MODELING AND VISUALIZATION OF SYNTHETIC GENETIC CIRCUITS

Tyler R. Patterson

Abstract

The iBioSim tool is being developed to facilitate the construction and simulation of synthetic genetic circuits. In this project, we have created a user interface that is similar to those used to construct schematic diagrams which are familiar to electrical engineers. Promoters, chemical species, and biological relationships can be placed visually on a schematic diagram. Another enhancement was the creation of a new simulation visualization tool which allows the user to associate chemical species with color schemes, opacity, and cell size. This tool allows the user to see the cells behavior as if through a microscope.

I. Introduction

In 1977 a group of researchers managed to modify the DNA of the E. coli bacteria and were able to make it to produce the human hormone somatostatin [1]. Since that time genetic engineers have leveraged bacteria to help produce a wide range of useful products from pharmaceuticals to food additives to laundry detergent [1]. Synthetic Biology is an emerging field that attempts to model the chemical reaction networks inside a living cell in ways similar to an electrical circuit. While there are many tools that have been created for synthetic biology [2][3][4][5][6] there is still great need for more efficient methods for their modeling, analysis, and design [7]. iBioSim is one of the tools being developed for these purposes.

As is shown in figure 1 the iBioSim user interface initially consisted of lists of biological components. While this interface was usable, it was believed that it could be improved by allowing the user to work on the model at the graphical level, similar to the way programs such as Spice allow the user to build electrical schematics. This process has several advantages.

One advantage of a graphical interface is that a visual picture of the biological model that a user is creating can provide insight into the structure of the model that would be very difficult to gain by clicking through menus and reading the way things are connected. Another advantage is that having a visual representation will make iBioSim more accessible to people who might potentially use it. This will allow the tools strengths to be more widely used, increasing the likelihood that iBioSim will end up helping those who are using the tool to design useful, potentially life saving biological circuits. A final advantage is that having access to a schematic diagram opens the possibility to present simulation data directly on the schematic as an animated movie. In this project both a graphical user interface and a movie visualization mode have been created.

This paper will begin by describing the Genetic Circuit Model which is the data structure used by iBioSim. Schematic Capture, the new interface created to design genetic circuits, will be described. Finally the section on Visualization of Genetic Circuits will give an overview on the use and effectiveness of the new visualization mode that was implemented.

