JigCell Model Connector: building large molecular network models from components

Thomas C Jones Jr1, Stefan Hoops3, Layne T Watson1, Alida Palmisano1,2, John J Tyson2 and Clifford A Shaffer1

Abstract
The growing size and complexity of molecular network models makes them increasingly difficult to construct and understand. Modifying a model that consists of tens of reactions is no easy task. Attempting the same on a model containing hundreds of reactions can seem nearly impossible. We present the JigCell Model Connector, a software tool that supports large-scale molecular network modeling. Our approach to developing large models is to combine smaller models, making the result easier to comprehend. At the base, the smaller models (called modules) are defined by small collections of reactions. Modules connect together to form larger modules through clearly defined interfaces, called ports. In this work, we enhance the port concept by defining three types of ports. An output port is linked to an internal component that will send a value. An input port is linked to an internal component that will receive a value. An equivalence port is linked to an internal component that will both receive and send values. Not all modules connect together in the same way; therefore, multiple connection options need to exist.

Keywords
computational systems biology, hierarchical model composition, JigCell, modeling tool, SBML, software

1. Introduction
The functions of a living cell are controlled by macromolecular interactions. These complex interactions between genes and proteins can be mapped as regulatory networks. In an effort to understand the dynamic properties of these networks, mathematical models of the biochemical reactions are often constructed.1–3 During this process, modelers have the difficult task of specifying reaction details between species connected in these complex regulatory networks.

Modeling a system accurately is an iterative process involving frequent changes to the underlying network.4 Once a model is drafted, the equations can be analyzed and simulated to describe the dynamical behavior of the regulatory network.2 These computational results can then be compared to existing experimental data, and if inconsistencies arise, the model can be modified. Once a model has been tested against existing experimental data, it can be used to make predictions that might guide the direction of future experiments.5 If further experiments uncover inconsistencies, then the model can be modified again.

As molecular biologists discover more information about how gene and protein interactions affect cell physiology, the size and complexity of the mathematical models tend to grow. Constructing these models is becoming more difficult. Historically, modelers have been limited in the scope of the behavior being modeled by the complexity of larger models. For these reasons, modelers have a need to deal with increasingly complex models, which require new modeling approaches.

Hierarchical model composition is a modeling technique where, instead of building one large, complex model, smaller models are combined together to form a larger model. By breaking a complex system into smaller parts, with clearly defined interactions between the parts, it can be more easily understood.

In this paper we present the JigCell Model Connector (JCMC), a software tool to support hierarchical model

1Department of Computer Science, Virginia Tech, Blacksburg, VA, USA
2Department of Biological Sciences, Virginia Tech, Blacksburg, VA, USA
3Biocomplexity Institute, Virginia Tech, Blacksburg, VA, USA

Corresponding author:
Clifford A Shaffer, Department of Computer Science, Virginia Tech, Blacksburg, VA 24061, USA.
Email: shaffer@cs.vt.edu.
composition. In our tool, the smallest models (called modules) are defined by small collections of reactions. Modules connect together to form larger modules through clearly defined interfaces, called ports. Modelers are able to regulate external access to internal components of a module by utilizing ports. We implement three port types that allow modules to connect in different ways. An output port is linked to an internal component that will send a value to an external reference. An input port is linked to an internal component that will receive a value from an external reference. An equivalence port is linked to an internal component that will both receive and send values from an external reference. Once a model is created in the JCMC, it can be exported to one of the existing standard model formats, at which point the model can be simulated and analyzed using other tools. Our goal is to develop large models in a modular way, making the result easier to comprehend. A major contribution of this paper over prior efforts to define systems for building models in a modular fashion are new port and node constructs that allow modules to connect in different ways using the JCMC.

The remainder of this paper is organized as follows. First, previous work on hierarchical model composition standards and tools is reviewed. Next, we present the JCMC environment and its components. Then the different types of ports and their impact on a model are discussed. Last, best practices for constructing hierarchical models, conclusions, and future work are presented in the form of a case study.

2. Background

In this section, we discuss the basic concepts of hierarchical modeling, and a standard format that enables the sharing of models. We also review previous attempts to create hierarchical modeling tools.

2.1. Hierarchical model composition

Randhawa and colleagues describe model composition as another approach to create large models from smaller models. The smaller models become submodels of a larger composed model. Composition involves describing how components from different submodels interact with one another, without changing the inner workings of each submodel. The interaction descriptions are stored in the overarching composed model. Here, large models are simply collections of submodels, and can be organized in a hierarchical fashion. Unlike fusion, model composition is a reversible process. If the interaction descriptions are removed, then the original submodels can be recovered.

Randhawa and colleagues also present the concept of model flattening. Model flattening converts composed or aggregated models to their “flattened versions.” The interaction details of the composed or aggregated models are used as instructions during the flattening process. The result is a single large (flat) model, which is equivalent to fusing the submodels. The flat model is in a standard format that can be read by existing software tools for the purpose of running simulations and further analysis. The flattening process loses the hierarchical and other relationships between the various modules, yielding a set of reaction equations.
2.2. The SBML standard

Systems Biology Markup Language (SBML) is a format that represents systems biology models electronically. Such a standard allows for the sharing and collaboration of models. SBML Level 1 was introduced in 2001. A model defined in SBML can consist of many components, such as compartments, species, reactions, and parameters. The interactions between components in the model are also defined in SBML. SBML is not designed with the goal of being an easily human-readable format. Modelers are not expected to write their models by hand in SBML. Instead, software tools are expected to read and write the format.

