CHAPTER 4

GROIMP AS A PLATFORM FOR FUNCTIONAL-STRUCTURAL MODELLING OF PLANTS

O. KNIEMEYER[#], G. BUCK-SORLIN[#,##] AND W. KURTH[#]

[#] *Brandenburgische Technische Universität Cottbus, Department of Computer Science, 03013 Cottbus, Germany*
[##] *Leibniz Institute of Plant Genetics and Crop Plant Research (IPK), Department of Cytogenetics, 06466 Gatersleben, Germany*

Abstract. The concept of relational growth grammars (RGG) is a recent effort to address the needs of functional-structural plant modelling. The programming language XL is the first language that implements this concept; it is made available for plant modelling as part of the modelling platform GroIMP. In this paper, an introduction to relational growth grammars and XL will be given using simple but instructive examples. It will be shown how the proven formalism of L-systems is integrated seamlessly into RGG/XL, which new modelling possibilities are provided, and how some standard tasks of modelling can be implemented in XL. Moreover, the examples will demonstrate some of GroIMP's 3D-geometric objects and algorithms that are useful for functional-structural plant modelling.

INTRODUCTION

By definition, functional-structural plant models combine functional with structural aspects of plants. This distinguishes them from models that are exclusively concerned with structure (*architectural* or *geometric models*) or physiology (*process-based models*). For the latter types of models, proven modelling techniques exist: process-based models can be implemented in mathematical software tools, while geometric models can be expressed using the formalism of L-systems (Prusinkiewicz and Lindenmayer 1990), because their underlying rule-based paradigm matches the observed plant growth at a macroscopic level.

A convenient modelling technique for functional-structural plant models has to combine the proven techniques for both functional and structural modelling. In combining these different techniques, one has to be aware of the fact that a whole is more than the mere sum of its parts:

- The structural view of L-systems, applied to process-based models, reveals network-like structures in those models, e.g., metabolic networks (systems of

J. Vos, L.F.M. Marcelis, P.H.B. de Visser, P.C. Struik and J.B. Evers (eds.), Functional-Structural Plant Modelling in Crop Production, 43-52.

chemical substrates and the reactions between them). However, networks cannot be represented reasonably within the L-system data structure, which is just a linear string of symbols.

- The procedural view of process-based models demands techniques for computation and information flow that assist in expanding these models from a single entity to the whole plant structure created by the structural part.

Thus, a convenient FSPM technique could be a combination of L-systems, extended to network-like structures, and a general-purpose programming language, extended by features that allow an easy access to, and easy computations on, these structures. The concept of *relational growth grammars* (RGG, Kniemeyer et al. 2004) is a formalization of such a modelling technique, the programming language *XL* its implementation.

Relational growth grammars were designed with the requirements of true functional-structural plant models in mind. L-systems form a conceptual foundation, but their underlying data structure of a linear sequence of symbols is replaced by a true graph, i.e., a set of nodes, connected by edges. Nodes generalize L-system symbols, edges generalize the sequential order of symbols in strings. This provides a natural way to represent sequences, trees (graphs without cycles) and networks in a consistent manner; typical relations of FSPM can be described as edges between nodes, e.g., the relations branch, successor, containment that were identified as fundamental by Godin and Caraglio (1998).

Graphs can be transformed by a system of rules (a graph grammar) similar to L-system rules; thus, the proven rule-based paradigm of L-systems is retained. Relational growth grammars are a special kind of graph grammars: graph-rewriting rules in general can be defined in a variety of ways, each leading to different semantics. The semantics of RGG rules is defined such that it covers L-system rules as a special case (at least in a practical sense). In particular, RGG rules are to be applied in parallel to the whole structure, and the syntax of concrete RGG languages should resemble that of L-systems.

In addition to these features of structure and its dynamics, an RGG language provides a general-purpose sublanguage that is powerful enough to allow for a concise modelling of other than structural aspects, e.g., processes. Such a language contains syntax and semantics for the integration of the structural part, e.g., it is possible to query for ancestors of a given node in the graph, or to calculate the total mass of all descendants using a concise language expression.

