J. McCormack: "A Developmental Model for Generative Media", in M. S. Capcarrere, A. A. Freitas, P. J. Bentley, C. G. Johnson and J. Timmis (eds),Proceedings of the 8th European Conf. on Advances in Artificial Life, (ECAL 2005), Canterbury, UK, 5-9 September 2005, LNAI Vol 3630, Springer-Verlag, Berlin, Germany, ISSN: 0302-9743, ISBN-10: 3-540-28848-1, ISBN-13: 978-540-28848-0, pp 88-97.

A Developmental Model for Generative Media

Jon McCormack

Centre for Electronic Media Art
School of Computer Science and Software Engineering
Monash University
Clayton, Victoria 3800, Australia
jonmc@csse.monash.edu.au

Abstract. Developmental models simulate the spatio-temporal development of a complex system. The system described in this paper combines the advantages of a number of previously disparate models, such as timed L-systems and cellular programming, into a single system with extensive modeling flexibility. The new system includes the ability to specify dynamic hierarchies as part of the specification, and a decoupling of cell development from interpretation. Examples in application areas of computer animation and music synthesis are provided.

1 Introduction

We are interested in generalized models that simulate the continuous development of some complex system in space-time. This paper describes a new developmental system for the dynamic simulation of organic forms and processes. By decoupling the generative process from the generated output, dynamic models can be created in a variety of different application domains, including biological and botanical simulation, music composition, interactive animation, and computer graphics.

The developmental system described in this paper is strongly influenced by related work in developmental modeling using L-systems, in particular *parametric*, *timed* and *differential* L-systems. The original formulation of L-systems by Lindenmayer in 1968 was a conceptually elegant, discrete, symbolic model of development in cellular biology [1]. In 1990, Lindenmayer and Prusinkiewicz published *The Algorithmic Beauty of Plants*, with an emphasis on three-dimensional, visually realistic models of herbaceous plants [2]. This introduced a number of variations and extensions to L-systems in order to overcome the discrete, symbolic nature of basic constructs such as D0L-systems[1]. Subsequent developments incorporated other continuous developmental control, such as the use of differential equations to model growth and signaling in modules [3] and the effects of environmental constraints [4]. More recent work uses Chomsky grammars in combination with interactive curve editing software to obtain greater visual modeling flexibility and control [5].

[1] D0L-systems are *deterministic* and *context free*.

Timed L-systems were proposed by Prusinkiewicz and Lindenmayer [2] as an extension for modeling continuous development with D0L-systems. Their description was limited to D0L-systems without parameters and they restricted their modeling examples to the development of the simple cellular structure of *Anabaena catenula*. The cellular developmental model described in the following section follows naturally from *timed, parametric*, *stochastic* L-system models developed by the author. Further details on these specific extensions can be found in [6, 7].

In parallel with these developments, a number of cellular models of development have been proposed. The cellular model of Fleischer and Barr [8] modeled developing cells that exchanged chemicals by diffusion with simple 'cell programs'. This research was applied to areas such as three-dimensional texture synthesis [9]. More recent work combines related cellular development models with evolution to create structures that grow into target shapes [10].

One area that these systems have difficulty in addressing is in the design of a hierarchy – a mechanism often observed in natural systems. Additionally, they are typically designed for some specific simulation or application and are not necessarily applicable to generalized development. The system described in this paper addresses these issues. The following sections describe the model in detail.

2 The Cellular Developmental Model

2.1 Cell Definitions

In developing this system, we will consider a basic automaton, which is referred to as a *cell*. The name is used as a metaphorical interpretation of cells as found in biological life. The model is 'biologically inspired', but is not designed to reflect a literal interpretation of cellular development. This cellular abstraction is capable (as a simulation) of functions a biological cell does not have, and reciprocally, the biological cell is capable of many functions not possible with the model described here. Cells exist in an abstract entity called the *world*. The world is responsible for the *creation*, *removal*, and *interpretation* of cells that exist within it. Details of these terms and the world itself will be discussed shortly.