Since SBML supports the concept of defining models as consisting of many components, it already includes many of the features necessary for hierarchical modeling. For those hierarchical modeling features that do not exist within the SBML standard, extra information about a component can be stored in annotations. Annotations can be thought of as comments in an SBML file. These comments can be associated with any SBML component, such as a reaction, species, or parameter. Software that processes standard SBML models can also be augmented to process this additional information carried in the annotations. Therefore, annotations can be used by software developers to include application-specific data.

The latest version of SBML, Level 3 Version 1 Core, was released in 2010. This version of SBML incorporates new features that support hierarchical model definitions. Such feature extensions are referred to as packages. Two SBML packages relevant to component-based modeling, comp and layout, are described next.

2.2.1. The SBML comp package. The latest version of the SBML comp package was released in 2013. This package allows instances of models to be incorporated as submodels within a model. The model structure is extended to include a list of submodels and a list of ports. A submodel is an instance of a model definition, which in turn is a complete, self-contained model. Model definitions (instantiated as submodels) are located either in the list of internal model definitions or the list of external model definitions. An internal model definition is stored within the SBML file. An external model definition is a placeholder that specifies the location of an external file containing the model definition. This external file can be on the local machine or available on the internet. Ports allow models to interact with other models through a designated interface. A port references some component within the model that is being explicitly exposed, such as a species or parameter. These extended features of comp enable hierarchical model composition in SBML.

2.2.2. The SBML layout package. The latest version of the SBML layout package was released in 2013. The layout package allows components of a model to be represented graphically, along with their positions. To do so, the model structure is extended to include a list of layouts. A layout can store information specifying the graphics representing some or all components of the SBML model. These graphics are referred to as glyphs in the layout package. A compartment, species, or reaction can be represented by a CompartmentGlyph, SpeciesGlyph, or ReactionGlyph, respectively. A GeneralGlyph can be used to represent parts of a model that are not specified in the Level 3 Version 1 Core, such as a submodel from comp. A glyph stores information pertaining to the location and dimension of a graphical object. A glyph does not include information describing the shape, color, or style of a graphical object; it is left up to the software tool reading the layout to display such details. These extended features of layout enable model visualization in SBML, and these features are used heavily by JCMC.

2.3. Related tools

There are numerous software tools available for the modeling and simulation of molecular networks. For example, Antimony is a model definition language that can be used to create, import, and combine models in a modular way. However, Antimony is text-based and only provides limited support for importing/exporting models using SBML comp.

Similar to Antimony, Genetic Engineering of living Cells (GEC) is a text-based formal language. GEC allows interactions between proteins and genes to be expressed in a logical manner. GEC programs can be used to create models in a modular way. However, GEC does not support exporting models using SBML comp.

COPASI is a tool used to model, simulate, and analyze biochemical networks. Its graphical user interface offers many features such as stochastic and deterministic simulation methods, parameter estimation, and data visualization. COPASI is an excellent tool for creating a single model, and it provides support for importing/exporting standard SBML (Level 3). However, COPASI lacks features to support hierarchical modeling and SBML comp.

TinkerCell is a tool that supports hierarchical modeling. The graphical user interface lets users create modules that can be connected together to form larger models. TinkerCell does not have a notion of port types and does not support SBML output.

The JigCell suite of tools can be used to model, simulate, and analyze biochemical networks. Previous iterations of JigCell have included a Model Builder, Aggregation Connector, Run Manager, Comparator, and Parameter Estimation Toolkit. The Model Builder is used to create and edit reactions, species, and other model properties in a tabular format. The Aggregation Connector is used to combine models in a modular way. The Run
Manager and Parameter Estimation Toolkit are used to define simulation properties and determine unknown parameter values within the model. The Comparator is used to compare the model simulations with experimental results. The JigCell suite provides support for importing/exporting standard SBML (Level 2).

JigCell Multistate Model Builder (JC-MSMB) is a tool that supports the modeling of biochemical networks. The graphical user interface builds on the tabular spreadsheet format used by Vass and colleagues. JC-MSMB reduces the complexity of model creation by introducing a new syntax to describe multistate species. This syntax requires fewer reactions to represent complex molecular systems. The tool has many editing support features such as flexible autocompletion and consistency checks to assist users during the model-creation process. It provides support for importing/exporting SBML (Level 3). However, JC-MSMB lacks features to support hierarchical modeling and SBML comp.

The current work on JCMC can be viewed as a new generation beyond the JigCell Aggregation Connector, working in connection with the JigCell Multistate Model Builder.

iBioSim is a tool for the modeling, analysis, and design of genetic circuits. In synthetic biology, genetic circuits can be used to design and construct networks to implement a particular cellular function. Although primarily designed for genetic circuits, it can be used to study biological networks as well. Its graphical user interface can be used to create, import, and combine models in a modular way. iBioSim offers multiple simulation methods, model analysis, and data visualization, along with support for importing/exporting hierarchical models using SBML comp. Both JigCell’s Aggregation Connector and iBioSim offer support for SBML and hierarchical modeling. However, JigCell lacks the features of different port types. iBioSim does offer input and output port types, but does not support equivalence ports.

2.4. Model simulation

Once a model has been created, it is typically used for simulation and further analysis. There are multiple approaches to simulate a hierarchical model. One option is to first flatten the model. Once flattened, copies of all submodels have been instantiated and replacements and deletions have been applied, the model is no longer in its hierarchical form. After flattening, the model is simulated. COPASI is an example of a tool that uses this approach.