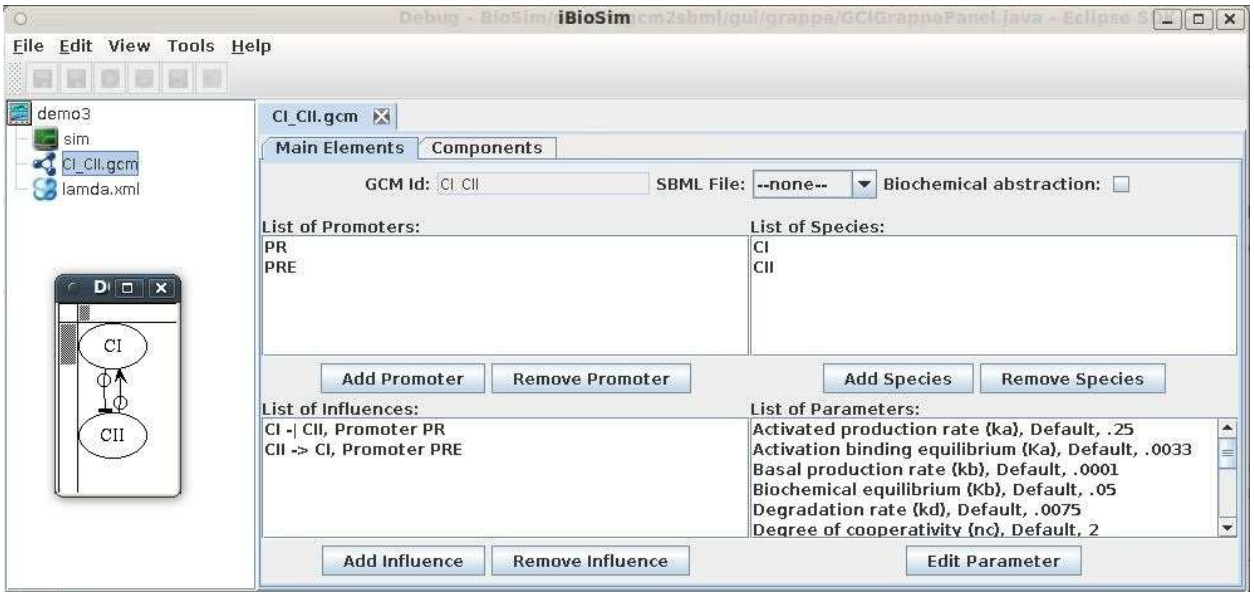

Fig. 1. iBioSim before the project was undertaken.

II. Genetic Circuit Model

Researchers often use chemical reaction networks to describe genetic circuits. These networks can be tedious because each chemical species and reaction must be explicitly defined [7]. Nguyen et al. created the Genetic Circuit Model (GCM) to allow researchers to work at a higher level and only specify relevant species and the relationships those species have with each other [8][9][7].

A Genetic Circuit Model (GCM) is a simplified data structure used to represent the function of a synthetic biological circuit. It contains lists of promoters, species, components, influences, and parameters. A *promoter* is a region on a strand of DNA that enables a particular gene to be transcribed. The transcription of a gene ultimately results in the creation of proteins or *species*. These species can have *influences* on each other in that they bind to other promoters, activating or repressing the production of their respective genes. *Components* are a construct enabling the use of hierarchy by allowing GCMs to import other GCMs and make connections to them through ports. *Parameters* in the GCM file and on individual promoters, species, and influences are used to track production and degradation rates, equilibrium constants, and other values relevant to the genetic circuit.

More formally [8][9][7], a GCM is a tuple $\langle S, Pr, M, N, O, T, G, I, C, V_g, A_g \rangle$ where:

- S is a finite set of species (i.e., proteins);
- Pr is a finite set of promoters;
- $N \subseteq S$ is a finite set of inputs;
- $O \subseteq S$ is a finite set of outputs;
- $M = (M_0, \ldots, M_n)$ is a finite set of other instantiated GCMs;
- $T \subseteq S \times \bigcup_{i=0}^{n} (N_{M_i} \cup O_{M_i})$ is a port mapping of species to ports on instantiated GCMs;
- $G : Pr \mapsto 2^S$ maps promoters to sets of species;
- $I \subseteq S \times Pr \times \{a, r\}$ is a finite set of influences;
- $C \subseteq S \times \mathbb{N} \times S$ is a finite set of complex formations;
- V_g is a finite set of variables;
- $A_g \subseteq (V_g \times \mathbb{R})$ is the assignment of the variables with defaults;

A simple GCM is shown in figure 2 in which two promoters in P activate the production of two species in S by facilitating the binding of RNA Polymerase (RNAP) to the strand of DNA. This allows the DNA sequence to be translated into mRNA which will then be transcribed into the respective protein species. figure 3 shows how the GCM simplifies the design of genetic circuits by hiding some of the low level details.

