Proceedings of the 7th International Conference on Functional-Structural Plant Models, Saariselkä, Finland, 9 - 14 June 2013. Eds. Risto Sievänen, Eero Nikinmaa, Christophe Godin, Anna Lintunen & Pekka Nygren.
http://www.metla.fi/fspm2013/proceedings. ISBN 978-951-651-408-9.

An extension of the graph-grammar based simulator GroIMP for visual specification of plant models using components

Michael Henke,[1]* **Katarína Smoleňová,**[1] **Yongzhi Ong**[1] and **Winfried Kurth**[1]
[1]*Department Ecoinformatics, Biometrics and Forest Growth, Georg-August University of Göttingen, Büsgenweg 4, 37077 Göttingen, Germany*
**correspondence:* mhenke@uni-goettingen.de

Highlights: The FSPM platform GroIMP was extended by a graphical editor for components of rule-based models, enabling a better re-use and comparison of submodels. Components can be arbitrarily nested and can have several types of connectors with different semantics. The new tool provides a parallel maintenance of model code and of the graphical view showing the component structure.

Keywords: components, visual programming, software for modelling, GroIMP platform, XL language

INTRODUCTION

The most frequently used formalism to specify functional-structural plant models (FSPMs) are L-systems. A powerful generalization are parallel graph grammars. The programming language XL combines these with the power of the general-purpose, object-oriented language Java and gives thus support to the combination of process-based and rule-based submodels (Kniemeyer 2008). However, when applied to complex tasks, models specified in XL like those in other languages still tend to grow in an intransparent manner. Methods which are commonly used in software engineering are not yet supported (this being a general problem for scientific software, see Wilson 2006). As a further step towards a more transparent and flexible modelling process, we have enhanced the language XL and the open-source development platform GroIMP by tools supporting the creation of independent components, fostering also the interchangeability and comparison of submodels (e.g., for photosynthesis, structure formation, etc.). Interactive graphical visualization tools will make this approach also accessible to scientists with low experience in classical programming.

The issue of component-based development of FSPMs was already addressed by the open-source platform OpenAlea (Pradal et al. 2008), a framework which provides the possibility to create plant models by connecting components and to extend the available set of components by user-defined ones. However, OpenAlea has been designed for model integration and connection rather than for modelling feedback and retroaction between submodels. Furthermore, graph-transformation based models are not yet supported by it.

In the field of software engineering, the component-based approach has been successfully established and has led to divese tools and models, like CORBA (Mowbray & Ruh 1997), .NET (Box & Sells 2002) and JavaBeans (Sun Microsystems 2009). According to Szyperski (2002), components are "executable units of independent production, acquisition, and development that can be composed into a functioning system". Typical features are a useful functionality, encapsulation, re-usability, configurability, independent deployment and GUI-aided design. Usually, each component has its specific interfaces or *slots* which can either send or receive information to / from other components.

THE GRAPH MODEL

The composition of a model in terms of nested and interconnected components is represented by a directed graph $(C \cup S, E)$. Its nodes stand for the components C and their slots S; its edges belong to one of the three disjoint subsets E_C (refinement or "contains" edges between components), E_{SC} (slot-component incidence edges which indicate that a slot is an entrance or exit point of a component) and E_S (connectors between slots of different components). The connectors carry the information flow within the model. The edges between the different sorts of nodes of the graph have to conform to certain restrictions which we cannot list here exhaustively; we mention only the most important ones:

(1) The restriction of the graph to node subset C is acyclic. It reflects the "containment" relationship between components, or, in the hierarchical view, their nestedness.

(2) For each slot $s \in S$ there is exactly one adjacent component $c \in C$, i.e., there is exactly one edge (s, c) or (c, s) in E_{SC}. c is the component to which s "belongs". In our current implementation, the components

carry all functionality (similar to classes in the sense of object-oriented programming), but principally, slots could also be equipped with some functional abilities, e.g., for checking correctness of input data.

(3) Edges from E_S can only connect components which are at most one refinement level apart. If same-level components are connected, they must belong to the same supercomponent at the next-coarser level.