3. JigCell Model Connector

We next describe the JCMC in detail. In order to better understand the purpose and key features of the JCMC, it helps to first have an overview of the system’s user interface. At this point, the reader need not worry too much about the underlying meaning of the various components that are presented here. This will be discussed later.

3.1. Interface

The JCMC interface consists of three panels, shown in Figure 2.

3.1.1. TreeView. The left panel contains the TreeView. This displays the hierarchical relationships between components of the model, somewhat like a traditional file folder display. An example is shown in Figure 3. A module can be selected from the TreeView using the left mouse button, and it will be highlighted. Double-clicks (using the left mouse button) will expand/collapse the selected module. After selecting a module, the user can add or remove submodules (using the Module menu).

3.1.2. DrawingBoard. The right panel contains the DrawingBoard. This displays the graphical view of a module, its submodules, and any connections among them, as shown in Figure 4. The currently loaded module is called the container module. In Figure 4, the container module is named RegulationExample. Submodules can be moved encountered. iBioSim is an example of a tool that uses this approach.
inside of the container module. In Figure 4, Cdh1 and CycB are submodules. If ports exist, they are displayed on the container module and submodules. Connections between ports, visible variable nodes, and equivalence nodes are also shown. Examples of these components are presented in later sections.

3.1.3. ModelBuilder. The bottom panel is the ModelBuilder, shown in Figure 5. The ModelBuilder in JCMC is actually a version of the JC-MSMB,24 which was previously implemented by our group. The version of JC-MSMB used by JCMC is a subset of the complete model editor because JCMC does not support multistate species. JC-MSMB has a tabular spreadsheet interface that displays the details of a module. Attributes such as reactions, species, parameters, and events can be modified.

4. Components

4.1. Container module

The container module is the module currently loaded into the DrawingBoard. The TreeView panel shows the container module’s name in bold font. The ModelBuilder panel displays the module definition for the container module. A module definition contains a module’s detailed information, such as reactions, species, parameters, and events. The DrawingBoard displays the container module and any submodules, ports, or connections in the module.

4.2. Submodule

A submodule is simply a module contained within another module. The TreeView panel lists a submodule under the container module to which it belongs. In the DrawingBoard, a submodule can be moved and resized within the bounds of its container module. A submodule’s information is listed as:

```
< Definition Name >
```
```
" < Submodule Name > "
```

Definition Name corresponds to the name of the module definition. A module definition contains detailed information, such as reactions, species, parameters, and events. Submodule Name corresponds to the name of a specific instantiation of the module definition. In Figure 4, submodule Cdh1 is an instantiation of module definition PhosDephos. Similarly, submodule CycB is an instantiation of module definition SynDeg. A single module definition can be instantiated multiple times. We show examples of this in later sections.

A submodule’s detailed information (reactions, species, parameters, events, etc.) is not listed in the ModelBuilder panel because the container module’s information is displayed instead. However, a submodule’s information can be previewed in the ModelBuilder panel. Each submodule has a button in the top left-hand corner. When this button is clicked, the information for that submodule will be...
displayed in the ModelBuilder. An example is shown in Figure 6. After the button for Cdh1 is clicked, the template information for Cdh1 is displayed in the ModelBuilder pane. Notice that the tables are grayed-out in Figure 6. This is because the information is a preview only, and cannot be modified. To modify the information, the user would load the submodule as the container module.

Submodules can be added and removed using the Module menu. Removing a submodule will remove the selected module, all of its submodules, and any connections associated with other modules.

5. Technical contributions

As discussed in Section 2, Randhawa and colleagues define model aggregation as a restricted form of model composition. With aggregation, modelers can regulate external access to internal components of a module by defining ports. A port allows an internal component to be referenced in a larger composed model. Modules can then be connected together by their interface ports to build larger models in a controlled manner. However, not all modules are the same. Internal components linked to ports do not necessarily serve the same purpose for every module.

5.1. Ports

Port types supported by JCMC are as follows. An output port is linked to an internal component that will send a value to an external reference. The component linked to the port may be modified inside the module, but the component is not meant to be modified outside the module. Consider the scenario in which a species is synthesized in a module and then used as a transcription factor outside of the module. An output port is appropriate because the species is not modified outside of the module. A detailed example is presented in Section 6.1. Output ports are represented as triangles on the edge of modules. They are oriented so the arrowhead points out of the module.

An input port is linked to an internal component that will receive a value from an external reference. The component linked to the port is not meant to be modified within the module. Consider the scenario in which a rate constant for a reaction within a module has a value determined outside of the module. An input port is appropriate because the rate constant is only used in calculations for the reaction, and not modified inside the module. A detailed example is presented in Section 6.1. Input ports are represented as triangles on the edge of modules. They are oriented so the arrowhead points into the module.

An equivalence port is linked to an internal component that will both receive and send values from an external reference. The component linked to the port may be modified inside and outside the module. Consider the scenario in which a species is synthesized in one module and phosphorylated in another module. An equivalence port is appropriate because the species is modified in both modules. A detailed example is presented in Section 6.2. An equivalence port is represented as a diamond on the edge of a module.

In the DrawingBoard panel, ports are displayed on module boundaries. In the ModelBuilder panel, ports are listed under the Ports tab (shown in Figure 7). The list is populated with ports from the container module and ports from any submodules in the container module. When a port is selected in the DrawingBoard panel, the Ports tab is be linked to either a species or a module quantity. Once created, the ports collectively form an interface to the module, with external access to a module’s internal components being regulated by the interface. Modules can be connected together by their interfaces to build larger models.

