Design Document For COMPUCELL and BIOLOGO

By: Trevor M. Cickovski
University of Notre Dame
Computer Science Graduate Student
College of Engineering
tcickovs@nd.edu

Under Supervision Of:

Jesus A. Izaguirre
University of Notre Dame
Computer Science Professor
College of Engineering
izaguirr@cse.nd.edu

June 20, 2003

Contents

Chapter 1

Introduction

We present here a proposed design for an edition to the current COMPUCELL framework. These editions will be made with the intention of increasing flexibility in the framework, improving software maintenance, but first and foremost to interface well with the domain-specific language BIOLOGO . Our hope is that in the future BIOLOGO will be the user's main method of running COMPUCELL , aside from web-based execution.

BIOLOGO is currently undergoing changes as well. Formerly just another programming language that is written in a text editor, now it is being changed to be XML-based, and the new compiler and code generator will generate code in an intermediate language to hopefully make the process simpler. The overall converter from the XML-based BIOLOGO file to these classes below is under development.

I have divided this document into three sections - each of these centering on different key features that this new design will offer with the help of BIOLOGO . Even without BIOLOGO , the design has been able to run with a C++ driver file, which we obviously won't need when it is interfaced with the current framework. Thus, we should be able to add this to the current framework.

Chapter 2

Fields And Evolvers

2.1 Description and Purpose

A field can be viewed as a multi-dimensional array - in our case, either a two or three dimensional array. Currently COMPUCELL works in two dimensions but there are current efforts being made to extend it to 3D. Example fields that have already been hardcoded into the COMPUCELL framework include: the lattice of pixels, the reaction-diffusion activator concentration, and the fibronectin concentration.

An evolver describes how a field changes overtime. Each field has one single evolver attached. The current COMPUCELL framework does not have evolvers explicitly, however the code that evolves the afore mentioned fields has been contained mostly in the classes `CompuCell`, `ChemotaxisLatticeConcentration`, `FibronectinLatticeConcentration`, and `RectangularPottsLattice`.

In this new design, we propose replacing the evolving code that is in various spots in the framework with classes that are *designed to evolve a specific type of field in a specific way*. Each evolver will contain an `init()` method which will be executed just once at the beginning of execution - usually used to initialize its field with values, and a `run()` method executed at every step, which evolves the field. Not only that, but we propose to have the user through a BIOLOGO program (1) control the order of the evolving of fields (for example, evolving the activator before evolving the cells), and (2) customizing the evolvers by passing certain values for parameters known by the evolvers.

Currently COMPUCELL uses three fields - one for the lattice, one for the activator concentration, and one for the fibronectin concentration. In this edition, both the field and evolver class hierarchies use inheritance. The root classes are `Field` and `Evolver`. `Field2D` and `Field3D` inherit from `Field`. Inheriting from these are any needed fields, for example, `FieldPotts2D` inherits from `Field2D` and is a field of type `Pixel`. `FieldConcentration2D` is a field of type `double` and also inherits from `Field2D`. A `FieldRD2D` or `FieldRD3D` field inherits respectively from `Field2D` or `Field3D`, but is a field of `Tuple`s. A `Tuple` is nothing more than a group of values. It is a doubly-templated class, with one template argument for the type of value within the `Tuple`, and another for the size of the `Tuple`. This is another extension that we are making to COMPUCELL - we want in reaction-diffusion to hold terms for both activator and inhibitor, not just activator. Thus, a RD field is a field of groups of two `double` values, or a field of type `Tuple<double, 2>`.

As mentioned, each of the three current fields in COMPUCELL uses a specific type of evolver (though now it is not specifically called an 'evolver'). In the new design, we create the classes `Metropoli-`

`sEvolver`, `SchnakenbergEvolver`, and `StandardFibroEvolver`, all inheriting from `Evoler`. Each are designed to evolve a `Field` of a certain type. This is summarized in the chart below, which shows each evolver, the type of field it evolves, and at a higher level what field is being dealt with. These fields are called morphogenesis fields for now, since that is what COMPUCELL simulates, but note that a future goal is to expand this language to work with processes other than morphogenesis. Thus, this chart is with respect to COMPUCELL and its current purpose.