A suitable modelling technique is an important prerequisite for modelling. However, features of a modelling software like interactivity or visualization play an important auxiliary role in the modelling process. It is these features that enable us to obtain a both intuitive and in-depth insight into the model. A realistic image of a virtual plant can have a value in itself, or be used for communication and presentation purposes. *GroIMP*, the growth-grammar-related interactive modelling platform, provides these features to the modeller, together with an easy-to-use integration of the programming language XL.

THE MODELLING PLATFORM GROIMP

GroIMP is designed as an integrated platform that incorporates modelling, visualization and interaction. It exhibits several features that make it suitable for FSP modelling:

- The 'modelling backbone' consists in the language XL. It is fully integrated, e.g., the source code is edited in an integrated text editor and automatically compiled. Errors reported by the compiler are shown in a message panel and contain hypertext links to their source code locations.
- GroIMP provides a complete set of 3D-geometric classes for modelling and visualization. This includes turtle commands, primitives like spheres, cones and boxes, height fields, lights and spline (NURBS) surfaces (Piegl and Tiller 1997). Spline surfaces can be constructed by several techniques such as surfaces of revolution and swept surfaces (generalized cylinders).
- In addition, GroIMP provides a material system for the definition of 3D materials. Materials can be built by combining image textures and procedural textures in a flexible, data-flow-oriented way as it is customary in up-to-date 3D modellers.
- The outcome of a model is visualized within GroIMP, the free ray-tracer POV-Ray can be used to obtain a rendered image of high quality.
- In the visual representation of the model output, users can interact with the dynamics of the model, e.g., by selecting, modifying or deleting elements.

GroIMP is open-source software; it is licensed under the terms of the GNU General Public License. The latest version and information can be found at the web page http://www.grogra.de/, together with a set of example models.

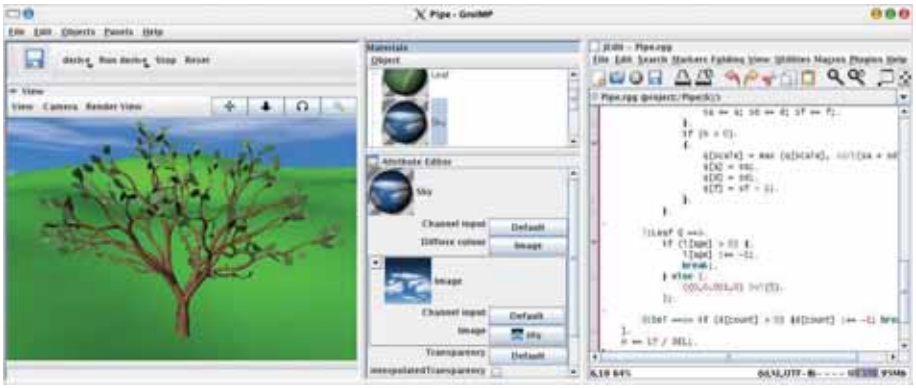

Figure 1 *A screenshot of GroIMP displaying a visualization of an XL model, GroIMP's material editor, and the XL source code in the integrated editor*

XL: AN IMPLEMENTATION OF RELATIONAL GROWTH GRAMMARS

The programming language XL is a concrete implementation of relational growth grammars. It is defined as an extension of Java, so all constructs of the Java language are available. However, several new features were added to Java, some of which will be presented in the sequel. It is outside the scope of this paper to describe and explain the syntax fully and formally. Therefore, the explanation is somewhat fragmentary. Nonetheless, we hope to give the reader an impression of the possibilities of RGG for FSPM. For a complete description, the reader is referred to the XL language specification (*The XL language specification* 2006). For the examples, we assume a working knowledge of L-systems and some experience with a programming language like C or Java.

