JigCell Model Connector: Building Large Molecular Network Models from Components

Thomas Carroll Jones Jr.

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters
in
Computer Science and Applications

Clifford A. Shaffer, Chair
John J. Tyson, Co-Chair
Layne T. Watson
Stefan Hoops

April 28, 2017
Blacksburg, Virginia

Keywords: Computational Systems Biology, Hierarchical Model Composition, SBML, Modeling Tool, Software, JigCell

JigCell Model Connector: Building Large Molecular Network Models from Components

Thomas Carroll Jones Jr.

(Abstract)

The ever-growing size and complexity of molecular network models makes them 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 together 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 different types of ports. Not all modules connect together the same way, therefore multiple connection options need to exist.

JigCell Model Connector: Building Large Molecular Network Models from Components

Thomas Carroll Jones Jr.

(General Audience Abstract)

Genes and proteins interact to control the functions of a living cell. In order to better understand these interactions, mathematical models can be created. A model is a representation of a cellular function that can be simulated on a computer. Results from the simulations can be used to gather insight and drive the direction of new laboratory experiments. As new discoveries are made, mathematical models continue to grow in size and complexity. 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 together 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 different types of ports. Not all modules connect together the same way, therefore multiple connection options need to exist.

Acknowledgments

I would first like to thank Dr. Clifford A. Shaffer and Dr. John J. Tyson for welcoming me into their research group. Both Dr. Shaffer and Dr. Tyson provided endless advice and support that guided me in my research. Their patience and encouragement allowed me to overcome many obstacles. I would like to thank the members of my committee, Dr. Layne T. Watson and Dr. Stefan Hoops, for their lively discussions that helped propel this research forward. I am grateful that each member of my committee, no matter how busy, always made time to meet, answer questions, and be involved. Their time was invaluable to me and the progress of this research.

Beside my committee, I would like to thank Dr. Lenwood S. Heath for introducing me to research at the crossroads of Computer Science and Biology. When I entered his office as an undergraduate student with interest but no experience in biology or research, Dr. Heath responded with "Well, let's get started." That kindness is something that has not been forgotten. I would like to thank Dr. Yang Cao for opening my eyes and helping me realize that attending Graduate School was a possibility.

I would like to give a big thanks to all the members of Dr. Tyson's Lab. Dr. Kathy Chen, Dr. Alida Palmisano, Dr. Kartik Subramanian, Dr. Pavel Kraikivski, and Dr. Dorjsuren Battogtokh each made the lab a great place. Thank you for welcoming me into such a special group.

Finally, I would like to thank my family for their everlasting encouragement. Without their support, I am sure none of this would have been possible. They have always believed in me, and for this I will be forever grateful.

Contents

List of Figures

List of Tables

Chapter 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 the networks, mathematical models of the biochemical reactions can be constructed [22, 21, 17]. Modelers have the difficult task of specifying reaction details between species connected in these complex regulatory networks.

Modeling an accurate system is not a one-step task. Modeling is an iterative process that involves frequent changes [5]. Once a model is drafted, the equations can be analyzed and simulated to describe the molecular behavior [21]. These computational results can then be compared to existing experimental data. 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 navigate the direction of future experiments [23]. If further experiments uncover inconsistencies, the model can again be modified.

As molecular biologists discover more information about how gene and protein interactions affect cell physiology, the size and complexity of the mathematical models continue to grow. Attempting to construct these models is becoming more difficult. In order to better comprehend increasingly complex models, new modeling approaches need to be explored.

Hierarchical model composition is a modeling technique that allows models to be submodels inside of another model. 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, it can be more easily understood.

We designed the JigCell Model Connector, a software tool to support hierarchical model composition. In our tool, 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. Modelers are able to regulate external access to internal components of a module by utilizing ports. We implement different port types that allow modules to connect in different ways. Once a model is created in the JigCell Model Connector, it can

be exported into a standard file format. The model can then be simulated and analyzed by other tools. Our goal is to develop large models in a modular way, making the result easier to comprehend.

In Chapter 2, hierarchical model composition, standards, and tools related to hierarchical modeling are reviewed. Chapter 3 presents the JigCell Model Connector environment and its components. In Chapter 4, the different types of ports and their impact on a model are discussed. Chapter 5 gives an example of hierarchical modeling with a full biological model. In Chapter 6, the best practices for constructing hierarchical models, conclusions, and future work are discussed.

Chapter 2

Background

In this chapter, we review several topics that serve as building blocks for this thesis. We discuss the style of modeling used and a standard format that enables the sharing of models. We also review previous modeling tools.

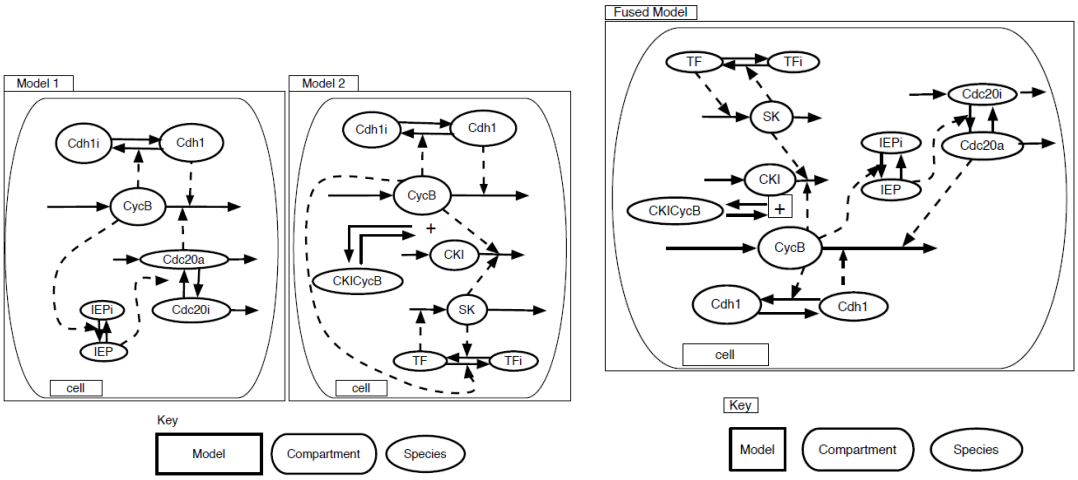
2.1 Hierarchical Model Composition

As the scope of research into molecular networks expands, the representative computational models continue to grow in size and complexity. The increasing complexity makes models difficult to construct and understand using traditional modeling practices. Below we review improved modeling approaches.