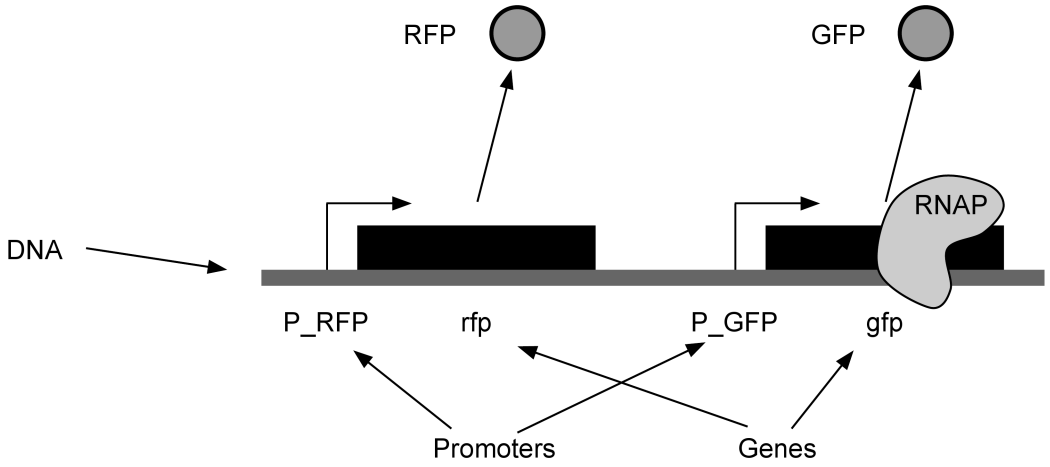

Fig. 2. A simple Genetic Circuit in which two promoters (P_RFP and P_GFP) activate the production of two respective species (RFP and GFP).

Fig. 3. A GCM representing the genetic circuit. Not shown are parameters dictating the production rates, initial species counts, species degradation rates, etc.

III. Schematic Capture of Genetic Circuit Models

Selecting the add species button allows the user to add new species to S by dropping them onto the schematic at a desired location as in figure 4. A species will be created where the user clicked with a default name such as “S1” or “S2” where the number automatically increments with each species created. Similarly to add a promoter to P the user should make sure the promoter button is selected and then click an empty space on the schematic. Adding a component to M is again similar, except that a menu will be displayed allowing the user to select the GCM file that the component should represent.

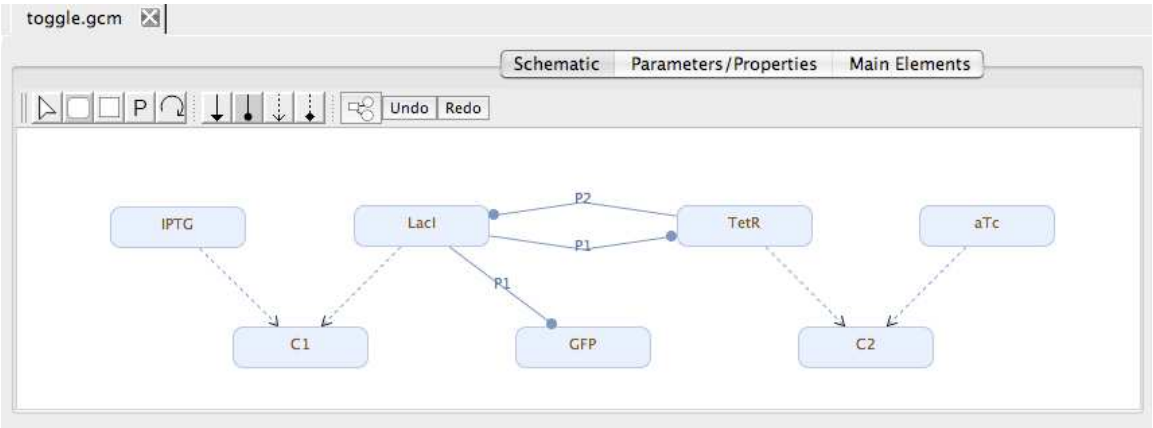

Fig. 4. The user interface for the schematic capture mode of iBioSim. The first set of five buttons are mutually exclusive and are from left to right, selection, drop species, drop component, drop promoter, and create self-influence. The next set of four buttons represent the type of influence to be created. The next button provides an interface to automatically lay out the graph using a number of layouts such as tree, circle, and organic. Below the toggle buttons is the schematic of the GCM being created or modified by the user.

Anything in the schematic view can be selected by clicking on it. Once a species, promoter, or component is selected it can be moved around on the schematic by clicking on it near the edge and dragging it to a new location. Right-clicking on a species, component, or promoter causes a drop-down menu to appear in which the user has the option to delete the object.