Specifying L-system rules in XL

Because XL extends Java at profound language layers, it was possible to retain the known syntax of L-system rules (cf. Kurth 1994) for XL. E.g., the following growth rule for a binary tree is a valid XL rule (`RU` stands for a rotation around the turtle's up-axis):

```
A(x) ==> F(x) [RU(30) A(x/2)] [RU(-30) A(x/2)]
```

To be a legal XL programme, the rule has to be enclosed in brackets inside a method, and the module `A` has to be declared:

```
module A(float x);
void derive() [
    Axiom ==> A(1);
    A(x) ==> F(x) [RU(30) A(x/2)] [RU(-30) A(x/2)];
]
```

Instead of `derive`, another name could have been chosen for the method. `Axiom`, `F` and `RU` correspond to Java classes of that name, which are provided by GroIMP. The actual graph consists of instances (nodes) of these classes, which are connected in a tree-like manner. This is a difference to conventional L-systems which treat [and] as symbols: XL treats them as syntactic tokens which indicate a true branching in the graph. The application of the rules to a given graph is done as in L-systems; e.g., if an instance of class `Axiom` is found in the graph, it is replaced by a new instance of `A`.

Features beyond L-systems

The introductory example above already hints at two new features: Parameters of modules have to be declared (`float x`), this mechanism allows for arbitrary (Java) types of parameters. Rules are grouped in bracketed blocks, which in turn build statements that can be executed as part of a method. Using the usual Java constructs

like loops, if statements, and method invocations, it is possible to control order, time and number of applications. As a consequence, the mechanism of *table* L-systems is a special case, but also *decomposition rules* and even *interpretive rules* (in a future version of GroIMP) are naturally integrated into the new framework.

For the left-hand side of rules, but also for *query expressions*, XL contains a versatile query syntax. The previous example contains simple queries for all nodes of classes `Axiom` and `A`. However, we can also ask for all nodes from a certain starting node. E.g., the following method `leaves`

```
Leaf* leaves(Node c) {
    yield (* c (-->)* Leaf *);
}
```

contains the query `c (-->)* Leaf` which searches for all nodes of class `Leaf` that can be reached from the given node `c` by traversing an arbitrary number of edges `(-->)`. These nodes are returned by the query because `Leaf` is the textually rightmost pattern. The query is enclosed in asterisked parentheses to form a valid XL query expression, the method `leaves` just yields the set of `Leaf` objects (`yield` and the asterisk in the method header are necessary instead of `return` to indicate that a set of `Leaf` objects is returned). The query could be used to determine the total leaf length of leaf descendants of `c` as in `sum(leaves(c).length)`. These queries, together with operators like sum that perform calculations on sets, are an essential ingredient of XL, because they assist in implementing the process-based part and linking it with the structure. They generalize GROGRA's *arithmetical-structural operators* (Kurth 1998).

A SIMPLE MODEL OF A CARROT FIELD

In this section a simple model of a carrot field will be developed step by step. A more elaborate model (of barley) is presented by Buck-Sorlin et al. (this volume).

A carrot including carbon allocation and transport

To start with, a single carrot will be modelled. The rules that create the root and the structure of three compound leaves follow the pattern of conventional L-systems rules:

```
Axiom ==> Carrot F(0) for((1:3)) (RH(120) [RL(13) F(0)
RL(13) A(0)]);
A(n) ==> if (n < 3)( //Continue development of compound
leaf
    for ((1:2)) (
      F(0) [RU(60) A(n+1)] [RU(-60) A(n+1)])
    F(0) A(n+1)
  ) else Leaf;     //Terminate growth by a Leaf-node
```

The compound leaves are grown for n=3 iterations.