Randhawa et al. [12, 14, 15] introduce the method of model fusion. Model fusion is a process where two or more complete models are combined, making one large model as shown in Figure 2.1. The models are modified in such a way that the process is irreversible. This means that the original smaller models (submodels) cannot be recognized and recovered once fused together. The goal of fusion is to create a single unified model containing all of the information from the submodels, without repetition. Model fusion is accomplished in two steps, name resolution and automatic merging. Name resolution involves removing repetition and identifying equivalences across the various components of the submodels. Automatic merging places the remaining submodel components together into a single model.

Randhawa et al. [12, 14, 15] 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.

Large models are simply collections of submodels, and can be organized in a hierarchical fash-



(a) Sample models (b) Fused model

Figure 2.1: Model fusion [12]

ion. Unlike fusion, model composition is a reversible process. If the interaction descriptions are removed, then the original submodels can be recovered.

Randhawa et al. [13, 15] characterize model aggregation as a restricted form of model composition. Here, they define a module as a collection of model components. A module also includes a specification for predetermined ports. A port is a link to an internal model component, such as a species or parameter. Therefore, a module is a submodel with ports. The ports of a module form an interface, which only allows access to specific components within the module. The process of grouping model components and assigning ports is referred to as modularization, as shown in Figure 2.2. In model composition, any component of a submodel could be referenced in a larger composed model. In model aggregation, only a component linked to a port can be referenced in a larger composed model. Modules are then connected together by their interface ports. With model aggregation, modelers can build larger models in a controlled manner.

Randhawa et al. [12, 13, 15] also presented the concept of model flattening. Model flattening converts composed or aggregated models to their "flattened versions", as shown in Figure 2.3a and Figure 2.3b. 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.

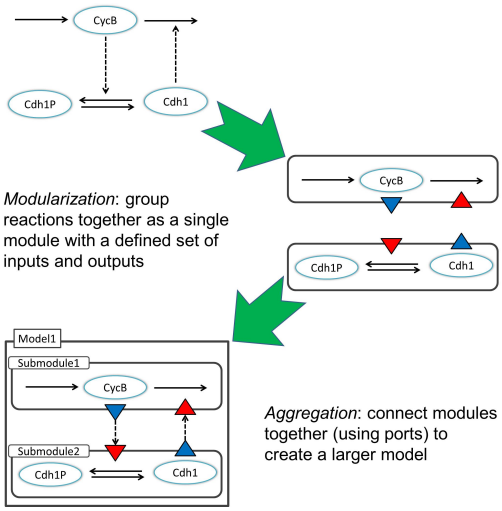


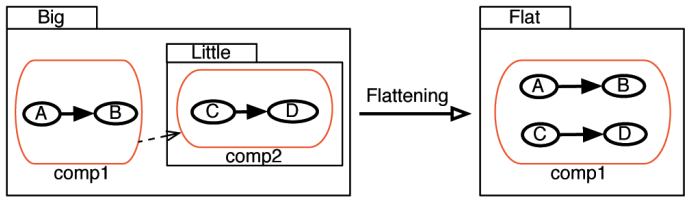
Figure 2.2: Model aggregation

2.2 The SBML Standard

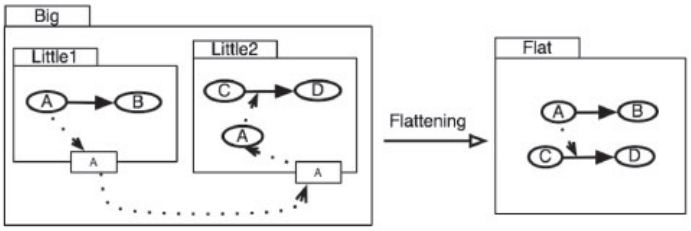
The Systems Biology Markup Language (SBML) is a format that represents systems biology models electronically. Such a standard allows for the sharing and collaboration of models. SBML Level 1 [8] was introduced in 2001. A model defined in SBML can consist of many components, such as compartments, species, reactions, and parameters. The interactions between components in the model are also defined in SBML. It is important to note that SBML is not designed to be 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.

As we just explained, a model defined in SBML can consist of many components. Extra information about a component can be stored in an annotation. Annotations can be thought of as comments in an SBML file. These comments can be left for any type of SBML component. Additional details found in an annotation pertain to the component which it is attached, such as a reaction, species, or parameter. The information in an annotation is software-generated and not meant to be in an easily human-readable format. Typically, annotations are used by software developers to include application-specific data.

The latest version, SBML Level 3 Version 1 Core [9], was released in 2010. In Level 3 Version 1, SBML incorporates new sets of related features in a modular form. These feature extensions are referred to as packages. Multiple packages can be used within a single SBML model. All packages used within a model must be declared at the beginning of the SBML document. Each package declaration includes a `required` attribute. The attribute is listed as `required=‘‘true’’` or `required=‘‘false’’`. If a package alters the mathematical meaning of the model, then this attribute must be set to `true`. A `required` attribute of `true` alerts software applications reading the model that the corresponding package must be taken



(a) Composed model [14]



(b) Aggregated model [13]

Figure 2.3: Model flattening

into consideration. Otherwise, a `required` attribute of `false` indicates that the package information can be ignored and the mathematical meaning of the model will not change. 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 [19] 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. A model definition is a complete, self-contained model. Model definitions instantiated as submodels are located 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, 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 [6] was released in 2013. The package allows components of a model to be represented graphically. Pieces of a model can be visually organized in an attempt to provide more clarity. The model structure is extended

to include a list of layouts. A layout can store the information for graphics representing some or all components of the SBML model. These graphics are referred to as glyphs in *layout*. A compartment, species, and reaction can be represented by a CompartmentGlyph, SpeciesGlyph, and ReactionGlyph, respectfully. There is also a GeneralGlyph that can 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 those details. These extended features of *layout* enable model visualization in SBML.

2.3 Related Tools

There are numerous software tools available for the modeling and simulation of molecular networks. For example, Antimony [18] 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*.

COPASI [7] 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. It provides support for importing/exporting standard SBML (Level 3). However, COPASI lacks features to support hierarchical modeling and SBML *comp*.

