The architecture of OpenAlea: A visual programming and component based software for plant modeling

Christophe Pradal[1], Samuel Dufour-Kowalski[2], Frédéric Boudon[1], Nicolas Dones[3]
[1] CIRAD, Avenue Agropolis, 34398 Montpellier Cedex 5, France
[2] INRIA, 2004 route des lucioles BP 93, 06902 Sophia Antipolis, France
[3] INRA, Site de Crouël, 234 avenue du Brézet, 63100 Clermont-Ferrand, France

Keywords: plant modeling, software architecture, interactive modeling, dataflow

Introduction

The FSPM community develops models to understand the biological processes involved in the function and growth of plants. Researchers in botany, ecophysiology, forestry, horticulture, applied mathematics and computer science have developed several models and software tools. Due to the different constraints and background of the teams, the available models have been developed in different programming languages on different operating systems with the goal of answering specific biological questions at a given scale. They are often developed as "monolithic" programs which generally lack of interoperability. In this work, we present the software architecture of OpenAlea, a flexible component-based framework designed to facilitate the integration and interoperability of heterogeneous models and techniques from different scientific disciplines. OpenAlea is developed in Python, a high-level, object-oriented, interpreted language. The OpenAlea architecture consists of: (a) a set of tools to integrate heterogeneous models implemented in various languages and on different platforms; (b) a component framework that allows for the dynamic management and composition of software components; and (c) a graphical modeling environment for enhancing the use of complex models and for rapid prototyping. To illustrate the integration of a complex component and its use through the graphical modeling environment, the PlantGL library for 3D plant modeling and visualization is presented.

Related work

In the plant modeling community, the idea of using a modular platform with components can be traced in the Virtual Laboratory (Prusinkiewicz et al., 1990). This interactive environment consists of experimental units called objects that encompass data files, and programs that operate on these objects. An inheritance mechanism allows refining objects in an object-oriented file system. However, stand-alone programs have low interoperability and the shell language used in this case to combine them has limited expressivity that makes difficult specification of complex control flows.

Alternatively, XFrog, a computer graphics software (Lintermann et al., 1999), provides an intuitive visual environment to design plant models with predefined generative components. Unfortunately, the system has limited extensibility.

OpenAlea was also inspired from different visual programming environments developed for other scientific topics such as Vision (Sanner, 2002) in bioinformatics or Orange (Demsar et al., 2004) in data-mining. However, we introduce in our systems different architecture principles such as the separation between functionalities and graphical interfaces.

Language integration

OpenAlea is a Python-based framework for the integration and the interoperability of heterogeneous components. Python is used as a glue as well as a flexible language for interactive scripting and rapid development of applications.

In our "language-centric" approach, existing C, C++ or Fortran libraries are written as extension to the Python language. Standard wrapping tools, such as Boost.Python, Swig, and f2py, are used to

support the integration process of such heterogeneous components. One of the key objectives of OpenAlea is to be multi-platform. While Python components are platform independent, others have to be built and installed on the target platforms, which may be a rather complex task. To ease the integration process, we have developed various tools such as SConsX and DistX. SConsX is an extension package of SCons (Knight, 2005). It simplifies the building of complex platform dependent packages by supporting different types of compilers (i.e. gcc, MinGW, Visual C++), and the different steps involved in compiling for Windows and GNU/Linux. DistX extends the standard Distutils Python library to facilitate package installation in the OpenAlea framework. Windows and RPM installer as well as source distribution can be automatically created.

Integrating models in a common python framework enhance usability by providing a unique modeling language to heterogeneous software. It allows to extend, compare and reuse existing functionalities. To improve software quality and ease maintenance, the component framework follows separation of concerns (e.g. data, algorithms, data-structure, GUI), by having independent modules dedicated to shared data-structure, computational task, graphical representation, etc.

A component framework

The core of OpenAlea is a *component framework* that allows users to dynamically reuse and combine existing and independent pieces of software into customized work-flows according to their specific needs. This type of framework emphasizes decomposition of application into separated and independent functional subsystems. Communication between components is achieved through their explicit interfaces (Szyperski, 2002).