Initially, the internodes F have a length of zero. Their growth is governed by carbon that is produced in Leaf objects and is transported downwards to the root:

```
x:Leaf ==>> if (probability(0.05)) (x [Carbon(0.03)])
else break;

n:. [c:Carbon] -ancestor-> a:F ==>> n, a[c];

n:F [c:Carbon] ::> { float v = 0.2 * c[value];
                     c[value] :-= v; n[length] :+= v; };
```

In the first line, if an object of type Leaf is found, it is referred to by the identifier x. With a probability of 5%, x is replaced by itself having a new Carbon particle with an initial amount of 0.03 appended; otherwise the rule application is terminated (break). This and the next rule are graph replacement rules, indicated by the arrow ==>>. In contrast to the L-system-like rules indicated by ==>, which embed the right-hand-side nodes exactly where the left-hand-side pattern matched in the graph, graph replacement rules can specify arbitrary locations for the insertion of nodes. E.g, the second rule looks for a node n of arbitrary type (symbolized by a dot) which has a Carbon particle c attached to it and which has an ancestor a (a node in basipetal direction) of type F. Then it removes c from n and attaches it to a, which in principle may be at an arbitrary graph distance from n. Basically, transports can be modelled by context-sensitive L-systems, too. However, the freedom of choosing source and target locations for transport is restricted by their local context matching.

The third rule is neither an L-system-style nor a graph replacement rule: Its rule arrow ::> indicates an *execution rule* that executes its right-hand-side statements for every match of the left-hand-side pattern. No structural change to the graph is performed by such a rule. Here, for each internode n of type F to which a Carbon particle c is attached, a fraction v of c's carbon pool is allocated by removing it from c and elongating n. The allocation uses another new feature of XL, namely *quasi-parallel* assignments indicated by a prefixed colon (here :+=, :-=). They take effect as if they were executed in parallel, which means that the changes they cause are not visible until all rules of the currently active rule set have been applied. This is of special importance for process-based models where the computation for one step should consistently use the current values for this step and not partially the newly computed ones.

Competing carrots

The model presented so far deals with a single carrot. For a field of carrots, effects of competition between individuals have to be considered. In our simple model, competition is based on mutual shadowing, which controls the amount of carbon that is produced in the leaves. The first step is, for every carrot c, the computation of the total length of all internodes F of other carrots d that have a base in c's light cone:

```
c:Carrot ::>
{ Tuple3d m = mean(location((* c (-->)* Leaf *)));
  c[shadow] := sum((*
   d:Carrot, ((d != c) && (distance(c, d) < 2)),
   d (-->)* f:F, (isInsideCone(f, m, HEAD, 60))
  *)[length]); }
```

Firstly, all descendants (nodes in acropetal direction) of `c` of type `Leaf` are determined using the transitive closure `(-->)*`, which stands for an arbitrary number of edges between `c` and a `Leaf`. The centre of their locations is computed and stored in `m`, which is of class `Tuple3d`, i.e., a 3D vector. Afterwards, for every other carrot `d` within a distance of 2 from `c`, all of its internodes (descendants of type `F`) that lie within a light cone around `m` are determined. The sum of their lengths is stored as `c`'s `shadow`-value. The rule demonstrates how XL's query expressions and GroIMP's 3D-algorithms like `distance` and `isInsideCone` cooperate, allowing the modeller to implement both flexibly and concisely the competition for light. Thus, the modeller is not bound to a set of predefined functions that are provided as black boxes by the modelling software.

This total shadowing length influences the carbon production in `Leaf` objects by a modification of the carbon production rule of the individual model: its fixed carbon amount of 0.03 has to be replaced by a function that depends on the `shadow`-value of the leaf's carrot.

A digging rodent

The carrot-field model has a tree-like structure and could also be described by a