A cell is composed of four principle components (refer Fig. 1):

- A *label*, $s \in V_T$, where V_T is an alphabet that is specific to the cell type (approximately corresponding to the single alphabet of an L-system). The type of a cell is distinguished by its ability to develop or be interpreted;
- A *state*, $\Sigma_T \in \left(\Re^* \times \Im^*\right)$ — a set of variables that reflect measurable properties (both internal and external) that the cell possesses. The state will change dynamically subject to the mechanisms of the cell;

A set of predicate *rules* or *productions*, $P_T \subset \left(V \times \Sigma^*\right) \times C(\Sigma) \times \left(V \times E(\Sigma)^*\right)^*$, where $C(\Sigma)$ and $E(\Sigma)$ are respectively the set of logical and arithmetic/functional expressions using parameters from Σ. Rules specify developmental changes to the cell, and may consider the cell state as well as the state of *neighbouring* cells (the concept of a neighbour is defined shortly);

- An *interpretation*, $I \subset \left(V_I \times E(\Sigma_T)\right)^*$, which is a set of instructions as to how the cell is to be realised in the world (V_I is the alphabet of a particular set of interpretative symbols). The interpretation can make use of the cell's state.

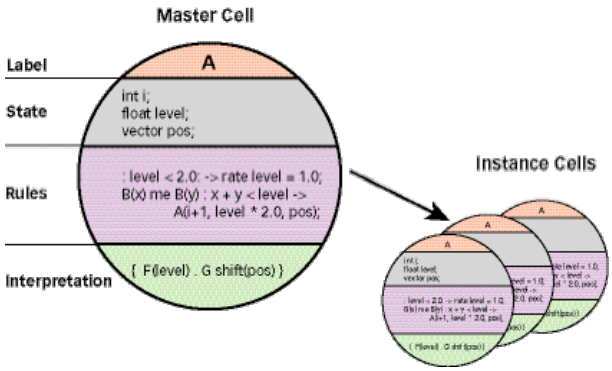

Fig. 1. Master and instance cells, and their principle components

Cellular instantiation follows the class/object model used in object-oriented programming [11][2]. Cells are instantiated into *pools* (spatial data structures), and there may be many *instance* cells with the same label, but each cell carries its own state, which develops independently. Conceptually, each cell also carries its own copy of the rules and interpretation defined for a cell of that label, although in most situations these are references to the rules and interpretation contained in the *master cell*. Thus, normally no distinction needs to be made between master cells and instance cells.

A special type of cell is called a *system*. A system has a label and contains state information, but does not have any rules or interpretation. Unlike a normal cell, a system cell may contain other cells, including other system cells. Thus, the system cell is capable of forming a hierarchical structure (Fig. 2). Systems contain an *initial state* (or *axiom*) that consists of a sequence of instance cells with particular state initialization information. They also maintain a pool wherein cells may be created, replaced, and deleted. A *root system* contains all other cells and systems and is created automati-

[2] Alan Kay, inventor of the *Smalltalk* programming language, used biological metaphors in its design, likening the concept of objects to 'cells' with walls, with the class providing a well-defined boundary between co-operating units [12].

cally by the world upon initialization. The root system's age will automatically reflect the developmental time of the entire system.

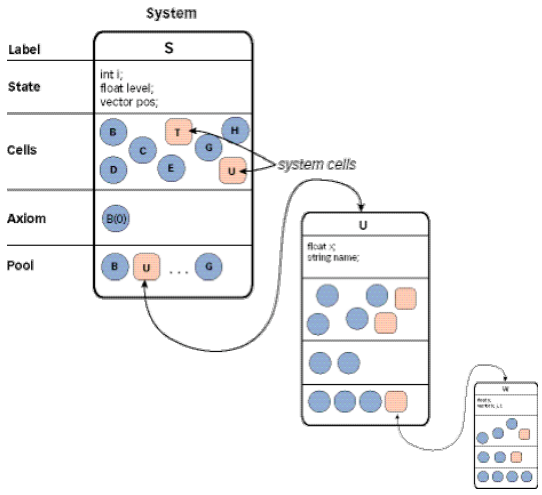

Fig. 2. A system cell may contain other cells, which in turn may be system cells. Thus the cellular hierarchy is formed

System cells may contain other cells (which may be systems too), but they cannot contain instances of themselves, nor can sub-systems contain instances of parent cells. This ensures the cellular hierarchy maintains a tree structure (rather than a cyclic graph, which would permit illegal circular definitions). A hierarchical structure is a good way of describing many natural patterns and forms [13].