Previous iterations of JigCell [1, 2, 13, 14, 16, 24, 25] have included a Model Builder, Aggregation Connector, Run Manager, Comparator, and Parameter Estimation Toolkit. The JigCell suite of tools can be used to model, simulate, and analyze biochemical networks. 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) [11] is a tool that supports the modeling of biochemical networks. The graphical user interface builds on the tabular spreadsheet format used by [24, 25]. JC-MSMB reduces the complexity of model creation by introducing a new syntax to describe multistate species. The 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*.

iBioSim [10] is a tool for the modeling, analysis, and design of genetic circuits. In syn-

thetic biology, genetic circuits can be used to design and construct networks to implement a particular cellular function [4]. 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. It provides support for importing/exporting hierarchical models using SBML *comp*. Both JigCell and iBioSim offer support for SBML and hierarchical modeling, but lack the features of different port types.

Chapter 3

JigCell Model Connector

This thesis is primarily about the JigCell Model Connector (JCMC). In order to better understand the purpose and key features of 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 in the following chapters.

3.1 Interface

The JCMC interface consists of three panels. These panels are shown in Figure 3.1 and described below.

3.1.1 TreeView

The left panel is the TreeView. It displays the hierarchical relationships between components of the model. An example is shown in Figure 3.2. A module can be selected 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 is the DrawingBoard. It displays the graphical view of a module, its submodules, and any connections among them. An example is shown in Figure 3.3. The currently loaded module is called the container module. In Figure 3.3 the container module is named Model. Submodules can be moved inside of the container module. In Figure 3.3, Cdh1

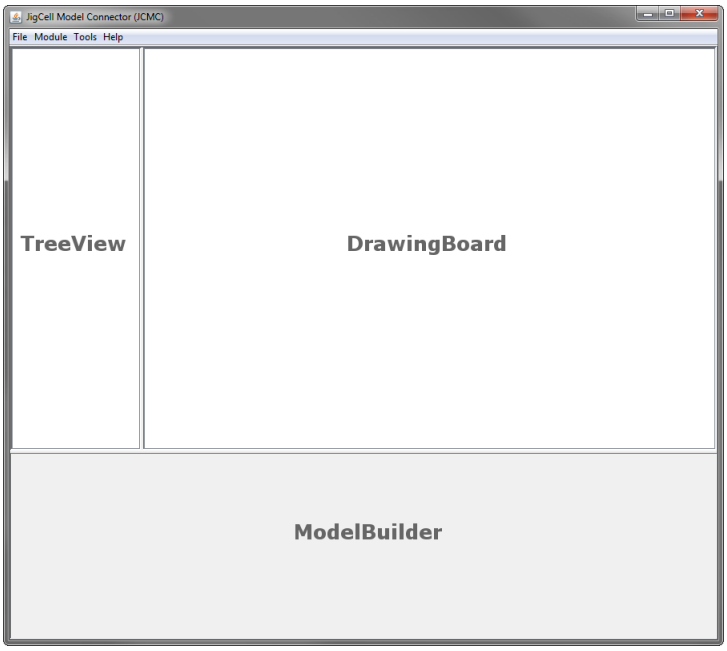


Figure 3.1: Three panels and the menubar

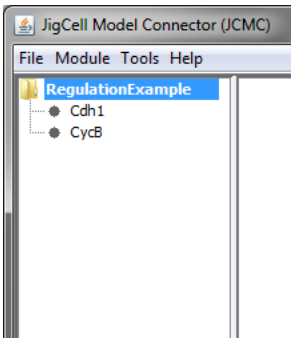


Figure 3.2: TreeView

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. We will show examples of these components in later sections.

3.1.3 ModelBuilder

The bottom panel is the ModelBuilder, shown in Figure 3.4. It is similar to the JigCell Multistate Model Builder (JC-MSMB) [11], which was previously implemented by our group. The ModelBuilder in JCMC is different from JC-MSMB because the ModelBuilder does not support multistate species. It is a tabular spreadsheet interface where the details of a module

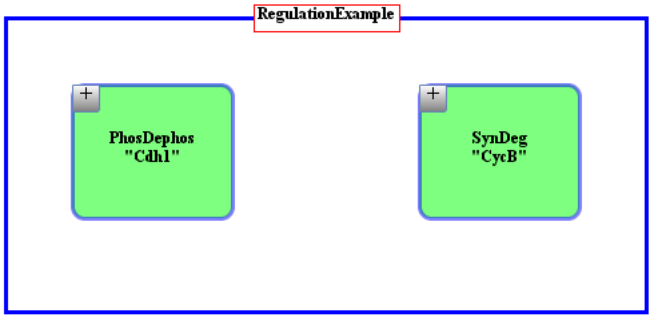


Figure 3.3: DrawingBoard

are displayed. Attributes such as reactions, species, parameters, and events can be modified.

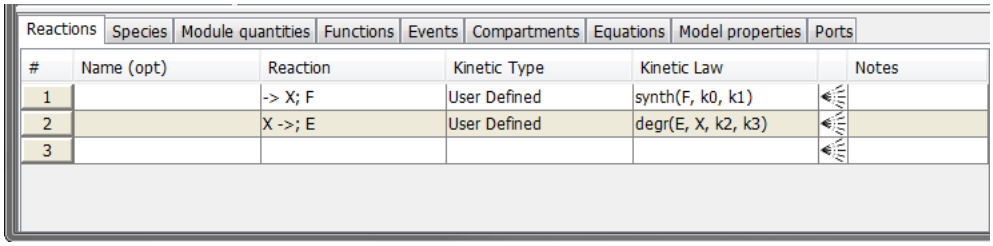


Figure 3.4: ModelBuilder

3.2 Components

3.2.1 Container Module

The container module is the current loaded module. The TreeView panel shows the container module's name in bold font. The ModelBuilder panel displays the module definition of 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. There can only be one container module loaded at a time.

3.2.2 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 sub-

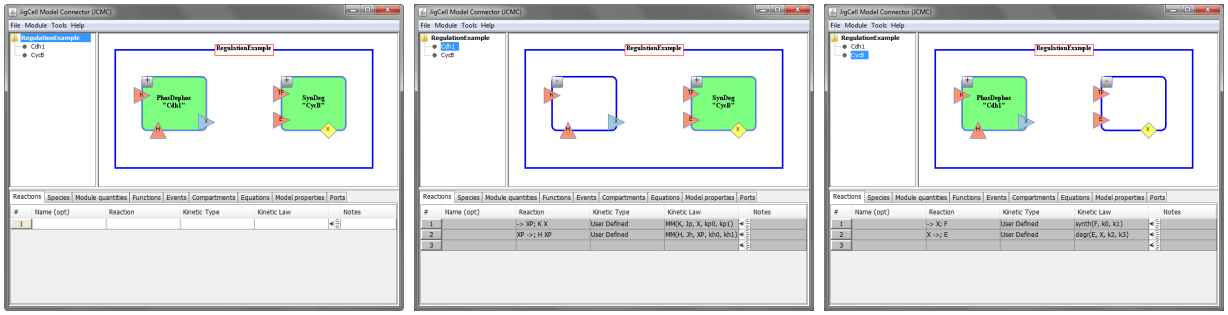
module can be moved and resized within the bounds of its container module. A submodule's information is listed as:

<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 3.3, 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 see 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. 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 pressed, the information for that submodule will be displayed in the ModelBuilder. An example is shown in Figure 3.5. After the button for Cdh1 is pressed, the template information for Cdh1 will be displayed in the ModelBuilder. This is shown in Figure 3.5b. Notice the tables are grayed-out. This is because the information is a preview only, and cannot be modified. To modify the information, load the submodule as the container module. Figure 3.5c shows when the button for CycB has been pressed.