Clicking and dragging from the center of the species, promoter, or component will cause an arc to be drawn, and if the mouse is released over a valid promoter, species, or component, a new influence or connection will be created. This is the way that influences are created. The user can choose from several toggle buttons to determine the type of influence that will be created such as activation or repression. Influences are created when the user drags an arc from a species to another species, or from a species to a promoter. Self-influences can be drawn on species by selecting the self-influence toggle button.

Arcs can be drawn from promoters to species, and when this is done a “production” arc is created. Biologically this means that the production of the species is facilitated by the promoter. Arcs drawn between species will have a default promoter created. This promoter will not be shown explicitly on the schematic, except as a label on the influence. Arcs drawn to or from components create connections to ports in the component. A connection between a species and a component means that the species both in the component and the species that is connected to the component will always have the same concentrations or counts.

Once created, any promoter, species, component, or influence object can be selected and modified by double clicking on it. Upon being double clicked a window will appear with all the object’s relevant parameters. The window used to modify influences connected between species allows the user to change the influence’s promoter. The choosing of the promoter is limited to those promoters which are not explicitly drawn.

The example in figure 4 demonstrates many of the features of the schematic capture utility. The figure shows a genetic toggle switch in which either the concentration of TetR or LocI is high at any given time. The system is toggled by increasing the concentration of IPTG or aTc which bind with LacI and TetR respectively through complex formations, shown by dashed lines. This binding lowers the concentrations of IPTG or aTc by sequestering them away in the C1 and C2 complexes, thus preventing them from taking part in repression.

IV. Visualization of Simulated Genetic Circuit Models

A GCM can be translated to SBML as described in [8][9] allowing the genetic circuit to be analyzed using *ordinary differential equation* (ODE) or stochastic simulation. The output of these simulators in iBioSim is a table of time series data (TSD) that has columns of species and rows that are values of these species at increasing time steps. The iBioSim tool provides a useful graphing interface to visualize the TSD output of these simulation runs by displaying the data as a series of lines plotted with time along the independent axis and concentrations or counts of molecules on the dependent axis. This method of visualizing data is effective when looking at small numbers of signals, but does not work as well when looking at large numbers of signals simultaneously.

We wanted to provide effective visualization of many circuits running simultaneously, each inside it's own bacteria cell. We wanted to display the results of the simulation by changing the colors of species and components directly on the schematic. This would allow the user to see "at a glance" the state of the simulation. It also allows the user to see a visualization similar to what a researcher would see when looking at living cells through a microscope.

To facilitate this addition a "movie" mode was added to the schematic where "movie" is the term we use to describe the playing back of a simulation run directly on the schematic. In this mode all controls to modify a schematic are disabled or removed and are replaced with movie player controls such as "Play/Pause" and "Stop", as well as controls to adjust the visual presentation of the movie.

Figure 5 demonstrates an example of a movie being played. There are movie playback controls to play and pause, rewind, single step, and a slider to take the move to any time-slice in the simulation data. As the movie plays at every time-slice the colors, sizes, or opacities of each component are updated to reflect the move data at that moment. Figure 6 shows the interface designed to choose mappings between simulation values and the appearance of components. Up to three species or subspecies inside a component can be chosen, each mapping to a different color gradient. The color gradients are then combined during movie playback. The size of the component can also be in place of a color gradient, allowing the growth of components to be visualized if the components' volume is changing. Opacity can also be mapped allowing species or components to disappear and reappear based on data in the TSD file. Figure 7 shows a photograph of e-coli cells that were programmed with a genetic circuit similar to the one being simulated [10] in figure 5.

There are "min" and "max" values associated with every mapping (see figure 6). These values are used to define how the simulation data is mapped to the color gradient, size, or opacity. Allowing the

Fig. 5. The visualization mode that was added to iBioSim. In this simulation many identical cells were tiled then simulated. Each cell contains the GCM defined in figure 3 in which promoters code for both red and green fluorescent proteins. Although genetically identical, due to randomness in the biological processes each cell exhibits a different final color as these fluorescent proteins combine.