Evolver	Type Of Field	Morphogenesis Field
MetropolisEvolver	Field<Pixel*>	Chicken Limb Cells
SchnakenbergEvolver	Field<Tuple<double, 2> >	Reaction-Diffusion
StandardFibroEvolver	Field<double>	Fibronectin

2.2 UML Diagram

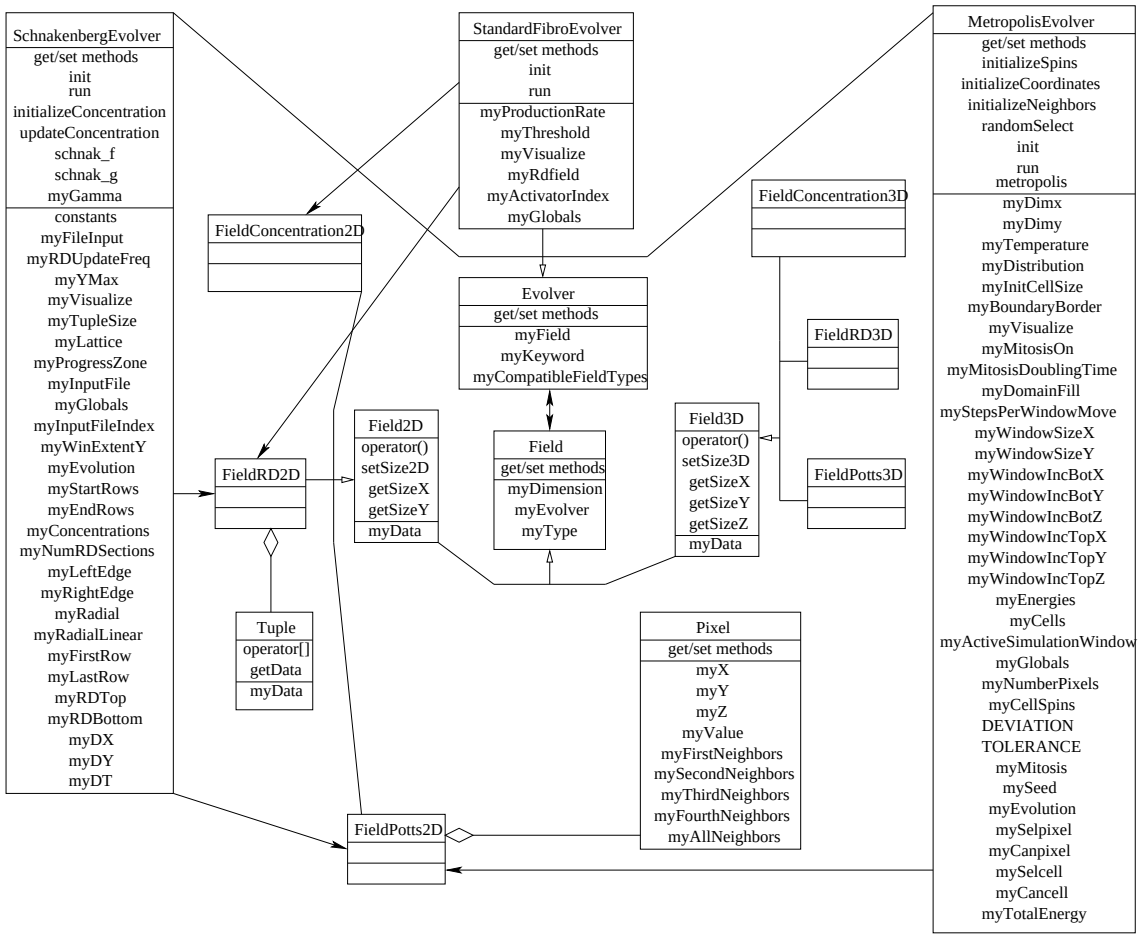

Figure 2.1: Field And Evolver Classes.

Chapter 3

Cell Classes

3.1 Description And Purpose

The main purpose of the objects here are to present the ground for defining a cell model. A cell model includes all variables and possible types for a cell that follows the model. When we say 'type', we mean a sort of state that the cell can be in, with certain unique properties. A cell can change type with the concept of rule-based state automaton, in other words - there are certain rules that govern the conditions in which a cell changes type. A cell of a certain model can only have a current type that is defined in the model, and can only change to another type defined in the model. In the current COMPUCELL , we will say that our cells follow the Chicken Limb Model. Within this model, each cell has several different variables such as J, lambdavolume, and targetvolume, and can be of one of three possible types: NonCondensing, Condensing, and Medium. In the new proposed design, we create a class `CellContext` from which all cell models will inherit from, and another class `CellType` which all cell types will inherit from. A `CellContext` object contains a pointer to an object of type `CellType`. Once again, in the current COMPUCELL there is only one model, `Chicklimb` - so in the new design this is a class that inherits from `CellContext`. We also create a `ChicklimbCellType` inheriting from `CellType`, and also `ChicklimbNonCondensing`, `ChicklimbCondensing` and `ChicklimbMedium` which inherit from `ChicklimbCellType` and are each possible types for a cell following the `Chicklimb` model. An important thing to note is that all of the afore mentioned classes in this section aside from `CellContext` and `CellType` will be generated by the BIOLOGO compiler/code generator. Thus the models that are followed in a simulation and all of their properties are completely under control by the user in the BIOLOGO program.

