to the input of another node which is "above" it (see figure 2). One solution is to identify the state variable of the problem and store them independently from the dataflow in a datapool. Hence, a node that modifies the value of a state variable can store it back and this new value will be used for the next evaluation of the dataflow (see the time part of the example below for a small such loop, figure 7).

Our example plant model depends on two state variables:

- The actual time is provided by a node that increases the current time by a fixed delay at each evaluation of the dataflow;
- The water content in the soil is modified by the soil node internally.

Other components of the models use these state variables to compute physiological processes.

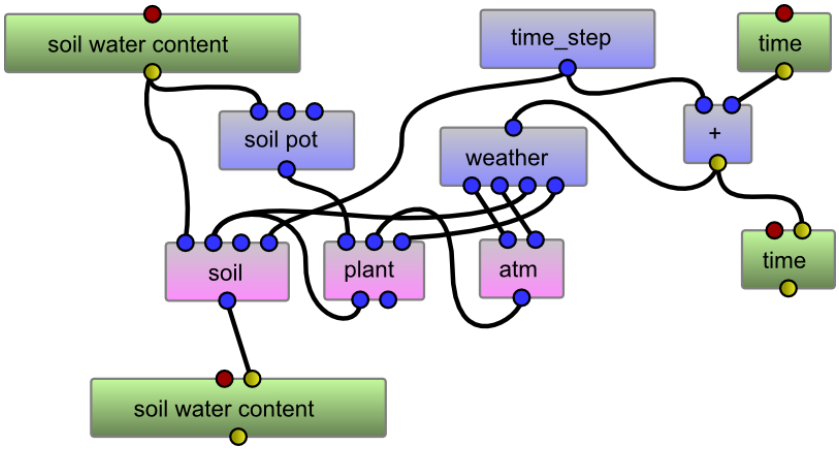


Figure 7. Dynamic model of water transfer in soil, plant and atmosphere. Green nodes stand for the two state variables of the simulation: time and soil water content. Pink nodes in the center are the three main processes.

4. CONCLUSIONS

Throughout this article we have described the method for developing new models in a modular way by assembling components together. The advantages of this approach are:

- It reduces the development time by reusing components developed by others;
- It offers an incremental approach in developing the complexity of the model: simple components can be replaced by more complex ones;
- It explicit the structure of the model. Each task is a separate component that can be developed and tested individually. When assembled together afterward, connections between tasks explicit visually their dependency;

The OpenAlea platform has been created with these goals in mind and offers a scientific environment to exchange pieces of models. This exchange is facilitated by the different levels of accessibility to the models:

- developers can develop new components in their favorite language;
- advanced users can use the Python language to write scripts to interact with models in new ways;
- final users can use VisuAlea to connect components through a graphical user interface without writing a single line of code.

The platform addresses the traditional difficulties associated with computer programming (compilation, installation...) on multiple environments. The users can then focus on implementing their models and share them with the community.

REFERENCES

Cokelaer T., Pradal C. and Fournier C. (2009). Plant modeling with Python components in OpenAlea. Proceedings of Euroscipy 2009.

Cokelaer T., Pradal C. and Godin C. (2010). Introduction to OpenAlea, a platform for plant modeling. Proceedings of 28th international horticultural congress.

Demsar J., Zupan B., Leban G. and Curk T. (2004). Orange: from experimental machine learning to interactive data mining. Book chapter in *Knowledge discovery and databases*, 3202, 537-539

Federl P. and Prusinkiewicz P. (1999). Virtual laboratory: an interactive software environment for computer graphics. *Proceedings of computer graphics international*, 93-100

Fournier C., Pradal C., Louarn G., Combes E., Soulie J-C., Luquet D., Boudon F. and Chelle M. (2010). Building modular FSPM under OpenAlea: concepts and applications. Proceedings of 6th international workshop on Plant Structural-Functional Models, 109-112

Hemmerling R., Kniemeyer O., Lanwert D., Kurth W. and Buck-Sorlin G. (2008). The rule-based language XL and the modeling environment GroIMP illustrated with simulated tree competition. *Functional plant biology*, 35, 739-750

Pradal C., Cokelaer T., Barbeau D., Moscardi E., Chopard J. and Godin C. (2010). Modeling strategies using visual programming in OpenAlea. Proceedings of 6th international workshop on Functional-Structural Plant Models, 101-103

Sanner M.F. (2005). A component-based software environment for visualizing large macromolecular assemblies. *Structure*, 13(3), 447-462

Smettem K. and Gregory P. (1996). The relation between soil water retention and particle size distribution parameters for some predominantly sandy Western Australian soils. *Australian Journal of Soil Research*, 34(5), 695