The OpenAlea framework proposes an implementation of these principles. The software architecture is organized according to several concepts: (a) a *node* represents a software unit and is also named logical component. It is a functor object (i.e. an operator or a function) which provides a certain type of service. It can exchange data through its input and output ports. (b) A *composite-node* is a node that encapsulates other nodes defining a hierarchy of components. Node composition allows creating extended and reusable subsystem. (c) A *dataflow* (Johnston et al., 2004) is a graph composed of nodes connected by edges representing the flow of data from one node to the next. It defines a high level functional process well suited for coarse grain computation and close to natural thinking.

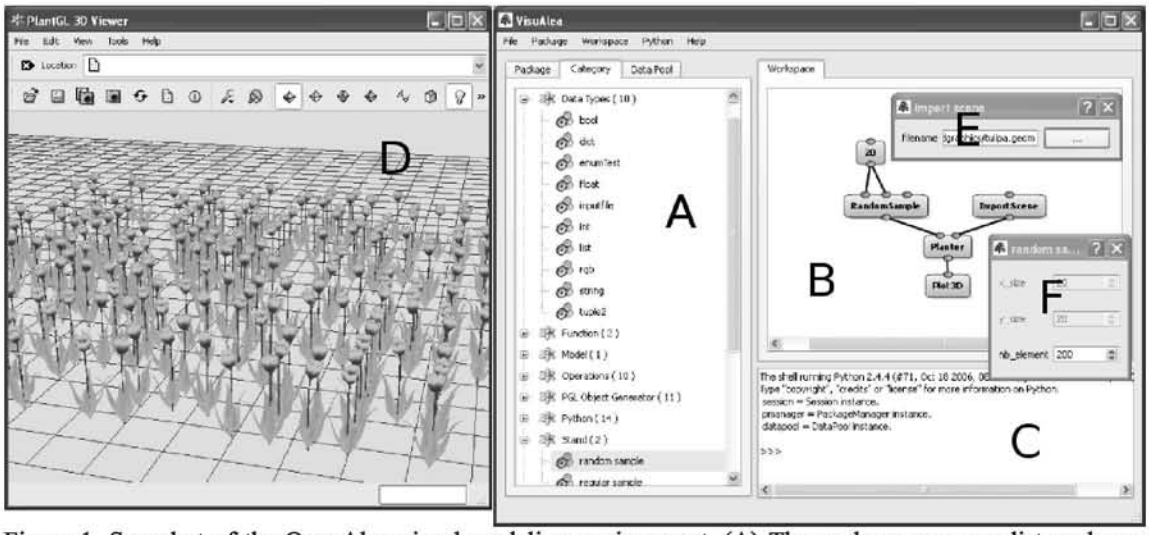

Figure 1. Snapshot of the OpenAlea visual modeling environment. (A) The package manager list packages and nodes found on the system. (B) The graphical programming interface enables users to build visual dataflow by interconnecting nodes. A 3D scene is built by associating a single geometry with a random distribution of points. (C) Low level interactions are done in the python interpreter. (D) PlantGL viewer is directly called by the Plot3D component. (E-F) Widgets specific to each component are automatically generated.

Others concepts are needed since the platform is developed in a distributed way. (d) A *package* is a deployment unit that contains a set of nodes, data as well as meta-information like authors, license, institutes, version, category, description and documentation. Finally, (e) the *package manager* allows for the dynamic discovery, introspection and loading of the available packages installed on the computer without requiring specific configuration. Thus, researchers can develop new functionalities that are added via the package manager at run-time without modification of the framework. Users can extend the framework by combining nodes into composite-nodes and share these macro nodes with other users. Dataflow containing nodes and composite-nodes can be saved as standalone application for end-user or as standard python script.

Data flowing through nodes are Python objects. An input and output port can be connected if their types are compatible, i.e. the output data can be implicitly cast to the type of the input port. Otherwise, an adapter has to be inserted between the two nodes to convert explicitly the data. A simple way to ensure input and output type compatibility between heterogeneous components is to use the standard data type available in python such as list, dictionary, etc. For more complex types such as graphs, some abstracts interfaces are provided in OpenAlea to standardize and ease communication.