Cells within a system develop asynchronously, however cells *may* synchronize development based on examination of each other's state. In addition, cells have access to the state of any parent cells, including the state of the root system. Specific components of the cell will now be described in more detail.

Cell State

Cell state captures the measurable components of a cell. The state is a vector composed of both user- and system-defined quantities. The user may define the cell state as required by the cell's particular type. In addition, all cells maintain a number of internal states that are defined and managed by the cell itself. Internal states are 'read-only' — available as symbols for use in productions, but they cannot be modified, hence they provide an introspection of various fixed components of the cell. The internal state includes the cell *age*, a continuous scalar that is automatically updated to reflect the age of the cell during its lifetime. An internal *status* contains four discrete states affecting overall cell behaviour: (i) *dormant* where no state changes are effected (the usual state of master cells); (ii) *birth* where state is initialized and the cell appears in its parent system's pool; (iii) *alive* where state development proceeds continuously

(e.g. states such as the cell's age are continuously updated); (iv) *dead* where the cell will be removed from the current pool it resides in.

Cell Rules

Cellular rules, denoted r_i are ordered sets of *predicate-action* sequences of the form:

$$r_i : \underbrace{\{context\} : predicate}_{\text{predicate component}} : \underbrace{(state\ calculations \mid cell\ actions)}_{\text{action component}} \tag{1}$$

Rules are numbered implicitly in the order of their declaration. While a cell is in the *alive* state, its set of rules is evaluated in ascending order. For a rule to be considered, first, the context requirements must be satisfied. Following this, if the predicate component evaluates to `TRUE` (non-zero), the action component is executed. The action component may consist of calculations that change the cell state, or actions that the cell should perform. The following sections describe each component in more detail.

Context

A context statement involves the position in the pool of the current cell in relation to other cells. A special reserved word, `me`, is used to represent the current cell. This allows cells with the same label to be used in context specifications. Context statements also specify the public state variables of cells involved in the context specification, and these identifiers may then be used in state calculations involving the current cell, i.e. a cell may update its state based on the state of its neighbours. Access to the state of neighbouring cells is read-only. Changing another cell's state directly is not permitted.

Here is an example context sensitive rule, where a cell maintains a component of its state to be the average of its neighbours[3], provided it exceeds some minimum threshold:

$$A(y)\ me\ B(z) : y > k_{\min}\ \&\&\ z > k_{\min} : x = \frac{y+z}{2} \tag{2}$$

which assumes the cell that owns this rule has a state variable, *x*. The rule first checks if the context is satisfied — that cells with labels 'A' and 'B' are at the 'left' and 'right' of the current cell. If that relationship is `TRUE`, the state parameters are then checked to see if they exceed some minimum constant value ($k_{\min}$), if so the current cell's state variable *x* is updated to be the average of the values of it's neighbours.

Context relations have a more flexible meaning than with context sensitive L-systems where the derivation string is a one-dimensional array, and so context matches are decided on by matching symbols to the left and right of the current symbol in the derivation string (a one-dimensional context). The pool in which the cells exist is designed in an abstract way, where the interpretation of neighbour relationships is flexible. This is achieved using polymorphic functions to match context based on

[3] This example uses a one-dimensional context relation; higher dimensional relations are defined in the next section.

pool type. It is important to match context dimension to pool dimension (e.g., a two-dimensional context relation makes no sense to a one-dimensional array).