Previous software tools support ports, and ports are a part of the SBML comp package. However, this prior work treats all ports the same. Internal components linked to ports do not necessarily serve the same purpose for every module. When the port mechanism gives no information as to how an internal component is used, modelers have no way to discern a component’s purpose within a module. So there exists a need for different port types. The port types supported by JCMC are as follows.

An output port is linked to an internal component that will send a value to an external reference. The component linked to the port may be modified inside the module, but the component is not meant to be modified outside the module. Consider the scenario in which a species is synthesized in a module and then used as a transcription factor outside of the module. An output port is appropriate because the species is not modified outside of the module. A detailed example is presented in Section 6.1. Output ports are represented as triangles on the edge of modules. They are oriented so the arrowhead points out of the module.

An input port is linked to an internal component that will receive a value from an external reference. The component linked to the port is not meant to be modified within the module. Consider the scenario in which a rate constant for a reaction within a module has a value determined outside of the module. An input port is appropriate because the rate constant is only used in calculations for the reaction, and not modified inside the module. A detailed example is presented in Section 6.1. Input ports are represented as triangles on the edge of modules. They are oriented so the arrowhead points into the module.

An equivalence port is linked to an internal component that will both receive and send values from an external reference. The component linked to the port may be modified inside and outside the module. Consider the scenario in which a species is synthesized in one module and phosphorylated in another module. An equivalence port is appropriate because the species is modified in both modules. A detailed example is presented in Section 6.2. An equivalence port is represented as a diamond on the edge of a module.
displayed and the corresponding port is highlighted in the ModelBuilder panel. Each port has three properties:

- Ref Name: the species or module quantity referenced by the port;
- Port Type: the type of port;
- Port Name: the name of the port.

Port additions or removals can only happen to the container module. To modify the ports of a submodule, the user must first load the submodule as the container module.

5.2. Nodes

A node allows connections to occur between module ports. The type of node used depends on the type of ports connected.

5.2.1. Visible variable node. A visible variable node is automatically created when a connection is made between input or output ports of two modules. Figure 8 shows two submodules after a connection has been made, a visible variable was created, and the new variable was added in the ModelBuilder panel. Another way to create a visible variable node is to right-click the active module and select “Show Variable.” Once selected, a pop-up window will appear with a drop-down box that contains a list of all the species and module quantities in the module. The user will select a variable and click “Add,” then a visible variable node will be created in the DrawingBoard panel.

A visible variable node can have at most one incoming connection. A single incoming connection lets the node receive values. A visible variable node can have multiple outgoing connections. The outgoing connections are used to send values.

5.2.2. Equivalence node. An equivalence node is created automatically when a connection is made between an equivalence port and any other port in the DrawingBoard panel. When created, the new variable is added in the ModelBuilder panel. An equivalence node can have multiple connections. Since values are both sent and received, there is no distinction between incoming and outgoing connections.

5.3. Connections

A set of connections link modules together. Connections can occur between the ports of different modules, visible variable nodes, and equivalence nodes. The rules for connections are listed in Tables 1, 2, and 3.

A connection can be created by dragging a line from a valid source to a valid target. Attempting to create an invalid connection will result in a warning message, and no connection will be created.

6. Port, node, and connection effects

In this section we explore the effect that different ports, nodes, and connections have on a model. In the equation notation used below, variables are represented by strings,
multiplication is denoted by $\times$, and differentiation by $\frac{d}{dt}[\text{name}]$. (Throughout this paper, the notation $[\text{name}]$ refers to the concentration of species name.)

### 6.1. Input and output ports

We begin with a simple synthesis and degradation module.

Figure 9 shows how a synthesis and degradation module would look in JCMC. The reactions are displayed in the ModelBuilder panel at the bottom. The rate of synthesis is determined by rate constant $k_0$ and transcription factor $F$. The rate of degradation is determined by mass action kinetics with rate constant $k_1$. The dynamics of species $R$ in module SynDeg are as follows:

$$\frac{d}{dt} \frac{R}{C_{138}} = \frac{(k_0 \times [F])}{C_0} - \frac{(k_1 \times [R])}{C_0}.$$  \hspace{1cm} (1)

Figure 9 also shows that module quantities $k_0$ and $k_1$ are connected to input ports. Because $k_0$ and $k_1$ are connected to input ports, they can receive values from external connections. Note that $k_0$ and $k_1$ are not modified within the module SynDeg, they are only used for the computations of other variables.

Figure 10 shows a phosphorylation and dephosphorylation module in JCMC. The rate of phosphorylation is determined by mass action kinetics with rate constant $k_p$, while the rate of dephosphorylation is determined by mass action kinetics with rate constant $k_h$. The species dynamics of module PhosDephos are defined as follows:

$$\frac{d}{dt} \frac{SP}{C_{138}} = \frac{(k_p \times [S])}{C_0} - \frac{(k_h \times [SP])}{C_0}.$$  \hspace{1cm} (2)

$$\frac{d}{dt} \frac{S}{C_{138}} = -\frac{(k_p \times [S])}{C_0} + \frac{(k_h \times [SP])}{C_0}.$$  \hspace{1cm} (3)