Graphical environment

One of the key goals of OpenAlea is to enhance the use and the accessibility of plant modeling tools with a user-friendly interface. Particularly, the visual programming environment *Visualea* (see Figure 1.B) provides users the possibility to build graphically a dataflow by combining existing nodes and interactively edit them without having to learn a programming language. A graphical user interface (GUI) is associated with each computational node and enables the configuration and visualization of the node's data. Complex components will have specifically designed dialog boxes (see Figure 1.D). For others, a dialog box can be automatically generated according to the type of the input ports (see Figure 1.E-F). For that purpose, a widget catalog provides common widgets such as simple type editors (e.g. integer, float, string, color, filename, etc.), 2D and 3D data plotters, and sequence and graph editors. Thus, models that do not provide GUI can be easily integrated in the OpenAlea visual environment. Moreover, the catalog can be extended by packages to provide widgets for new data types. Finally, a Python shell has been integrated (see Figure 1.C) and provides a flexible way for programmers to interact procedurally with the components and to extend their behavior while taking advantage of the graphic representation of the data.

Integration of a component

PlantGL is a geometric library dedicated to plant modeling. It can be used as a versatile tool for functional structural plant modeling (Pradal et al., 2007). This library contains a hierarchy of geometric objects that can be assembled into a scene graph, a set of algorithms to manipulate the geometric objects and some visualization tools.

This module is written with 200k lines of C++ code that use various libraries such as Qt, OpenGL, qhull, *etc.* Relying on devoted tools such SConsX and DistX for compilation and installation makes the task easier since customization for a particular environment can be shared with other users of these tools.

Wrapping methods using Boost.Python have been implemented that make PlantGL accessible from python. Procedural composition of geometric objects can thus be quickly achieved with this language to build particular vegetal structures. Integration to *Visualea* enables graphical manipulation of the objects in the spirit of XFrog where visual programming is used for geometric plant modeling. Several levels of node abstraction have been implemented. The simplest one allows creating and editing the different geometric primitives of PlantGL (e.g. sphere, cylinder, NURBS surface, etc.). Basic primitives are graphically assembled in a scene graph and eventually passed on the visualization node. To simplify the construction of a complex model, more abstract nodes can

be defined with high level procedural geometric construction methods. In particular, some nodes that manifold a geometry and arrange the instances according to various biological patterns have been implemented. Figure 1.D shows for example a "planter" node that takes a random distribution of points on inputs and the geometry of an individual plant that can be used to create a natural scene by inferring a plant at each point. Additionally, nodes can easily be extended through the Python language to create new functionalities.

Several other components have been integrated in OpenAlea with similar techniques (Pradal et al., 2004). Some scenario, coupling different modules for plant architecture and ecophysiologial modeling will be presented during the poster session at the conference (Dufour-Kowalski et al., 2007).

Acknowledgments

This research has been supported by the developer community of OpenAlea and by grants from INRIA, CIRAD, and INRA (the Réseau Ecophysiologique de l'Arbre).

References

Prusinkiewicz, P. and Lindenmayer, A. 1990. The Algorithmic Beauty of Plants. Springer-Verlag New York, Inc.

Lintermann, B. and Deussen, O. 1999. Interactive Modeling of Plants. IEEE Comput. Graph. Appl. 19, 1, 56-65.

Fayad, M. and Schmidt, D. C. 1997. Object-Oriented Application Frameworks. Commun ACM 40, 10, 32-38

Johnston, Hanna, J., and Millar. 2004. Advances in dataflow programming languages. ACM Comp. Surv. 36, 1, 1-34

Pradal, Boudon, Donès, Durand, Fournier, Sinoquet, and Godin. 2006. OpenAlea: A platform for plant modelling, analysis and simulation, in: EuroPython 2006

Pradal C., Boudon F., Nouguier C., Chopard J., and Godin C. PlantGL : a Python-based software for 3D plant modelling at different scales. To be submitted.

Sanner M.F., Stoffler D. and Olson A.J. 2002. ViPEr, a visual Programming Environment for Python. In Proceedings of thre 10th International Python conference.

Szyperski, C. 2002. Component Software: Beyond Object-Oriented Programming. 2nd ed. Addison-Wesley Pro. Boston.

Demsar J, Zupan B, Leban G. 2004. Orange: From Experimental Machine Learning to Interactive Data Mining, White Paper (www.ailab.si/orange)

Knight, S. 2005. Building software with SCons. Computing in Science & Engineering 7, 1, 79-88

Pradal, Donès, Godin, Barbier de Reuille, Boudon, Adam, Sinoquet. 2004. ALEA: A software for integrating analysis and simulation tools for 3D architecture and ecophysiology, in FSPM04.

Dufour-Kowalski et al. 2007. OpenAlea: An open source platform for the integration of heterogeneous FSPM components, in FSPM07, Poster.