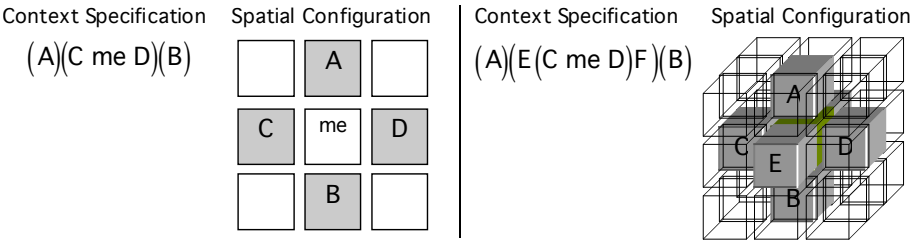

Fig. 3. Higher dimensional context relations and their specification

In the cellular developmental system, context may include other *spatial* relationships where context relations are satisfied when the spatial position of the cell is less than some Euclidian distance, or topological relationships such as the Von Neumann neighbourhood (Fig. 3) used in cellular automata simulations [14]. The use of parenthesis demarks dimensions when specifying context.

As the system described here uses polymorphic objects to represent the pool (e.g. linear set, multi-dimensional array, spatial structure), the interpretation of context is determined by the way the specific pool interprets the context statements. This permits flexibility in the types of simulations the system can perform. For example, such context relationships can be used in music generation where context relations work in two dimensions: pitch and time (hence context matching can be with chords, rather than notes.

State Calculations and the Differential Operator

State calculations are mathematical expressions that affect a cell's state. They are expressed in a similar manner to expressions in the C programming language. A rich set of functions is provided, including basic mathematical functions, trigonometric functions and a variety of stochastic and noise functions. A unary differential operator, `rate`, performs a specific form of ordinary differential equation solving with initial values. The operand for rate is a local cell state variable. The variable is integrated based on its initial value at birth according to the mathematical formula specified in the state calculation. The differential operator is useful for simulating processes such as chemical diffusion between neighbouring cells.

Actions

Actions correspond to the re-writing process in L-systems, being similar to the successor word of L-systems. However, the concept of rewriting can be misleading, due to the way cell actions are handled and cells are placed in the pool. In the case of L-systems, symbols are always *replaced* by rewriting productions. Whereas, for the cellular developmental model, in addition to replacement, cells may continue to exist in the pool while their actions cause new cells to be added (i.e. the cell action does not

necessarily replace (rewrite) the cell instigating that action). Possible actions are outlined in Table 1.

Table 1. Rule actions

Type	Description	Example action syntax
None	No action (the current cell remains)	$\rightarrow me$
Replace	The current cell is replaced	$\rightarrow A(x, y)$
Add	New cells are created and the current cell remains in the pool.	$\rightarrow A(x, y)\ me\ B(x + y)$
Delete	The current cell is deleted	$\rightarrow \varnothing$ (empty string)

2.2 Cell Interpretation

Thus far, cells have been considered in abstraction, without any method of realising them in the world. The interpretation component specifies how the cell will be interpreted as the system develops. In the case of L-systems, developmental words are interpreted by a turtle, which creates geometry as it interprets the list of symbols. The case here is similar, with the exception that each cell may contain multiple instructions to the turtle, permitting individual cells to create more complex geometry without polluting the cellular developmental model with cells used only as part of some complex geometry building sequence.

The interpretation system is extremely flexible, in the sense that interpretation contains a list of special cells representing instructions. These cells are of a different type than normal developing cells (they do not develop), but may have associated parameters. These parameters may be set with expressions involving the cell's state. If a cell's state is changing over the lifetime of that cell (the *age* component for example), then the interpretation permits the ability to animate the parameters of interpretive instructions as the cell state changes. This is a more flexible and general form of the development functions associated with timed L-systems [7]. There is no direct dependence between cell development and interpretation, so a number of different interpretative sets can be used on the same developmental system. For example, a musical interpretative set issues musical instructions rather than geometric building ones.