In the literature, different types of connectors and their characterization by algebraic properties are discussed (Bruni et al. 2011). Our graph model is designed for two types, "signal" and "call/receive" connectors, but could easily be extended to include more than two types. A "signal" connector mediates an unidirectional signal which can trigger an event in the receiving component. "Signal" connectors have thus a "push" semantics. A variant of "signal" connectors are "send" connectors which can transport a data stream. A "call/receive" connector mediates a function or method call or a request for a service. It carries information in both directions (the request in one direction, and the delivered result from the addressed component in the opposite direction). It has thus a "pull" semantics. In our current implementation, only "call/receive" and "send" connectors are realized.

Fig. 1 shows a simple example of a component structure. On the left-hand side, we see the classical graph view with the components as labelled circles, the slots as triangles and all edges as arrows. Edges marked with "/" are from E_C , they stand for the "containment" relationship. The dotted edge sends the output of the model (the result of some calculation in "structure") to the exterior output slot at the coarsest level, it is thus a "send" connector. The right-hand side shows the same component structure in a hierachical, nested view. The "plant" component (at the coarsest level) contains (visually) the three other components. Only edges from E_S are shown as arrows here.

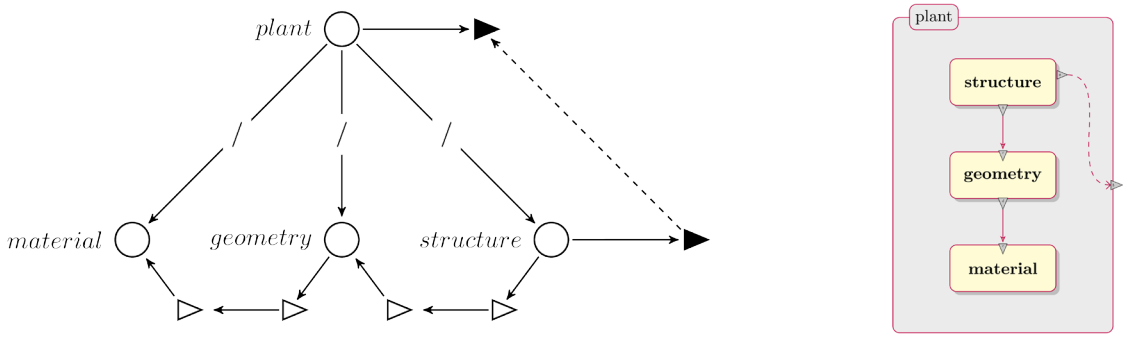

Fig. 1. A graph representing a component structure of a plant model, shown in two alternative views. Left: classical graph view. Right: hierarchical view, "containment" relationships are visualized by geometric inclusion of rectangles.

RESULTS

The here-discussed framework is written as an extension of the software GroIMP (Growth-grammar related Interactive Modelling Platform, Kniemeyer et al. 2007; http://sourceforge.net/projects/groimp/), a 3-d modelling platform based on graph transformations and employed for model implementation and visualization of results. GroIMP is designed as an integrated platform which incorporates modelling, simulation, visualization and user interaction, and provides a compiler and development tools for the rule-oriented language XL with specialized facilities for plant modelling. GroIMP's basic data structure is an attributed graph, with the nodes generally standing for objects (e.g., plant organs) and the edges for relationships. This graph can undergo transformations, specified in XL. The GroIMP graph offers itself in a natural way to represent the component structure of a model, expressed by the above-described graph, as well. Each node from C is then associated with a piece of code which represents the functionality of the corresponding component. Currently, all components have to be encoded in XL or Java and have to be accessible for the integrated XL compiler of GroIMP. Frequently used components can be kept permanently in an inherent library of GroIMP and are then accessible by a browser. With drag-and-drop, the user can take them from the browser window and link them graphically to other components in the component editor window.