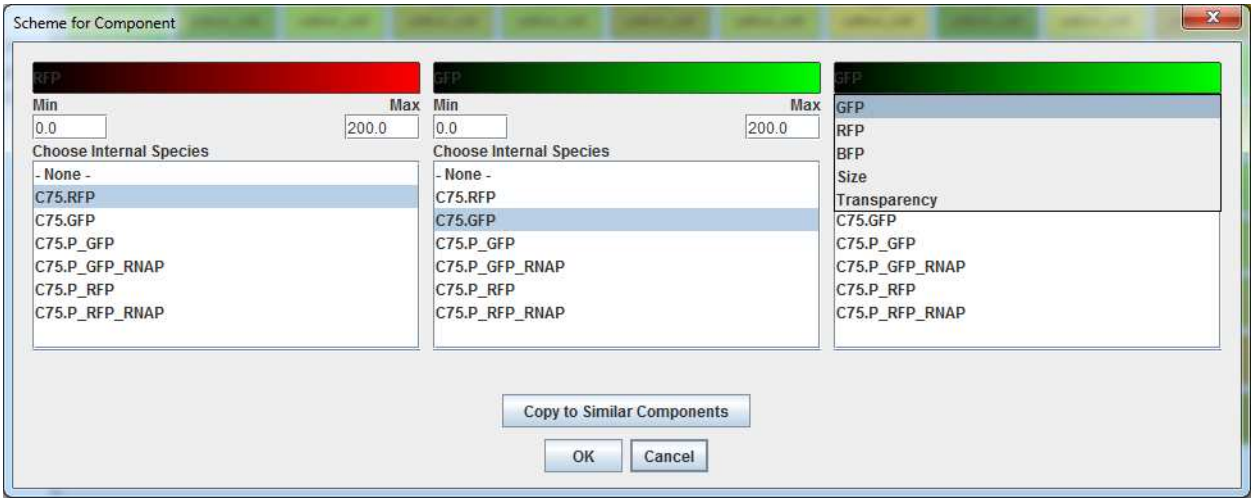

Fig. 6. The menu panel built to choose how colors, sizes, and opacity is mapped to components. Up to three different species in a component or sub-component can be mapped to color gradients, transparency, or sizes.

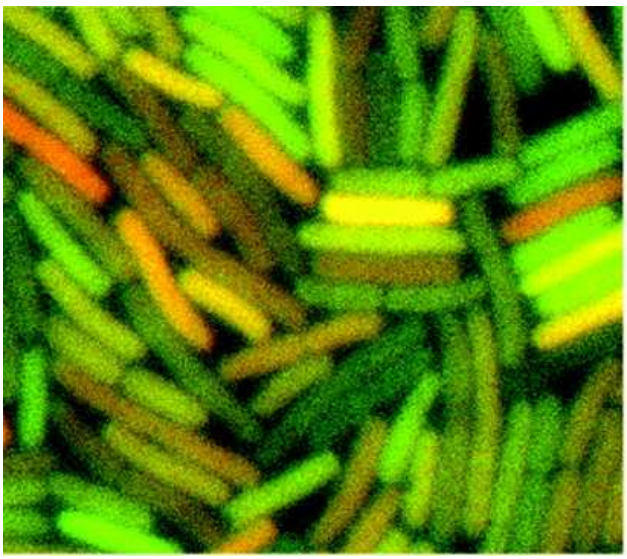

Fig. 7. An experiment in which e-coli bacteria were genetically altered to produce both red and green fluorescent proteins [10]. Note the similarities in colors between the simulation and the experiment.

user to specify min and max is a solution to the problem of mapping a nearly unbounded range of simulation values to the limited range of possible color, size, and opacity values. Simulation values such as concentrations or numbers of species can vary widely and with no upper limit, especially since the units of these values are user-definable in iBioSim.