Figure 11 displays Model01, where $S$ and $R$ are submodules. Submodule $R$ is an instantiation of SynDeg, shown in Figure 9. Submodule $S$ is an instantiation of PhosDephos, shown in Figure 10. Species $TF$, module quantity $ks$, and module quantity $gd$ are displayed as visible variable nodes. Submodule $S$ has an output port linked to species $SP$ and submodule $R$ has input ports linked to species $F$, module quantity $k_0$, and module quantity $k_1$. Node $ks$ is connected to the $k_0$ port on submodule $R$. This means that $k_0$ will receive the value of $ks$. To accomplish this, $k_0$ will be replaced by $ks$ in submodule $R$. The same will happen with $k_1$ and $gd$. The replacement is not immediate, but will occur when the entire model is flattened. There is one connection from submodule $S$’s $SP$ port to node $TF$, and another from node $TF$ to submodule $R$’s $F$.
port. Submodule S’s SP port will send its internal value to node TF. Node TF will then send the value to submodule R’s F port. Finally, the internal species F in submodule R will receive the value. When the model is flattened, TF will replace SP in submodule S and F in submodule R. The species dynamics after flattening are as follows:

\[
\frac{d}{dt}[S] = -(kp \times [S]) + (kh \times [TF]), \tag{4}
\]

\[
\frac{d}{dt}[TF] = (kp \times [S]) - (kh \times [TF]), \tag{5}
\]

\[
\frac{d}{dt}[R] = (ks \times [TF]) - (gd \times [R]). \tag{6}
\]

Equation (3) has been replaced by Equation (5) and TF has replaced SP in Equation (2) to form the updated Equation (4). Similarly, TF has replaced F in Equation (1) to form the updated Equation (6).

6.2. Equivalence port

Figure 12 shows the ModelBuilder panel for three different modules.

Figure 12(a) shows the reaction details for the synthesis and degradation of species X in module SynDeg. The rate of synthesis is determined by ks and the rate of degradation is determined by mass action kinetics with rate constant gd. The dynamics of species X in module SynDeg are defined as follows:

\[
\frac{d}{dt}[X] = ks - (gd \times [X]). \tag{7}
\]

Figure 12(b) shows the reaction details for the phosphorylation and dephosphorylation of species Y in module PhosDephos. The rate of phosphorylation is determined by mass action kinetics with rate constant kp. The rate of dephosphorylation is determined by mass action kinetics with rate constant kd. The species dynamics of module PhosDephos are defined as follows:

\[
\frac{d}{dt}[Y] = -(kp \times [Y]) + (kh \times [YP]), \tag{8}
\]

\[
\frac{d}{dt}[YP] = (kp \times [Y]) - (kh \times [YP]). \tag{9}
\]

Figure 12(c) shows the reaction details for the association and dissociation of species Comp in module AssocDissoc. The rate of association is determined by mass action kinetics with rate constant ka. The rate of dissociation is determined by mass action kinetics with rate constant kd. The species dynamics of module AssocDissoc are defined as follows:

\[
\frac{d}{dt}[Z] = -(ka \times [Z] \times [W]) + (kd \times [Comp]), \tag{10}
\]

\[
\frac{d}{dt}[W] = -(ka \times [Z] \times [W]) + (kd \times [Comp]), \tag{11}
\]

\[
\frac{d}{dt}[Comp] = (ka \times [Z] \times [W]) - (kd \times [Comp]). \tag{12}
\]

Figure 13 shows Model02, where X, Y, and Comp are submodules. Submodule X is an instantiation of SynDeg, submodule Y is an instantiation of PhosDephos, and submodule Comp is an instantiation of AssocDissoc. Species A is displayed as an equivalence node. Submodule X has an equivalence port linked to species X, submodule Y has an equivalence port linked to species Y, and submodule Comp has an equivalence port linked to species Z. Each of these equivalence ports are connected to node A in Model02. When the model is flattened for simulation, A will replace X in submodule X, Y in submodule Y, and Z in submodule Comp. The species dynamics after flattening are as follows:
therefore information from each connection is kept. Equivalence ports, values are both sent and received, and Equations (7), (8), and (10). Since node (14) and (15). Notice that Equation (16) is an aggregate of Equations (11) and (12) to form the updated Equations 5.1 are not. In order to store this extra information in however, the different port types introduced in Section 6.3.1. Ports. Ports are included in the comp package. Similarly, A has replaced Y. Similarly, A has replaced Z in Equations (11) and (12) to form the updated Equations (14) and (15). Notice that Equation (16) is an aggregate of Equations (7), (8), and (10). Since node A is connected to equivalence ports, values are both sent and received, and therefore information from each connection is kept.

6.3. SBML syntax

We discussed the SBML standard in Section 2.2. JCMC is able to store the information that describes a hierarchical model by using the SBML comp package. The submodules in JCMC can be stored as model definitions within the SBML file. In this section we explain how the port and node constructs are stored in SBML.

6.3.1. Ports. Ports are included in the comp package. However, the different port types introduced in Section 5.1 are not. In order to store this extra information in SBML, we decided to use SBML annotations and Systems Biology Ontology (SBO) terms. The annotation stores the variable type and variable name by using the tags vType and refName. The SBO term stores the port type. SBO terms can be used and read by other software tools, such as iBioSim.

SBML Syntax in the Supplemental Material shows an example of two ports using the comp package. The layout package does not have a specific glyph to represent ports. It does support a GraphicalObject, which can be used to store general information about an object. We decided to use GraphicalObjects combined with annotations to describe the port layout. The GraphicalObject stores the position of each port. The additional annotation stores the port type, variable type, variable name, variable id, and the name of the module in which the port is located. SBML Syntax in the Supplemental Material shows the graphical information for two ports using the layout package.