(a) (b) (c)

Figure 3.5: Submodule information preview

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.

Chapter 4

Technical Contributions

As discussed in Chapter 2, Randhawa et al. [13, 15] 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. Not all modules connect together in the same way, therefore multiple connection options need to exist. In this chapter, we present different port and node constructs to enable multiple connection options. We also discuss how connections can impact a model.

4.1 Ports

Ports allow internal components of a module to be referenced outside of that module. A port can be linked to either a species or module quantity. Once created, the ports combine to form an interface. External access to a module's internal components is regulated by the interface. Modules can be connected together by their interfaces to build larger models.

Previous software tools [10, 13, 14, 16] support ports and ports are included in the SBML *comp* package [19]. However, these treat all ports the same. Internal components linked to ports do not necessarily serve the same purpose for every module. Currently, a port gives no information as to how an internal component is used. Based on the port, modelers have no way to discern a component's purpose within a module. There exists a need for different port types, where the type of port is dependent upon how the linked internal component is used within a module. Using port types, modelers are able to convey their intent for a component. The port types are described below.

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 where 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 explained in Section 4.4.1. 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 where 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 explained in Section 4.4.1. 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 where 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 explained in Section 4.4.2. Equivalence ports are represented as diamonds on the edge of modules.

In the DrawingBoard panel, the ports are displayed on the module boundaries. In the ModelBuilder panel, ports are listed under the Ports tab (shown in Figure 4.1). 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 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, load the submodule as the container module then proceed with the modification.

4.2 Nodes

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

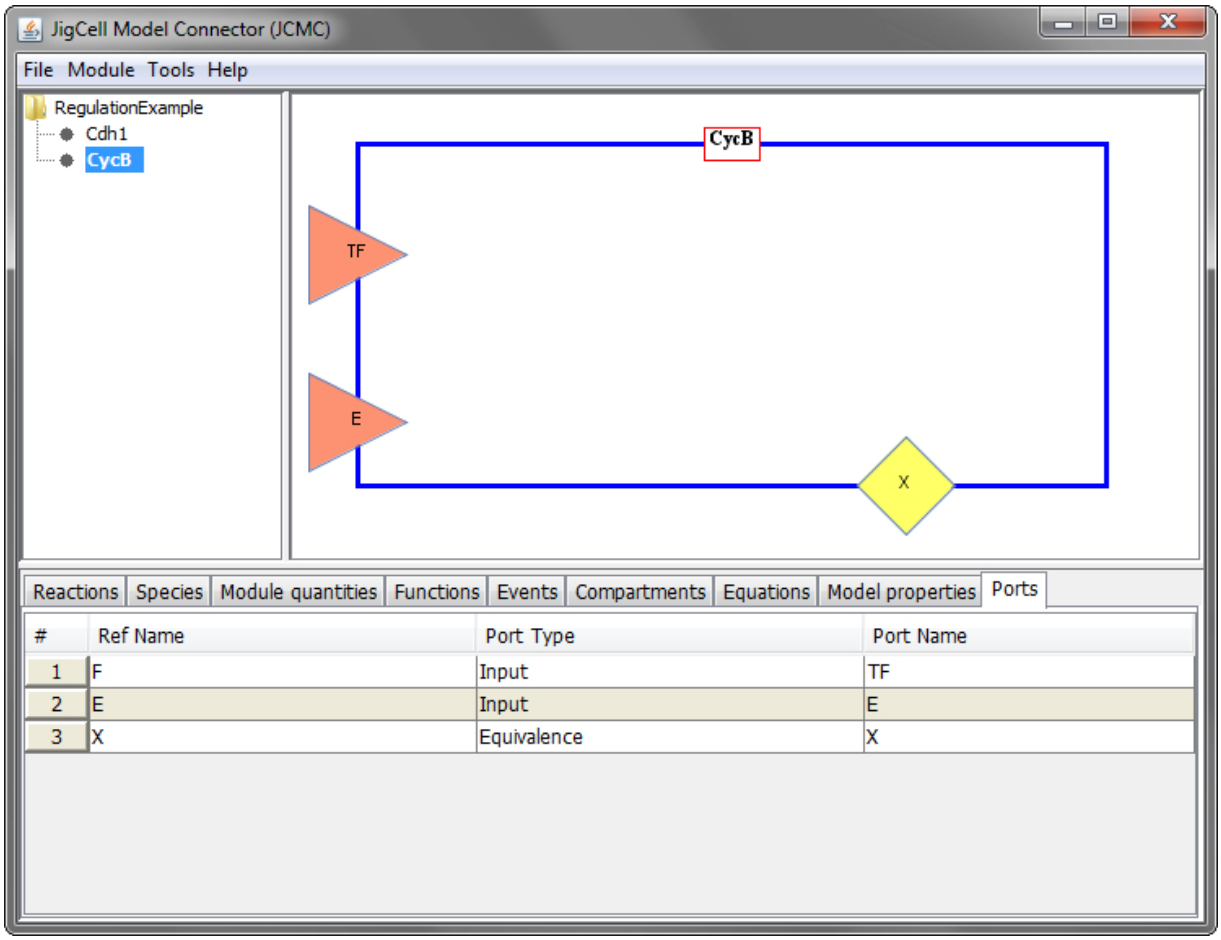


Figure 4.1: Ports tab

4.2.1 Visible Variable Node

A visible variable node is automatically created when a connection is made between input or output ports of two modules. Figure 4.2 shows two submodules after a connection is made, a visible variable is created, and the new variable is 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. Select a variable, click Add, and 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.