At one point we considered having the software automatically scan the TSD file when it was loaded and set the min and max values for each column of data. We decided against this because it would have made species having different scale ranges appear similarly in the animation. For instance, one species that had a max of 10 would have appeared just as bright or large as a species that had a max of 100. This could have been misleading to a user of our software that didn't realize what had happened. By making the user set the min and max explicitly he or she can still make these species appear the way described, but because the user has made the mapping himself or herself the user should be aware of the differing scales. The equation used to map simulation values to appearance parameters is as follows:

$$C_{rgb} = clamp\left(\frac{s - min}{max - min}\right)$$

$$C_s = clamp\left(\frac{s - min}{max - min}\right)$$

$$C_o \quad = \quad clamp\left(\frac{s - min}{max - min}\right)$$

where C_{rgb} is the output color for each of the red, green and blue color channels. C_s is the component size as a ratio of the size the user set it to where 0 has 0 width and height and 1 is the full size of the component. C_o is the opacity where 0 is fully transparent and 1 is fully opaque. s is the value of the sample in the simulation file for the given time-step and min and max are the values the user chose from the user interface. For any color channel, size, or opacity that doesn't have a mapping, that channel will be ignored. The $clamp()$ function ensures keeps values within the range of [0, 1]:

$$clamp(x) = \begin{cases} 0: & x < 0 \\ x: & 0 \leq x \leq 1 \\ 1: & x > 1 \end{cases}$$

A "Copy to Similar Components" button was added as a convenience to apply the current settings to all components in M where the elements in the component's tuple match those of the component being edited. This is useful when the user has many of the same component and does not want to have to click on every one to set its properties individually. This feature was to apply appearance settings to all the components in figure 5.

V. Discussion

A new schematic capture facility was added to iBioSim that allows a researcher to create biological circuits graphically using a point and click interface. This makes it possible to build models much faster then was previously possible. The schematic capture functionality supports promoters, chemical species such as proteins, biological influences, and hierarchical design through the use of components. A new visualization system was also created that allows simulations to be animated like a movie directly on the schematic by modifying the colors and sizes of species and components according to iBioSim's simulation data.

Much of the work we have been doing has been paving the way toward the creation and simulation of large populations of cells. For instance, the visualizer that was created allows for the visualization of components which could model subsections of a biological process like individual inverters in a ring oscillator, but which could also model entire cells. We implemented a tiling feature that will automatically tile a grid of components onto the schematic as a step toward creating a petri dish full of cells. This automatic tiling allows the researcher to see the simulation results of many independent cells simultaneously, but provides no easy way to simulate cells that are interacting through chemical signals passed through the common medium. Much work can also be done to simulate biological circuits running in cells that are undergoing life processes such as cell division, movement through a medium, and death.

The tool spoken of here (`iBioSim` [5]) is freely available at
`http://www.async.ece.utah.edu/iBioSim/`.

APPENDIX I

Construction of the Schematic Capture Module

The user interface to build GCM was built on top of the JGraphX library [11]. This is a general Java library for manipulating graphs consisting of nodes and edges. All nodes and edges are objects of JGraphX's mxCell class. mxCells that are edges can be connected to nodes via references to other mxCells. All mxCells support Cascading Style Sheet (CSS) type styling attributes that control the color, shape, and other visual aspects of the nodes and edges. All mxCells also support a reference to any arbitrary data. This was helpful and allowed the mxCells to directly track the GCM objects (species, promoters, etc) that they represented.

The JGraphX objects emit numerous events based on user interaction (such as mouse click and release), as well as higher level events such as when an mxCell is created, moved, or deleted. The JGraphX library implements many useful features such as node and edge selection, dragging, and deletion. These events were used to mimic these changes in the GCM. Mouse click events were used to pull up relevant options menus when the user clicks on a node or edge representing a species, influence, etc.

One design decision that was made involved keeping the JGraphX objects synchronized with the GCM that they represent. Initially code was written so that any time the modified the GCM or the graph, analogous changes would be carefully made in the other structure. For example, if the user clicked a button to change an influence from activation to repression, the influence would be changed in the GCM, then the correct edge would be found in the graph and modified accordingly. It was found however, that keeping these structures in sync was tedious and error prone as all modifications had to be mirrored in two places, although the data structures (GCM and JGraphX) were very different.