In the new design, we do assume certain properties for any cell, regardless of the model it follows. We give each cell variables for spin, volume, and surface area, along with a couple of variables for mitosis. However, it is of course optional whether or not these values are used in a simulation depending on what the user codes in BIOLOGO , thus control is still given to the user.

Taking the example of `Chicklimb`, `ChicklimbCellType` defines the new variables that the user defined in the BIOLOGO file. All of the subclasses of `ChicklimbCellType` give these variables values, and also contain a function that describes how a cell of that current type (example `ChicklimbNonCondensing` changes to one of a different type. These rules are written by the user in BIOLOGO .

Viewing the diagram in the next section, in addition the afore mentioned design, we have included some other classes which contain arrays of type `CellContext`, or to put it simpler, arrays of cells. `Globals` will also be generated by BIOLOGO (in the current design we have an array of cells), but `MetropolisE-`

`volver` is designed to evolve a cell lattice and `Mitosis` is designed to work on an array of cells.

3.2 UML Diagram

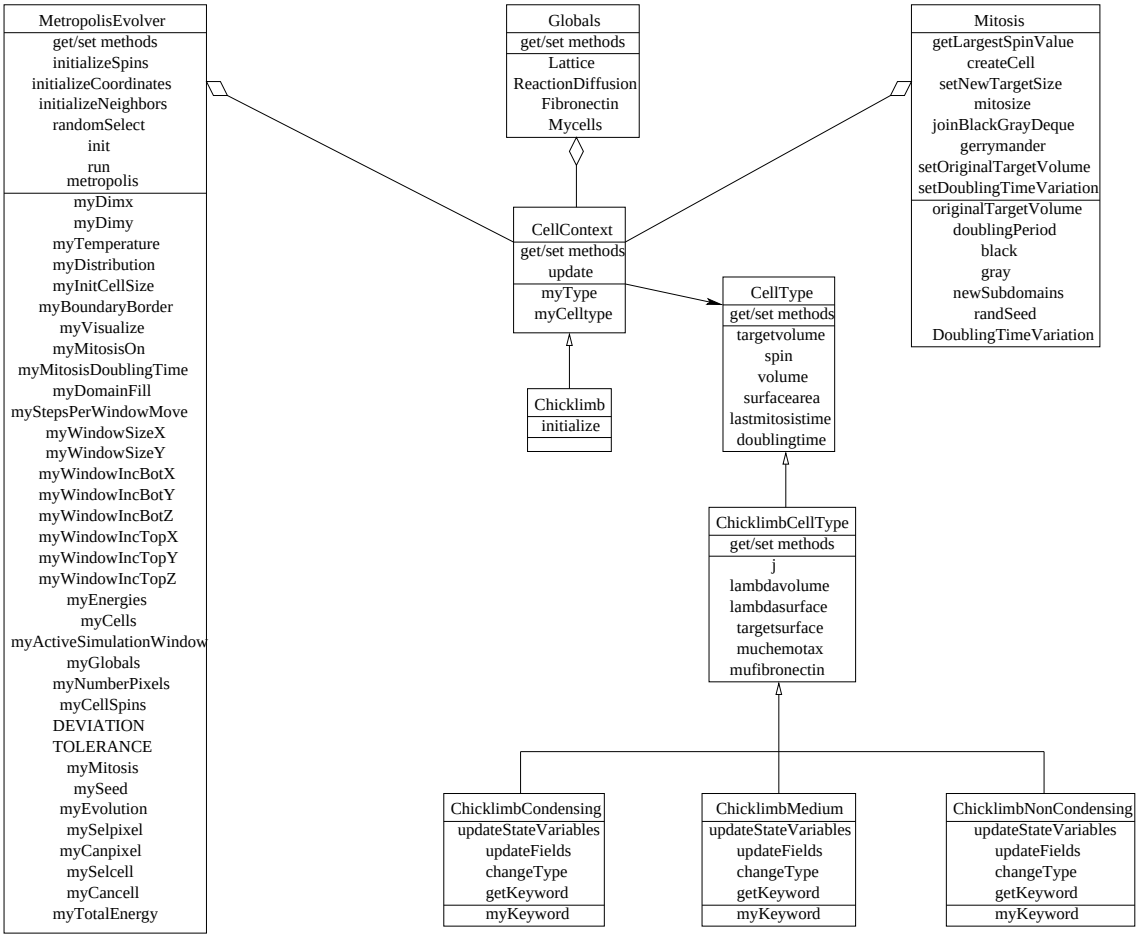

Figure 3.1: Cell Classes.