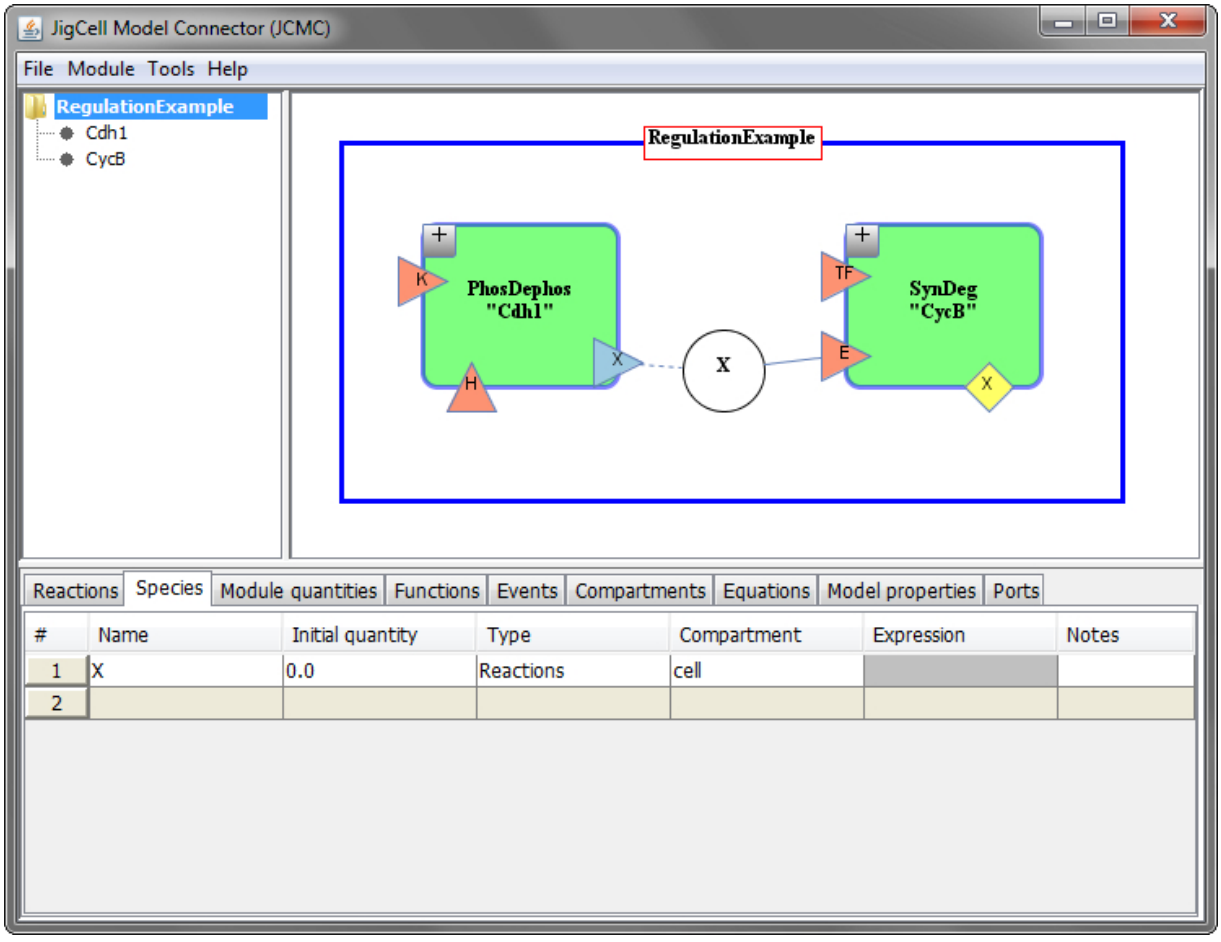


Figure 4.2: Visible Variable created with a connection

4.2.2 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.

4.3 Connections

A set of connections can 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 4.1, 4.2, and 4.3.

Table 4.1: Rules for Submodule to Submodule connections

		Target Submodule Port		
		Input	Output	Equivalence
Source Submodule Port	Input	Invalid	Invalid	Invalid
	Output	Valid	Invalid	Invalid
	Equivalence	Valid	Invalid	Valid

Table 4.2: Rules for Container Module to Submodule connections

		Target Submodule Port		
		Input	Output	Equivalence
Source Container Module Port	Input	Valid	Invalid	Invalid
	Output	Invalid	Invalid	Invalid
	Equivalence	Valid	Invalid	Valid

Table 4.3: Rules for Submodule to Container Module connections

		Target Container Module Port		
		Input	Output	Equivalence
Source Submodule Port	Input	Invalid	Invalid	Invalid
	Output	Invalid	Valid	Invalid
	Equivalence	Invalid	Invalid	Valid

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 created.

4.4 Port, Node, and Connection Effects on a Model

In this section we will 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 $\frac{d}{dt}[name]$.

4.4.1 Input and Output Ports

We will begin with a simple synthesis and degradation module.