This flexibility allows users of the system to realise their developmental system in a variety of ways, without the need to completely re-specify the grammar. The use of multiple instruction sequences in a single cell is a different solution to a similar problem encountered by Prusinkiewicz and colleagues in the development of their interactive system to model plants [5]. Here they combined a C-like programming language and Chomsky grammars to enable sequential rewriting of strings, rather than the parallel development specified by L-systems.

3. Examples

Complex structures are often modeled by decomposing them into a hierarchy. In this section, an example of this method is shown for developing a model of a multi-segmented, articulated animal with a walking gait. By using a hierarchical description, it is possible to specify structure in an intuitive way. The hierarchical structure of the animal is shown below in Fig. 4.

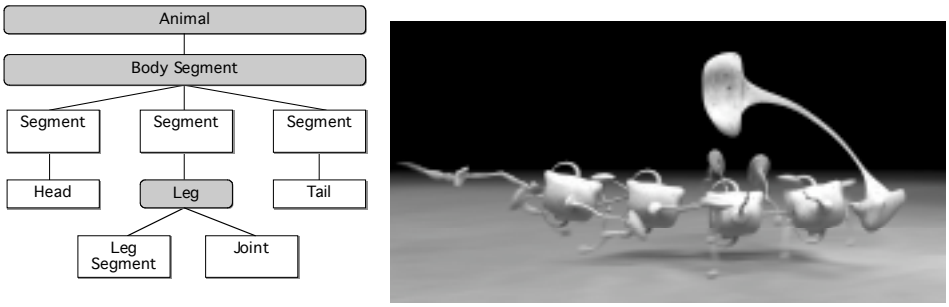

Fig. 4. Hierarchical specification of system cells (grey boxes) and module cells (left); the articulated creature in motion (right)

This structure is specified using the cellular programming language, containing cell definitions in a human readable form. The system simulates forward kinematics, with locomotive drivers at the joints to create legged gaits. The drivers function like a state machine controlling gait movement in each leg. The key advantage of the developmental system is in the flexibility of specification, permitting morphological changes to be made easily. For example, the number of body segments is controlled by a single parameter. Geometry is constructed using generalized cylinders; the shape and configuration controlled by lower-level cells driving geometry construction. A resultant still from the animated output of this system is also show in Fig. 4.

4.2 Music Generation

Interpretation of cell states is not limited to geometric constructs. Using an object-oriented approach allows the interpretation of cell states by methods other than a turtle interpretation. To provide different interpretations a collection of global modules are imported into the namespace of a particular system (the root system by default). These modules are directly interpreted as generative commands.

The musical commands use the concept of a state-based *player*, which is responsible for converting commands into actual music. The player maintains a state that includes the current pitch (note) and volume. The player converts incoming commands

into midi[4] messages, enabling any midi compatible device to play music generated by the system.

4. Summary

The developmental system described here unifies a number of previous L-system and cellular models in the application domain of temporal developmental systems. Based on both discrete cellular changes and continuous state development, the model successfully integrates these two modes of development, and permits complex temporal sequences not achievable using previous techniques.

The hierarchical nature of the cellular developmental system allows management of complexity from the point of view of the user specifying a system model. Hierarchical ordering increases the control over structure at variety of levels, hence reducing the 'brittleness'[5] of a flat grammar specification. This permits a more intuitive control over creation of modeled systems. As show in Fig. 5, the complex nature of cell development stands in contrast to the more simplified discrete D0L-system model.