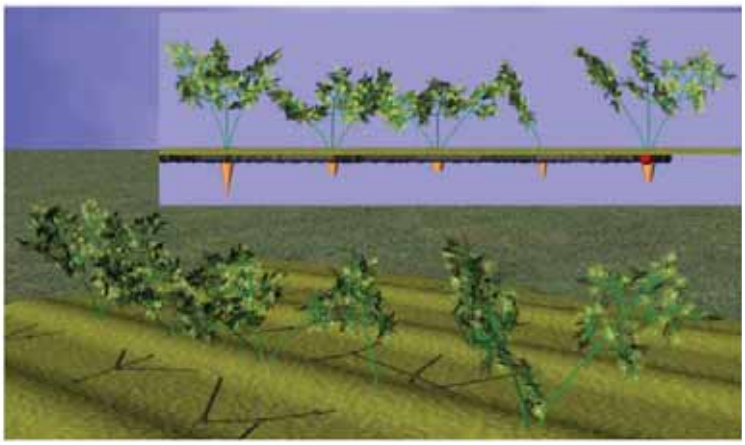

Figure 2*. Snapshot of the carrot-field model. The user has intervened by cutting off two leaves of the second carrot from the right; this favours its neighbours. However, the median carrot and its left neighbour suffer from being damaged by the vole (the leaves sink down, thereby reducing the light interception capability). The burrow system shows through the soil's surface*

globally-sensitive L-system. To show the potential of the true graph structure of RGG, let us introduce a water vole (*Arvicola terrestris*, a rodent) that digs a burrow system. It cycles through the burrow and feeds on carrot roots. Tunnels form edges between burrow nodes, which may happen in a cyclic way. The rules consist of about 35 lines of source code, which have been omitted here due to space limitations. However, the complete example model is available as part of GroIMP's example gallery, see http://www.grogra.de/. Figure 2 shows a snapshot of the model after a number of steps.

COMPARISON WITH L-SYSTEMS

Relational growth grammars are not the only extension of the original L-system formalism: in order to incorporate more and more aspects of true plant growth in L-system models, the formalism of L-systems itself has been extended in several directions:

- *Context-sensitive* L-systems allow the specification of rule contexts that enable the modelling of local transport. The inclusion of *new context* (Karwowski and Prusinkiewicz 2003) allows for a faster transport (in terms of the number of steps that are necessary for a given distance).
- *Globally sensitive* and *open* L-systems take the global environment into account (Kurth 1994; Měch and Prusinkiewicz 1996). While globally sensitive L-systems are equipped with a fixed set of environmental query functions (e.g., shadowing calculations), open L-systems make use of a generic interface to an external programme.
- *L+C* adds L-system rules to the C++-language by means of a source code preprocessor (Karwowski and Prusinkiewicz 2003).

However, each extension remains in the framework of L-systems, i.e., the data structure remains a string of symbols. By contrast, relational growth grammars extend L-systems along the lines of graphs and graph grammars to encode and transform the natural complexity of FSPM in a natural way. The advantages of RGG over L-systems lie in the increased expressiveness of the formalism:

- Tree- and graph-like structures are stored as they are, whereas L-systems encode them as a sequence of symbols. The latter is acceptable for tree-like structures, but it is inadequate for graph-like structures as they arise in process-based models or in geometric models at the cellular scale. Nodes carry arbitrary attributes, generalizing modules of parametric L-systems.
- The immediate representation of the structure, without the need for a turtle interpretation step, simplifies the specification of rules that include local or global context, and it simplifies computations on the structure. As an example, consider the shadow calculation in the carrot model where geometric relations are defined directly on the nodes of the structure.
- The inclusion of a general-purpose programming language which has a complete and easy access to the whole structure helps in implementing the process-based part of an FSPM (e.g., consider once again the shadow calculation). In principle, this can be done for L-systems, too. The programming language L+C (Karwowski and Prusinkiewicz 2003) integrates L-systems and the programming

language C++, but defines no syntactic support for access to and computations on the whole structure.

On the other hand, there exists a downside of the increased expressiveness:

- A more complex formalism is more difficult to learn. However, L-systems are seamlessly integrated, even on the syntactic level. This helps in getting familiar with RGGs if some knowledge of L-systems is already present.
- The execution speed at runtime cannot reach the speed of L-systems, because L-system rules can be applied much faster than graph-rewriting rules.

INTEGRATION IN RESEARCH PROJECTS

Both XL and GroIMP are open-source software, written in Java. It is possible to integrate XL or GroIMP within one's own framework. In the case of GroIMP, this amounts to the inclusion of (a subset of) GroIMP's Swing-based GUI-components in a Java application, together with some housekeeping code.