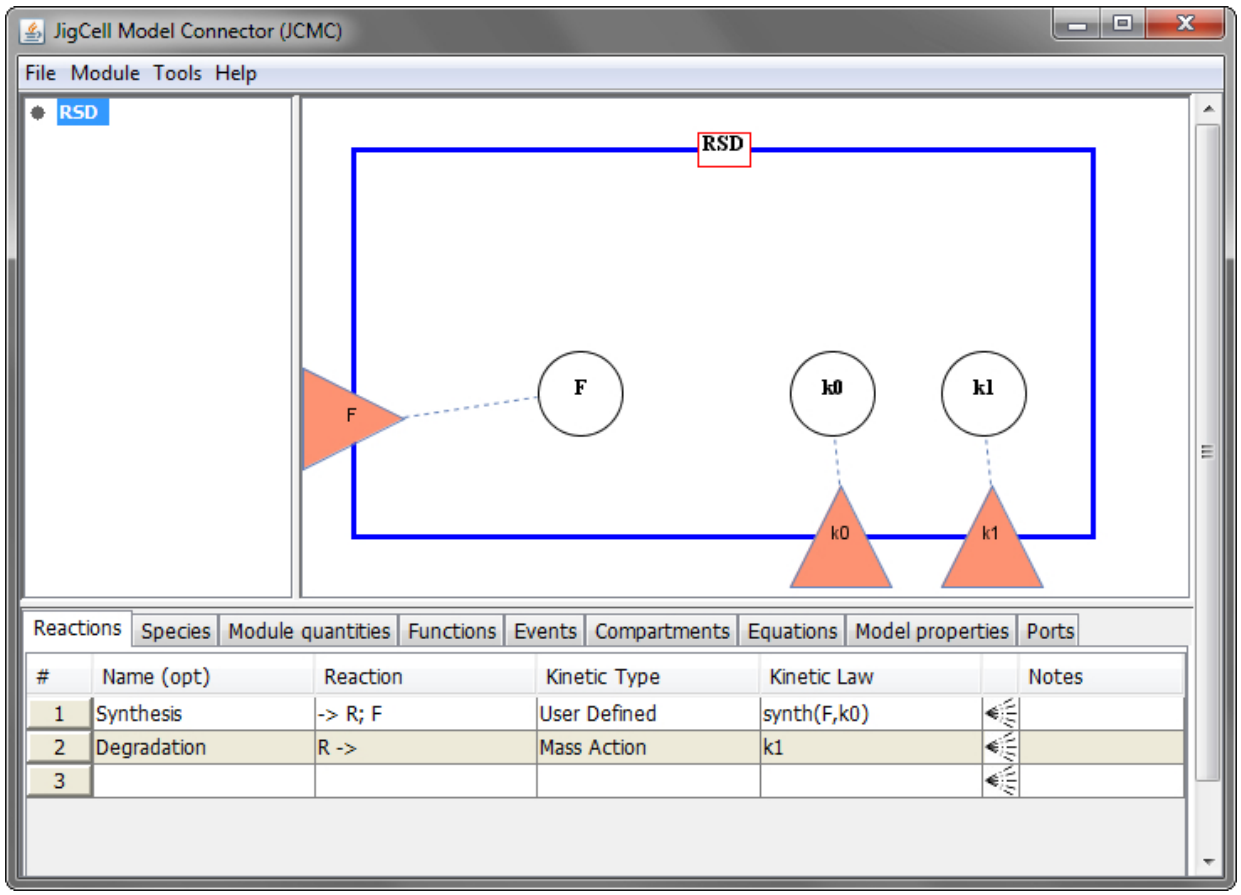


Figure 4.3: Synthesis and degradation module RSD

Figure 4.3 shows the synthesis and degradation module RSD in JCMC. The reactions are displayed in the ModelBuilder panel at the bottom. The rate of synthesis is determined by rate constant $k0$ and transcription factor F. The rate of degradation is determined by mass action kinetics with rate constant $k1$. The dynamics of species R in module RSD are shown

in

$$\frac{d}{dt}[R] = (k1 \times [F]) - (k0 \times [R]). \tag{4.1}$$

Figure 4.3 also shows that module quantities $k0$ and $k1$ are connected to input ports. Because $k0$ and $k1$ are connected to input ports, they can receive values from external connections. Note that $k0$ and $k1$ are not modified within the module RSD. They are only used for the computations of other variables.

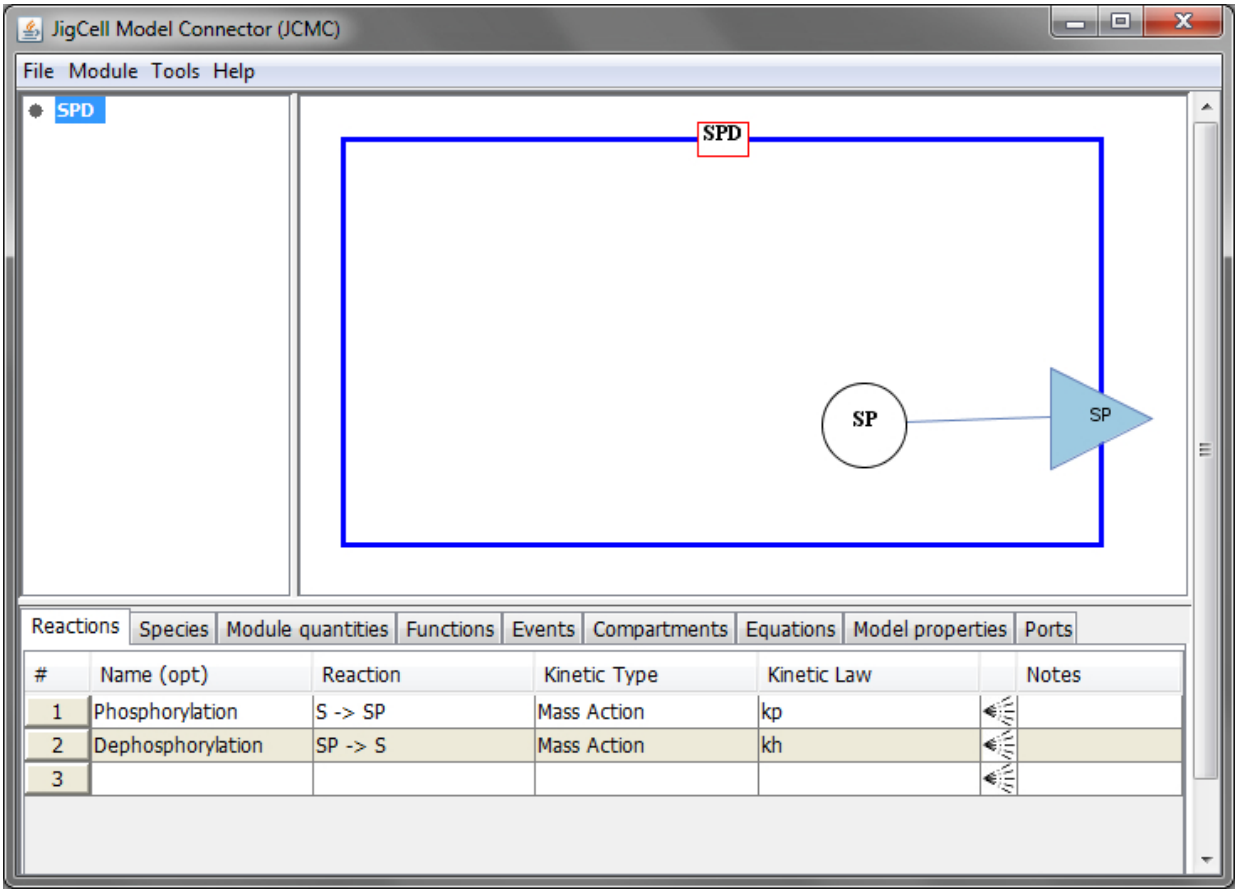


Figure 4.4: Phosphorylation and dephosphorylation module SPD

Figure 4.4 shows the phosphorylation and dephosphorylation module SPD in JCMC. 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 kh. The species dynamics of module SPD are described by