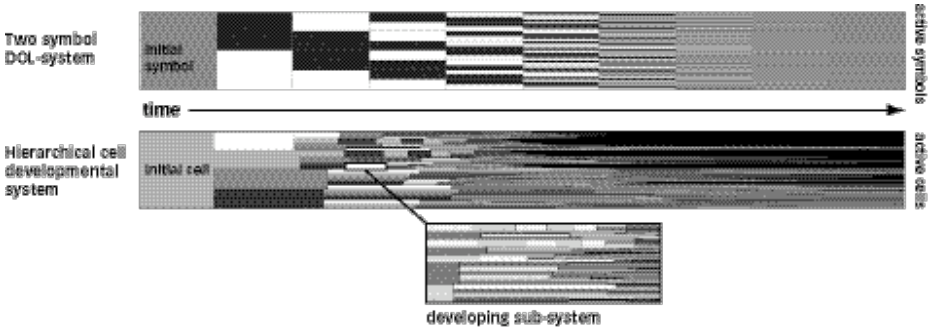

Fig. 5. Time-state diagram contrasting the developmental differences between discrete L-systems and the cellular developmental system described in this paper. The diagram shows the temporal development (from left to right) of each system. Shaded rectangles represent the symbols present (vertical axis) at any given time (horizontal axis). Both examples start from a single symbol or cell (the axiom). In the case of the DOL-System (top) each iteration is clearly synchronized and regular as the rewriting process proceeds in discrete time steps. In the case of the developmental cellular system, the sequence quickly becomes irregular and 'fractal' due to the individual developmental nature of cells. A single cell at the top level is shown expanded as a segment of a developing sub-system, illustrating the complexity that is contained by the hierarchy

[4] MIDI is a low-level serial communication protocol for musical instruments and musical controllers — see [15].

[5] That is, more robust to configuration changes — a change at one level in a hierarchy can have fewer side effects than for the equivalent description in a non-hierarchical system.

References

1. Lindenmayer, A.: Mathematical Models for Cellular Interactions in Development, Parts I and II. Journal of Theoretical Biology 18 (1968) 280-315
2. Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. The Virtual Laboratory. Springer-Verlag New York (1990)
3. Prusinkiewicz, P., Hammel, M., Mjolsness, E.: Animation of Plant Development. Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993) In Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, New York (1993) 351-360
4. Prusinkiewicz, P., James, M., Mech, R.: Synthetic Topiary. Proceedings of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994) In Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, (1994) 351-358
5. Prusinkiewicz, P., Mündermann, L., Karwowski, R., Lane, B.: The Use of Positional Information in the Modeling of Plants. Proceedings of SIGGRAPH 2001 (Los Angeles, California, August 12-17) In Computer Graphics Proceedings Annual Conference Series, ACM SIGGRAPH, (2001) 289-300
6. McCormack, J.: The Application of L-Systems and Developmental Models to Computer Art, Animation, and Music Synthesis. Ph.D. thesis, School of Computer Science and Software Engineering, Monash University, Clayton (2003)
7. McCormack, J.: Generative Modelling with Timed L-Systems. In Gero, J.S. (ed.) Design Computing and Cognition '04, Kluwer Academic Publishers, Dordrecht (2004) 157-175
8. Fleischer, K.W., Barr, A.H.: A Simulation Testbed for the Study of Multicellular Development: The Multiple Mechanisms of Morphogenesis. In Langton, C.G. (ed.) Artificial Life III, Addison-Wesley, Reading, Massachusetts (1994) 389-416
9. Fleischer, K.W., Laidlaw, D.H., Currin, B.L., Barr, A.H.: Cellular Texture Generation. Proceedings of SIGGRAPH 95 (Los Angeles, California, August 6-11, 1995) In Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, (1995) 239-248
10. Kumar, S., Bentley, P.J.: Mechanisms of Oriented Cell Division in Computational Development. First Australian Conference on Artificial Life (ACAL 2003). Canbera, Australia (2003)
11. Wirfs-Brock, R., Wilkerson, B., Wiener, L.: Designing Object-Oriented Software. Prentice Hall Englewood Cliffs, N.J. (1990)
12. Kay, A.C.: The Early History of Smalltalk. The second ACM SIGPLAN conference on History of programming languages. Cambridge, Massachusetts, ACM Press (1993) 69-95
13. Simon, H.A.: The Sciences of the Artificial. 3rd Edition. MIT Press Cambridge, Mass. (1996)
14. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning Ways for Your Mathematical Plays. Vol. 2. Academic Press New York (1982)
15. Selfridge-Field, E. (ed). Beyond MIDI: The Handbook of Musical Codes. MIT Press Cambridge, Massachusetts (1997)