6.3.2. Nodes. Visible variable nodes and equivalence nodes are not defined in the comp package. However, a node always represents either a species or a module quantity. Since both species and module quantities are present in SBML, we include node information with an annotation. SBML Syntax in the Supplemental Material shows an example of a visible variable node ClbS and an equivalence node ClbM.

The annotation stores the variable name and node type. The layout package does have a SpeciesGlyph to represent species, but it does not have a specific glyph to represent module quantities. Since a node can be a species or a module quantity, we decided to use GraphicalObjects combined with annotations to describe their layout. SBML Syntax in the Supplemental Material shows an example of the graphical information for visible variable node ClbS and equivalence node ClbM using the layout package.

Similar to ports, the GraphicalObject stores the position of each node. The annotation stores the variable name and variable type.

6.3.3. Connections. In SBML, there is no definition for a connection. Instead, the comp package introduced the notion of replacements. A replacement allows one component to replace another. Replacements are the interaction details used as instructions if the model is flattened. Most connections in JCMC are represented as replacements in SBML. All of our replacements are top-down, which makes them easy to follow and they do not require any rules for resolution. It is important to note that the top-down approach applies to initial values as well. After flattening, initial values may need to be reassigned before simulation.
7. Case study
In this section, we will demonstrate the features of JCMC by building a complex biological model. Barik and colleagues\textsuperscript{29} published a model of yeast cell-cycle regulation, consisting of 58 species and 220 reactions. We will reconstruct this model with a more efficient approach by utilizing modules. We will show how submodules connect together and the role that ports play in the process.

7.1. Biological model
Figure 14 shows a wiring diagram of the biological model from the work of Barik and colleagues.\textsuperscript{29} Species are represented by the labeled shapes. Chemical reactions are represented by solid arrows and enzymatic activities are represented by dashed arrows. Reversible binding reactions are represented by T-shaped arrows with balls on the cross-bars. For clarification purposes, Figure 14 only displays some of the major regulatory interactions contained in the model. For example, the synthesis and degradation reactions for Whi5, SBF, Cdh1, Net1, Hbf, Hi5, and Ht1 are not shown.

The model by Barik and colleagues\textsuperscript{29} captures the molecular controls of cell-cycle events, including the initiation of DNA synthesis (by ClbS) and of mitosis (by ClbM), and “exit” from mitosis, including cell division (by Cdc14). When a mother cell divides, the volume of the cell and the number of molecules of each chemical species within the cell are evenly split between the two resulting daughter cells. (For simplicity, we are ignoring the fact that budding yeast cells divide asymmetrically.) In the model, the event of cell division is triggered by ClbM. When the concentration of ClbM drops below 12 nM, the cell will divide evenly.

7.2. The hierarchical model
First, we will introduce a transcription and translation module that will appear multiple times in our model. Next, we will modularize the model (Figure 15) and build up each module individually. Then, we will connect the modules together to form the final hierarchical model. Finally, we will validate the hierarchical model by comparing simulation results with the original model. In the following descriptions, variables refer to the number of molecules.

7.2.1. Transcription and translation coupling.
From Figure 14 we can see that some of the regulatory functions in the model are similar. The mechanisms regulating ClbM, ClbS, and Cln3 appear to follow the same pattern. The synthesis of the protein is dependent upon the synthesis of its mRNA. This is called transcription and translation coupling. Since it occurs multiple times in the model, we can build it as a reusable, generic module.

Figure 16 shows module TTCoupling, short for transcription and translation coupling. The ModelBuilder panel at the bottom displays the four reactions in the module that describe the synthesis and degradation of mRNA and protein X. Module quantities $k_{sm}$, $g_{dm}$, $k_{sx}$, and $g_{dx}$ are the constants that determine the rates of the four reactions. Protein X is linked to an output port and the rate constants are linked to input ports. The ports allow external proteins and rate constants to connect to the module and utilize the interior transcription and translation reaction structure. We will see how this is done in later sections.

There are also two more input ports for species V and ClbM. These species are used to calculate when the concentration of ClbM falls below 12.5 nM, which is when cell division occurs. Since $V$ is the volume of the cell and...
ClbM is the number of ClbM molecules, they both must be included to calculate the concentration of ClbM. When the cell divides, most species are divided in half. To accomplish this, an event is used. The event calculates the concentration of ClbM to determine when the cell divides. When the cell divides, the species numbers within the module are reassigned appropriately. An event like this occurs in most modules, which is why most modules require V and ClbM. In the TTCoupling module, species X and mRNA are reassigned to half their current values, assuming that the protein and mRNA molecules are divided equally between the two daughter cells.

7.2.2. ClbM regulation. Step 1 of the model will consist of the interactions contained in area A of Figure 15. The regulation of Cdh1 and ClbM play big roles in step 1. Cdh1 has 1 unphosphorylated state as well as 10 phosphorylated states, for a total of 11 phosphorylation states. In this model, only the unphosphorylated state of Cdh1 is active. ClbM affects many parts of this model.

Figure 17 displays module Step01. Step01 contains submodules Cdh1Reg and ClbMTT. Visible variable nodes Cdh1 and Cdh1Pt, receive connections from Cdh1Reg’s output ports Cdh1a and Cdh1i. Cdh1 and Cdh1Pt are used to calculate module quantity gdbmt. The relationship between Cdh1, Cdh1Pt, and gdbmt is not explicitly shown on the DrawingBoard. The calculation defining gdbmt is viewable in the Module Quantities tab of the ModelBuilder panel (not shown in the figure). Visible variable nodes ksmbm, gdmbm, ksbmt, and gdbmt connect the module quantities to ClbMTT’s input ports ksm, gdm, kss, and gdx. Visible variable node ClbM connects the species to ports on both submodules as well as module Step01. Node ClbM receives its value from submodule ClbMTT, where ClbM is regulated. Node ClbM then sends its value to submodule Cdh1Reg, where ClbM influences the phosphorylation of Cdh1. Node ClbM is also connected to Step01’s output port ClbM, so it can be referenced by other parts of the model. Step 01 has three input ports connected to visible variable nodes. These nodes are then connected to submodules, where their values can be used for calculations. Details for the submodules in Step01 can be found in the Supplemental Material.