$$\frac{d}{dt}[S] = -(kp \times [S]) + (kh \times [SP]), \tag{4.2}$$

$$\frac{d}{dt}[SP] = (kp \times [S]) - (kh \times [SP]). \tag{4.3}$$

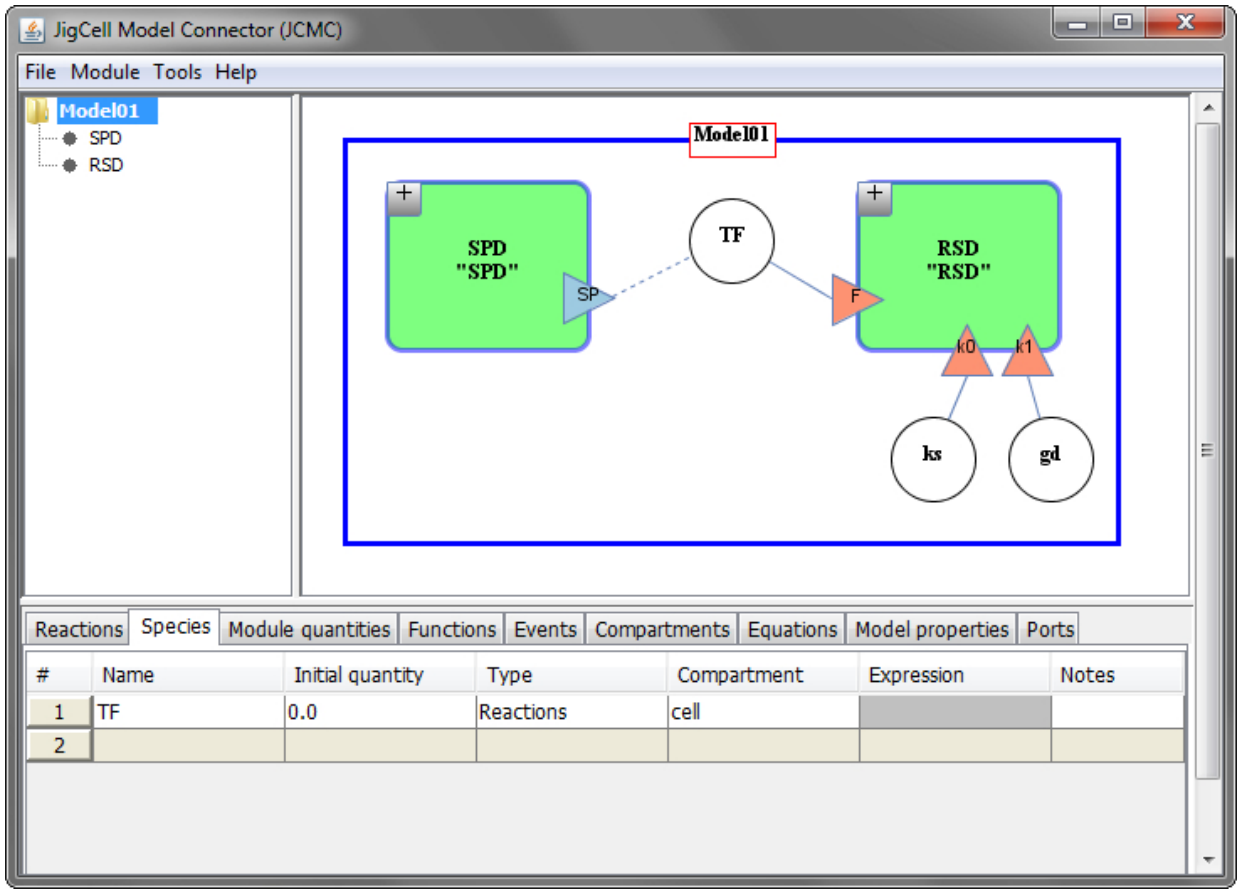


Figure 4.5: Output port example

Figure 4.5 displays Model01, where SPD and RSD are submodules. Species TF, module quantity ks, and module quantity gd are displayed as visible variable nodes. SPD has an output port linked to species SP and RSD has input ports linked to species F, module quantity $k0$, and module quantity $k1$. Node ks is connected to the $k0$ port on submodule RSD. This means $k0$ will receive the value of ks. To accomplish this, $k0$ will be replaced by ks in submodule RSD. The same will happen with $k1$ and gd. The replacement is not immediate but will occur when the entire model is flattened. There is one connection from module SPD's SP port to node TF and another from node TF to module RSD's F port. SPD's SP port will send its internal value to node TF. Node TF will then send the value to RSD's F port. Finally, the internal species F in RSD will receive the value. When the model is flattened, TF will replace SP in SPD and F in RSD. The species dynamics after

flattening are

$$\frac{d}{dt}[S] = -(kp \times [S]) + (kh \times [TF]), \tag{4.4}$$

$$\frac{d}{dt}[TF] = (kp \times [S]) - (kh \times [TF]), \tag{4.5}$$

$$\frac{d}{dt}[R] = (ks \times [TF]) - (gd \times [R]). \tag{4.6}$$

(4.3) has been replaced by (4.5) and TF has replaced SP in (4.2) to form updated (4.4). Similarly, TF replaced F in (4.1) to form updated (4.6).

4.4.2 Equivalence Port

Figure 4.6 displays the ModelBuilder panel for three different modules.

Reactions | Species | Module quantities | Functions | Events | Compartments | Equations | Model properties | Ports

#	Name (opt)	Reaction	Kinetic Type	Kinetic Law	Notes
1	Synthesis	-> X	User Defined	synth(ks)	
2	Degradation	X ->	Mass Action	gd	
3					

(a) XSD

Reactions | Species | Module quantities | Functions | Events | Compartments | Equations | Model properties | Ports

#	Name (opt)	Reaction	Kinetic Type	Kinetic Law	Notes
1	Phosphorylation	Y -> YP	Mass Action	kp	
2	Dephosphorylation	YP -> Y	Mass Action	kh	
3					

(b) YPD

Reactions | Species | Module quantities | Functions | Events | Compartments | Equations | Model properties | Ports

#	Name (opt)	Reaction	Kinetic Type	Kinetic Law	Notes
1	Association	Z + W -> Comp	Mass Action	ka	
2	Dissociation	Comp -> Z + W	Mass Action	kd	
3					