In addressing this problem it was noted that there was already code to parse a GCM object and build an JGraphX object out of it. This code was leveraged and the new policy is that every change is made directly to the GCM object, and then the JGraphX is rebuilt from scratch. This made it very straightforward to keep the GCM and JGraphX objects in sync, and simplified the code greatly. The downside is a decrease in UI responsiveness because the expensive operation of rebuilding the graph has to be done more often. This downside has minimal however, as even with the most complex GCMs the author has manipulated the JGraphX regeneration seems to happen instantly.

Another design decision that had to be made was the implementation of an undo-redo interface. It was decided that snapshots would be taken of the GCM object whenever a change was made. Since the GCM implementation was already capable of serializing itself for saving to a file, the snapshots are

simply in-memory strings of the serialized GCM. These snapshots are then stored on two stacks, one for undo and one for redo. When the user presses the undo or redo button a string is popped off the respective stack, the GCM object is reconstructed from it's serialized string representation, and the graph is instructed to refresh itself.

Construction of the Appearance Modules

One significant design task encountered in this project was the creation of a system to capture and store the user's mappings of values to colors, sizes, and opacity (termed appearances, or appearance parameters). It was desired that during playback of a simulation the species in a GCM could change color and size depending on their values, and that the colors and sizes of components could change based on the values of some number of user defined internal species. More specifically, the requirements were as follows:

1) For every species in S an interface must be provided allowing the user to choose whether to map the species' value to a color gradient, size, or opacity. A way must also be provided to set the min and max parameters.
2) For every component in M it should be possible for the user to choose several species in M_S, and for each species in M_S chosen the same options should be available as are for species (item 1).
3) All the user's appearance preferences must be serializable and deserializable so that they can be written to and read from a file.

Several different data structures were considered to satisfy these constraints. One such structure was Java's Properties object, which is a dictionary mapping string keys to string values. This structure is used by iBioSim to keep track of the mappings for each of the values in the GCM tuple, and because it is already used in the iBioSim it seemed at first to be a good choice. On closer inspection however this structure was not a good fit because it is inherently flat, while the data structure being represented was recursive in nature. A dictionary could still have been used if it's keys had been constructed out of many concatenated string values, but this would have complicated the code and made it less sustainable as maintenance and additional features were needed in the future. Instead of using dictionaries another data structure was designed as shown in figure 8.

The data structure is used to track the way simulation data maps to appearance (size, color, and opacity) attributes. A dictionary mapping species in S to SpeciesScheme objects, and components in M to ComponentScheme objects is kept. SpeciesScheme defines a mapping from one species' simulation data to one appearance attribute. ComponentScheme contains zero or more ComponentSchemePart objects which also map one species' simulation data to one visual attribute. Since several ComponentSchemePart

objects can belong to one component, these appearance attributes are combined at playback time.

The TSDKey attribute in ComponentSchemePart is a reference to a column in a simulation file. This can reference a species directly in M, or a species inside a component in M, up to any arbitrary depth. Each TSDKey is a string of the form $A\_\_B\_\_C...\_\_N$ where A is the name of a component in M, B is a component in A, and C is a component in B. The final entry is a species in S in the final component listed. Double underscores are a convention used to distinguish items in the string. To ensure that double underscores are reliable delimiters between species, no species or component name may include double underscores or leading or trailing underscores.