The component editor offers two alternative views, corresponding to the two graph representations in Fig. 1. Furthermore, the XL code of each component can be directly accessed in a code window. With this, we deliberately violate the "black-box principle" of strict component-oriented development, thus answering to the demands of modellers from the FSPM community who have expressed their wish to have full insight and control of the software components they are expected to use. An additional developer environment with a

template component as a base allows such users to implement their own components quickly.
As example we use the simple tree with dichotomous branching and with coloured, spherical buds which comes along as a basic L-system template code within the GroIMP software (activate "open File" / "New" / "RGG project"). A separation of the XL code into the three components "structure", "geometry" and "material" is possible and leads to the structure shown in Fig. 1 (without the output slots pointing to the right; code not shown). In the "structure" component the morphological plant structure is defined, while the "geometry" component associates geometrical objects to the L-system labels used in "structure". The "material" component finally encapsulates the visual representation of the geometrical objects (plant organs) for the 3-d view. This example demonstrates that components can be used in XL to realize *aspect-oriented programming* in the sense of Cieslak et al. (2011). Aspects are the result of decomposing an organism into sub-entities which represent functionalities rather than spatial units. In this simple example, topological functions or rules reside in "structure", geometrical ones in "geometry", and optical properties are confined to "material".

A considerable amount of information is exchanged along the "call/receive" edges in this example, namely, the complete graph which represents the plant. E.g., the "geometry" component makes use of interpretive rules to insert some rotation nodes into the graph provided by "structure". Thus, the complete graph must be accessed by the components. This is somewhat contradictory to the parsimony principle of component-based development, which demands that inter-component information flow is small and strictly gate-kept by the slot specifications. Again, practical demands justify our wide interpretation of the component-based paradigm.

DISCUSSION AND OUTLOOK

It is our expectation that our component framework for XL will considerably improve the reusability of graph-transformation based submodels of FSPMs and will thus reduce the development time for plant models and enable a better maintenance and exchange between different teams. Although the development of the framework itself is not yet finished, we have started to create a library of ready-to-use components. Particularly, a set of photosynthetic rate models was implemented in XL, differing in their complexity from simple light-response curves to biochemical Farquhar-type models. In the future, we want to enable the usage of components written in other languages by utilizing wrapper components with Babel as glue (cf. Dahlgren et al. 2009).

ACKNOWLEGEMENTS

This research was funded by DFG under project identifier Ku 847/8-1.

LITERATURE CITED

Box D, Sells C. 2002. *Essential .NET. Vol. I: The Common Language Runtime.* Addison Wesley Professional.

Bruni R, Melgratti H, Montanari U. 2011. A connector algebra for P/T nets interactions. In: Katoen JP, König B (eds.): *CONCUR 2011*, *LNCS* **6901**, Springer, Berlin, 312-326.

Cieslak M, Seleznyova AN, Prusinkiewicz P, Hanan J. 2011. Towards aspect-oriented functional-structural plant modelling. *Annals of Botany* **108**(6):1025-1041.

Dahlgren T, Kumfert G, Epperly T, Leek K. 2009. Babel user's guide. Technical Report UCRL-SM-230026, Center for Applied Scientific Computing, Lawrence Livermore Nat. Lab., PO Box 808, Livermore, Cal.

Kniemeyer O. 2008. Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling. PhD thesis, University of Cottbus, http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:kobv:co1-opus-5937

Kniemeyer O, Buck-Sorlin G, Kurth W. 2007. GroIMP as a platform for functional-structural modelling of plants. In: Vos J, Marcelis LFM, deVisser PHB, Struik PC, Evers JB (eds.), *Functional-Structural Plant Modelling in Crop Production*. Springer, Dordrecht, 43-52.

Mowbray TJ, Ruh WA. 1997. *Inside CORBA. Distributed Object Standards and Applications.* Pearson.

Pradal C, Dufour-Kowalski S, Boudon F, Fournier C, Godin C. 2008. OpenAlea: A visual programming and component-based software platform for plant modeling. *Functional Plant Biology* **35**:751-760.

Sun Microsystems. 2009. *JavaBeans specification.*

Szyperski C. 2002. *Component Software. Beyond Object-Oriented Programming.* Addison-Wesley, London.

Wilson G. 2006. Where's the real bottleneck in scientific computing? *American Scientist* **94**(1):5.