7.2.3. Cdc14 regulation. Step 2 of the model will consist of the interactions contained in area B of Figure 15. In step 2, the active phosphorylation states of Net1 combine with Cdc14 to form the RENT complex. Ht1 causes dephosphorylation of both Net1 and RENT phosphorylated states.

Figure 18 displays module Step02. Step02 contains submodules Ht1Reg, Cdc14Reg, Net1Reg, and RENTReg. Visible variable node Ht1 connects the species to three different submodules. Node Ht1 receives a value from submodule Ht1Reg, where it is regulated. Node Ht1 sends its value to submodules Net1Reg and RENTReg, where it promotes dephosphorylation. Equivalence nodes for each of the Net1 phosphorylation states connect to Net1Reg and RENTReg because the species are modified in both submodules. Equivalence node Cdc14 connects to submodules Cdc14Reg and RENTReg. Node Cdc14 also connects to an output port on module Step02, so it can be referenced by other parts of the model. Details for the submodules in Step02 can be found in the Supplemental Material.

7.2.4. SBF regulation. Step 3 of the model will consist of the interactions contained in area C of Figure 15. In step 3, the active phosphorylation states of SBF and Whi5 combine to form the Cmp complex. Hi5 is involved with the
dephosphorylation of Whi5 and Cmp, while Hbf is involved with the dephosphorylation of SBF.

Figure 19 displays module Step03. Step03 contains submodules Hi5Reg, SBFReg, Whi5Reg, and CmpReg. Visible variable node Hi5 connects the species to three different submodules. Node Hi5 receives a value from submodule Hi5Reg, where it is regulated. Node Hi5 sends its value to submodules Whi5Reg and CmpReg, where it promotes dephosphorylation. Equivalence nodes for each of the Whi5 phosphorylation states connect to Whi5Reg and CmpReg because the species are modified in both submodules. Equivalence node SBF connects to submodules SBFReg and CmpReg. Node SBF also connects to an output port on module Step03, so it can be referenced by other parts of the model. Step 03 has five input ports connected to visible variable nodes. These nodes are then connected to different submodules, where their values can be used for calculations. Details for the submodules in Step03 can be found in the Supplemental Material.

7.2.5. Final model. We have created all of the modules necessary for our model, so now it is time to put them together. A JCMC user would start by importing all of the modules we have discussed. After the modules are imported, the CellCycle model contains submodules Step01, Step02, Step03, ClbSReg, and Cln3Reg.

Figure 20 displays the completed CellCycle model. Visible variable node ClbS connects the species to three submodules. Node ClbS receives a value from submodule ClbSReg, where it is regulated. Node ClbS sends its value to submodules Step01 and Step03, where it promotes phosphorylation. Visible variable node Cdc14 connects the species to two submodules. Node Cdc14 receives a value from submodule Cln3Reg, where it is regulated. Node Cdc14 sends its value to submodule Step02, where it is regulated and part of the RENT complex. Node Cdc14 sends its value to submodules Step01 and Step03, where it promotes phosphorylation. Visible variable node SBF connects the species to two submodules. Node SBF receives a value from submodule Step03, where it is regulated and part of the Cmp complex. Node SBF sends its value to submodule ClbSReg, where it promotes phosphorylation. Visible variables nodes Cln3 and V are connected to each of the submodules through either input or output ports. ClbM and V are present in every submodule because their values are used to calculate when cell division occurs.

7.2.6. Simulation results. To validate the hierarchical model, we compared its simulation results with results from the original model. Figure 21 displays the two model simulations.

We exported the hierarchical model into SBML format with the comp package using JCMC. Since the original
model is a non-hierarchical model, we decided to use the software tool COPASI to flatten and simulate the hierarchical model. COPASI was able to flatten the model by following the module interaction details stored with SBML comp. Parameter values and initial concentrations were set as described by Barik and colleagues. We compared the flattened hierarchical model to the deterministic version of the original model, in which the cell divides symmetrically. The simulation results from the flattened hierarchical model in Figure 21(b) match the results from the original model in Figure 21(a). This was verified by confirming that the time series simulation data from both models were identical.

8. Conclusions and future work

When building a mathematical model, templates can provide a compact and expressive way to re-use model components. A template is characterized as a generic module definition. Templates do not hold information related to one specific species. Instead, they contain general molecular mechanisms that are common in the regulatory network. A good example is the transcription and translation coupling module from the case study, shown in Figure 16. Here, species X is linked to an output port and the rate constants for the reactions are all linked to input ports. The ports let external entities utilize the interior transcription and translation reaction structure of the module. A template can be instantiated multiple times to describe the molecular behavior of different species without needing to be modified. The desired species and rate constants can be connected to the appropriate ports. This is done with ClbM, ClbS, and Cln3 in the case study. Alternatively, we could have created individual module definitions for ClbM, ClbS, and Cln3. In this case, each module definition would have contained the same reaction structure and we would have needed to write 12 repetitive reactions. Using a template allowed us to avoid such inefficiencies.