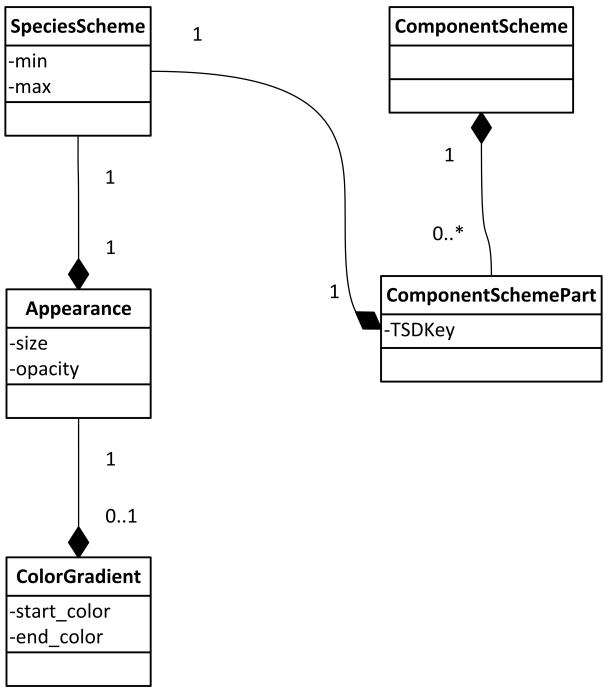

Fig. 8. The data structure used to map simulation data to visual attributes such as color, size, and opacity.

This structure was found to track all the data needed for the movie appearance mappings. In order to serialize and deserialize the data for storage and retrieval the java library GSON was used [12]. To use this library an instance of a Gson object is created, then the java object to be serialized is passed to the gson.toJson() function which returns a string containing JSON formatted data. This data is then written to a file. To reconstruct the original object the contents of the file in String form, along with the class of the object to be created are passed into gson.fromJson() which returns the deserialized object. The Gson library was found to be very effective and easy to use for this data structure.

The final portion of the appearance modules was the user interface. The interface to choose a component's movie apperances is shown in figure 6. The internal structure of the user interface is very similar to that of the data structure, and has separate classes allowing the user to modify the SpeciesScheme, ColorScheme, Appearance, TSDKey, and ComponentScheme objects. Composition (one class containing another class as a member variable) is used to enhance the maintainability of the project by preventing the duplication of code.

REFERENCES

[1] J. B. LUCKS and A. P. ARKIN, “Synthetic biology’s hunt for the genetic transistor,” *IEEE Spectrum*, March 2011.

[2] Y. Cai, M. L. Wilson, and J. Peccoud, “GenoCAD for iGEM: a grammatical approach to the design of standard-compliant constructs,” *Nucleic Acids Res.*, vol. 38, no. 8, pp. 2637–44, 2010.

[3] D. Chandran, F. T. Bergmann, and H. M. Sauro, “TinkerCell: modular CAD tool for synthetic biology,” *J. Biol. Eng.*, vol. 3, no. 19, 2009.

[4] M. A. Marchisio and J. Stelling, “Computational design of synthetic gene circuits with composable parts,” *Bioinform.*, vol. 24, no. 17, pp. 1903–10, 2008.

[5] C. J. Myers, N. Barker, K. Jones, H. Kuwahara, C. Madsen, and N.-P. D. Nguyen, “iBioSim: a tool for the analysis and design of genetic circuits,” *Bioinform.*, vol. 25, no. 21, pp. 2848–2849, 2009.

[6] M. Pedersen and A. Phillips, “Towards programming languages for genetic engineering of living cells,” *J. R. Soc. Interface*, vol. 6, no. (Suppl. 4), pp. S437–S450, 2009.

[7] C. Madsen, C. Myers, N. Roehner, C. Winstead, and Z. Zhang, “Utilizing stochastic model checking to determine the robustness of genetic circuits,” 2011.

[8] N. Nguyen, “The design of a genetic muller c-element,” Master’s thesis, U. of Utah, 2008.

[9] N. Nguyen, C. Myers, H. Kuwahara, C. Winstead, and J. Keener, “Design and analysis of a robust genetic muller c-element,” *Journal of Theoretical Biology*, vol. 264, no. 2, pp. 174 – 187, 2010.

[10] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain, “Stochastic gene expression in a single cell,” *Science*, vol. 297, pp. 1183–1186, August 2002.

[11] J. Ltd., “Jgraphx, a java graphing library.” [Online]. Available: www.jgraph.com

[12] inder123, joel.leitech, and limpbiz...@gmail.com, “google-gson, a java library to convert json to java objects and vice-versa.” [Online]. Available: http://code.google.com/p/google-gson/