The integration of XL is even more flexible: the language XL is defined and implemented without reference to GroIMP or a particular graph model. By implementing a suitable interface to XL, any relational data source can be accessed and transformed immediately by XL's query and rewriting features. Such a relational data source may be, e.g., a tree-like structure of an existing model or a scene graph of a 3D-software. Interfaces for the commercial 3D-modelling programmes CINEMA 4D, Maya, and 3ds Max, were implemented by René Herzog, Udo Bischof and Uwe Mannl, as part of their bachelor theses (unpublished, but see http://www.grogra.de/).

OUTLOOK

Though XL and GroIMP have reached an advanced stage, there are enough interesting topics left for implementation and research:

- XL interfaces for existing plant-modelling tools could be implemented, e.g., for the open-source project ALEA (Pradal et al. 2004).
- The XL interface not only supports binary relations (edges), but also *n*-ary relations (hyperedges). An implementation of such relations could be used for rule-based modelling of cellular structures. E.g., the vv formalism (Smith et al. 2003) uses the ternary relation *i nextto j in k* in order to model surfaces imperatively; using XL this could be done in a rule-based manner.
- Process-based models often contain differential equations. GroIMP does not yet provide solvers for this kind of equations; it is an interesting question how such numerical algorithms can be combined conveniently with rules.

ACKNOWLEDGEMENTS

This research was funded by the DFG, partially under grant Ku 847/5 and Ku 847/6-1 in the framework of the research group 'Virtual Crops'. All support is gratefully acknowledged.

REFERENCES

Godin, C. and Caraglio, Y., 1998. A multiscale model of plant topological structures. *Journal of Theoretical Biology*, 191 (1), 1-46.

Karwowski, R. and Prusinkiewicz, P., 2003. Design and implementation of the L+C modeling language. *Electronic Notes in Theoretical Computer Science*, 86 (2), 1-19.

Kniemeyer, O., Buck-Sorlin, G.H. and Kurth, W., 2004. A graph grammar approach to Artificial Life. *Artificial Life*, 10 (4), 413-431.

Kurth, W., 1994. Growth Grammar Interpreter GROGRA 2.4: a software tool for the 3-dimensional interpretation of stochastic, sensitive growth grammars in the context of plant modelling: introduction and reference manual. *Berichte des Forschungszentrums Waldökosysteme der Universität Göttingen*, Ser. B (38). [http://www.grogra.de/]

Kurth, W., 1998. Some new formalisms for modelling the interactions between plant architecture, competition and carbon allocation. *Bayreuther Forum Ökologie*, 52, 53–98. [http://www.grogra.de/]

Měch, R. and Prusinkiewicz, P., 1996. Visual models of plants interacting with their environment. *In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. SIGGRAPH, New Orleans, 397-410. [http://delivery.acm.org/10.1145/240000/237279/p397-mech.pdf?key1=237279&key2=3522901611&coll=portal&dl=ACM&CFID=11111111&CFTOKEN=2222222]

Piegl, L. and Tiller, W., 1997. *The NURBS book*. 2nd edn. Springer, Berlin. Monographs in Visual Communication.

Pradal, C., Dones, N., Godin, C., et al., 2004. ALEA: a software for integrating analysis and simulation tools for 3D architecture and ecophysiology. *In:* Godin, C., Hanan, J., Kurth, W., et al. eds. *Proceedings of the 4th International workshop on functional-structural plant models, 7-11 June 2004, Montpellier*. Montpellier, 406-407. [http://www-sop.inria.fr/virtualplants/Publications/2004/PDGBBAS04/4thFSPM04_S8Pradal.pdf]

Prusinkiewicz, P. and Lindenmayer, A., 1990. *The algorithmic beauty of plants*. Springer-Verlag, New York.

Smith, C., Prusinkiewicz, P. and Samavati, F., 2003. Local specification of surface subdivision algorithms. *Lecture Notes in Computer Science*, 3062, 313-327.

The XL language specification, 2006. Brandenburgische Technische Universität Cottbus. [http://www.grogra.de/]