Chapter 4

Global Classes

4.1 Description And Purpose

The global classes represent everything that the user declared in the BIOLOGO file that is accessible throughout the file. In the new design, everything centers around a `Globals` class. This class will also be generated by the BIOLOGO compiler/code generator. Specifically contained in this class are:

- All of the fields that the user defined.
- All of the evolvers that were user employed with the fields.
- All of the energy hamiltonians that the user defined (these are within an `Energies` object).
- All of the global variables defined by the user (including arrays of cells).

4.2 UML Diagram

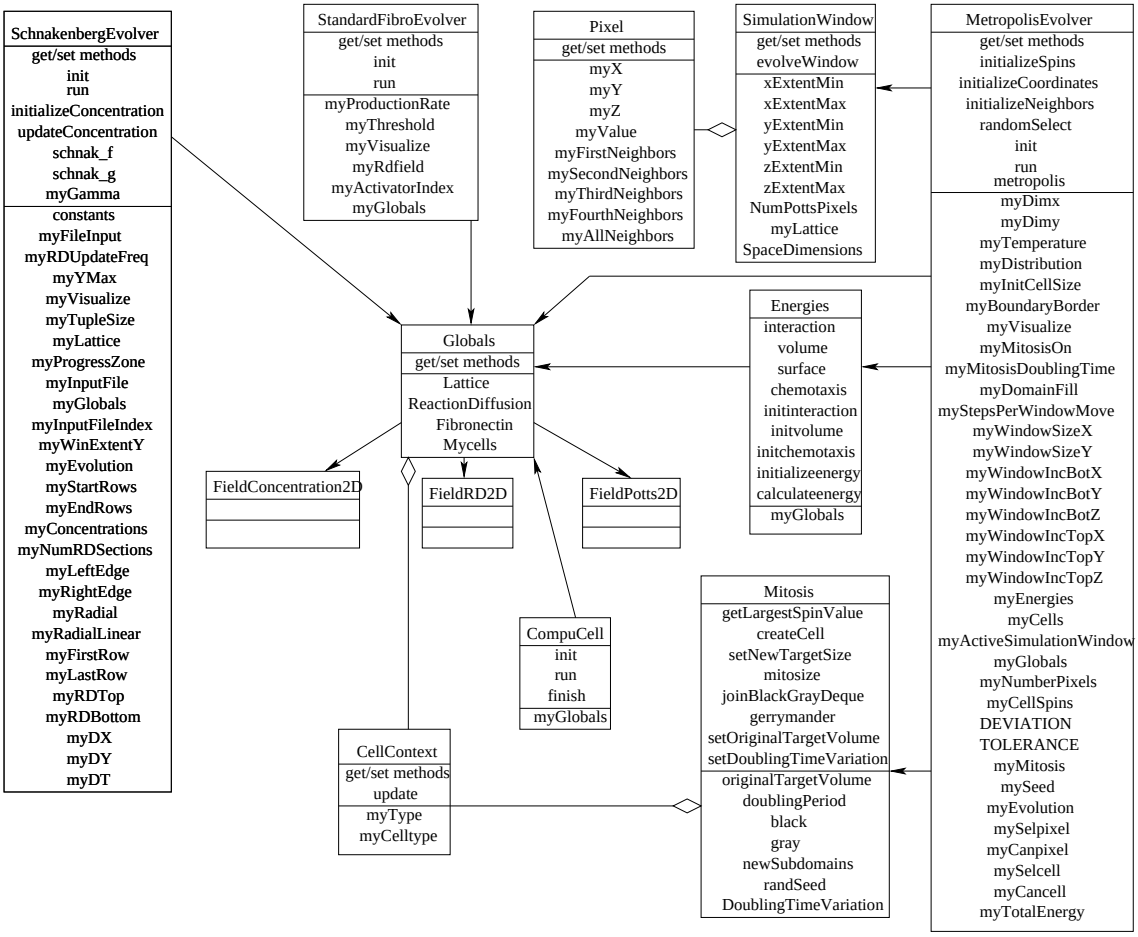

Figure 4.1: Global Classes.

Chapter 5

Conclusions

In conclusion I would like to reiterate the potential of this design and my hope is that it can be implemented soon with the redesign of old CompuCell by Joseph. Hopefully we can get these working together.