The ability to test a model is important, and frequent testing should occur during the model-building process. In software engineering, development testing ensures components are correct as they are developed. Each component is tested individually before it is added to the system. This allows errors to be discovered, and hopefully fixed, early in development. A similar approach should be taken when building a mathematical model. As modules are created, their inner reaction structures can be verified using simulation tools such as COPASI. As modules are connected together, their connections can be verified by checking the resulting equations. These tests should be done throughout the model-building process. Waiting until the end to test can make it extremely difficult to find the cause of an error.

As molecular network models continue to grow in size and complexity, traditional modeling practices are becoming obsolete. Building complex models in a controlled, organized manner is important. Without proper organization, complex models become difficult to understand. In order to construct and comprehend such models, we must evolve our modeling approach. In this paper, we have reviewed improved modeling approaches related to hierarchical model composition, along with software standards that support this modeling approach. We proposed enhancements to the way modules interact in model aggregation. We introduced new port and node constructs that enable multiple connection options for modules. We also described how different connections can impact a model.

Our tool that supports hierarchical modeling is named the JigCell Model Connector (JCMC). We described the JCMC interface and explained how the components of a hierarchical model are represented. We detailed how input, output, and equivalence ports form the module interface. We also explained how visible variable and equivalence nodes are utilized in JCMC. Finally, we demonstrated the benefits of hierarchical modeling with JCMC by reconstructing a complex biological model in a modular fashion.

A possible enhancement to the JCMC would be to incorporate multistate modeling. Currently, the JCMC only builds models with single-state species. Multistate modeling can drastically reduce the size of models containing species that have multiple states. Adding this enhancement to JCMC would give modelers the opportunity to create complex multistate models by connecting smaller submodules together.

Another possible enhancement to the JCMC would be to let any model component link to a port. Currently, JCMC only links species and module quantities to ports. SBML comp allows any model component to link to a port. Adding this feature to JCMC would allow for the import and export of reactions, events, and other model components between modules.

Disclaimer

Alida Palmisano contributed to this manuscript as a follow up to work done while employed at Virginia Tech. The opinions
expressed in this manuscript are the author’s own and do not reflect the view of the National Institutes of Health, the Department of Health and Human Services, or the United States government.

Availability
JCMC runs on all major operating systems. Installer, source code, user manual, tutorial, and examples can be found at the COPASI website: www.copasi.org/Projects.

Funding
This work was supported in part by NIH grants 2-R01-GM078989-05 and 2-R01-GM080219-09.

ORCID ID
Thomas C. Jones https://orcid.org/0000-0001-9264-1507
Clifford A. Shaffer https://orcid.org/0000-0003-0001-0295

References


**Author biographies**

**Thomas C. Jones, Jr** received his MS degree in Computer Science from Virginia Tech in 2017. He now works for Foundation Software.

**Stefan Hoops** has an education as a physicist and mathematician which makes him well-suited for any modeling research as the training focuses on the ability to describe the real world in mathematical terms. He combines this with more than 5 years of commercial software development experience and more than 17 years of scientific software development as the leading developer of the simulation software COPASI. Additionally, his experience with Life science at Virginia Tech’s Biocomplexity Institute (formerly VBI) provides him with great knowledge of the problems biologists, chemists, and medical doctors face. He is actively involved in Systems Biology standards efforts.

**Layne T. Watson** received the B.A. degree (magna cum laude) in psychology and mathematics from the University of Evansville, Indiana, in 1969, and the Ph.D. degree in mathematics from the University of Michigan, Ann Arbor, in 1974. He has worked for USNAD Crane, Sandia National Laboratories, and General Motors Research Laboratories and served on the faculties of the University of Michigan, Michigan State University, and University of Notre Dame. He is currently a professor of computer science, mathematics, and aerospace and ocean engineering at Virginia Polytechnic Institute and State University. He serves as senior editor of Applied Mathematics and Computation, and associate editor of Computational Optimization and Applications, Evolutionary Optimization, Engineering Computations, and the International Journal of High Performance Computing Applications. He is a fellow of the National Institute of Aerospace and the International Society of Intelligent Biological Medicine. He has published well over 300 refereed journal articles and 200 refereed conference papers. His research interests include fluid dynamics, solid mechanics, numerical analysis, optimization, parallel computation, mathematical software, image processing, and bioinformatics.

**Alida Palmisano** received her Ph.D. in Computer Science from the University of Trento, Italy in 2010. From 2011 to 2014, Dr. Palmisano served as a Post-Doctoral Fellow in the Departments of Computer Science and Biological Sciences at Virginia Tech. She then joined the Biometric Research Program, Division of Cancer Treatment and Diagnosis, at the National Cancer Institute (NIH), where she works on informatics systems for clinical trial management, and the development of software tools to visualize genomic sequencing data.

**John J. Tyson** is a University Distinguished Professor of Biological Sciences at Virginia Tech. He received his PhD in chemical physics from the University of Chicago in 1973 and has been specializing in theoretical cell biology since that time. His current interests revolve around the gene-protein interaction networks that regulate features of cell physiology such as cell division, circadian rhythms, intracellular signaling networks, and programmed cell death.

**Clifford A. Shaffer** received the PhD degree from the University of Maryland in 1986. He is a professor in the Department of Computer Science at Virginia Tech. His current research interests include computational biology, digital education, algorithm design and analysis, and data structures. He is a senior member of the IEEE and a Distinguished Educator in the ACM.