(c) CompAD

Figure 4.6: Module reaction information

Figure 4.6a shows the reaction details for the synthesis and degradation of species X in module XSD. 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 XSD are shown in

$$\frac{d}{dt}[X] = ks - (gd \times [X]). \tag{4.7}$$

Figure 4.6b shows the reaction details for the phosphorylation and dephosphorylation of species Y in module YPD. 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 kh. The species dynamics of module YPD are

$$\frac{d}{dt}[Y] = -(kp \times [Y]) + (kh \times [YP]), \tag{4.8}$$

$$\frac{d}{dt}[YP] = (kp \times [Y]) - (kh \times [YP]). \tag{4.9}$$

Figure 4.6c shows the reaction details for the association and dissociation of species Comp in module CompAD. 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 CompAD are

$$\frac{d}{dt}[Z] = -(ka \times [W]) + (kd \times [Comp]), \tag{4.10}$$

$$\frac{d}{dt}[W] = -(ka \times [Z]) + (kd \times [Comp]), \tag{4.11}$$

$$\frac{d}{dt}[Comp] = (ka \times [Z] \times [W]) - (kd \times [Comp]). \tag{4.12}$$

Figure 4.7 displays Model02, where XSD, YPD, and CompAD are submodules. Species A is displayed as an equivalence node. XSD has an equivalence port linked to species X, YPD has an equivalence port linked to species Y, and CompAD 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, A will replace X in XSD, Y in YPD, and Z in CompAD. The species dynamics after flattening are

$$\frac{d}{dt}[YP] = (kp \times [A]) - (kh \times [YP]), \tag{4.13}$$

$$\frac{d}{dt}[W] = -(ka \times [A]) + (kd \times [Comp]), \tag{4.14}$$

$$\frac{d}{dt}[Comp] = (ka \times [A] \times [W]) - (kd \times [Comp]), \tag{4.15}$$

$$\frac{d}{dt}[A] = \overbrace{ks - (gd \times [A])}^{\text{from } X \text{ in XSD}} - \overbrace{(kp \times [A]) + (kh \times [YP])}^{\text{from } Y \text{ in YPD}} - \overbrace{(ka \times [W]) + (kd \times [Comp])}^{\text{from } Z \text{ in CompAD}}. \tag{4.16}$$

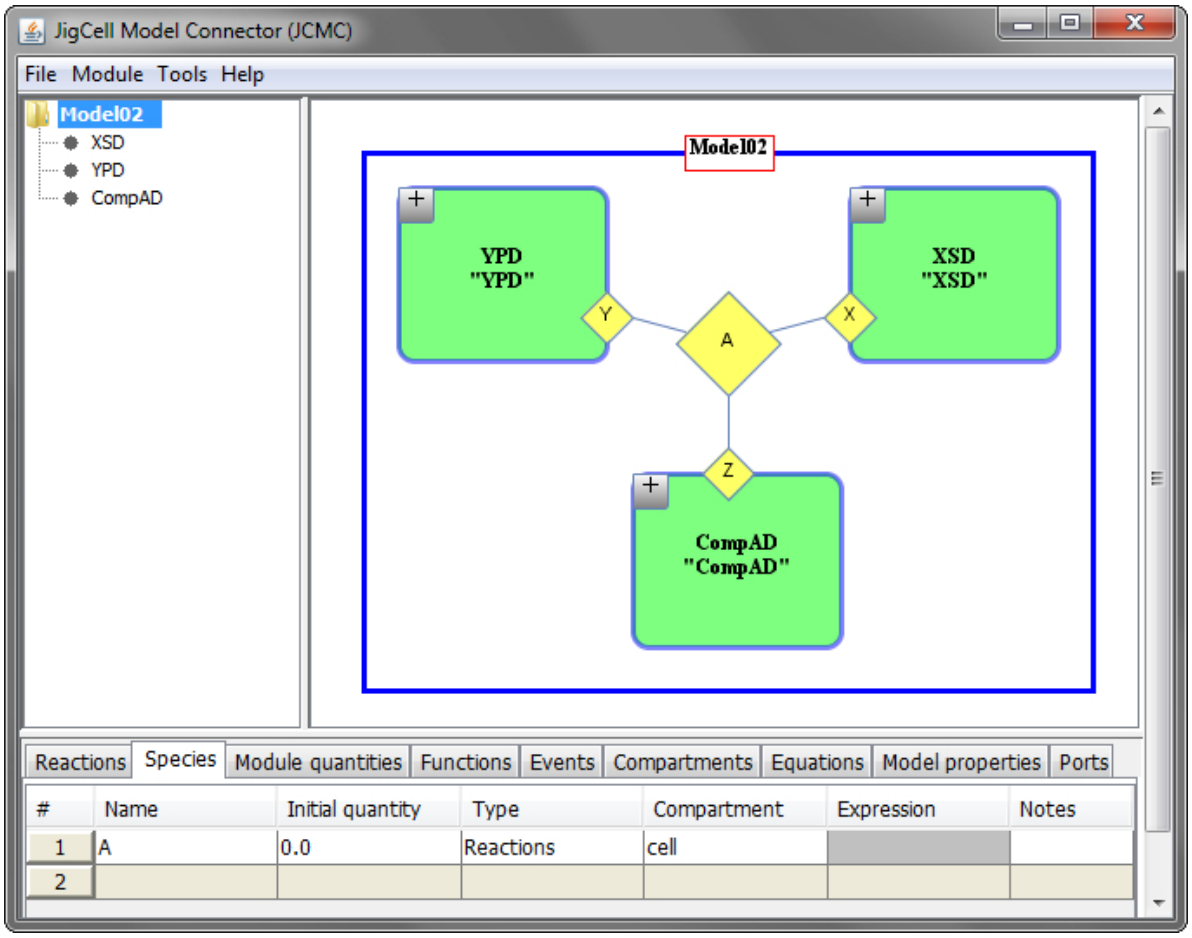


Figure 4.7: Equivalence port example

(4.13) is an updated version of (4.9), where A has replaced Y. Similarly, A has replaced Z in (4.11) and (4.12) to form updated (4.14) and (4.15). Notice that (4.16) is an aggregate of (4.7),(4.8), and (4.10). Since node A is connected to equivalence ports, values are both sent and received, therefore information from each connection is kept.

4.5 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.

4.5.1 Ports

Ports are included in the *comp* package. However, the different port types introduced in section 4.1 are not. In order to store this extra information in SBML, we decided to use annotations. Below is an example of two ports using the *comp* package.

```
<comp:listOfPorts>
  <comp:port comp:idRef="ksx" comp:id="ksx" comp:name="ksx">
    <annotation>
      <jcmc:portInfo
      xmlns:jcmc="http://www.copasi.org/Projects/JigCell_Model_Connector"
      jcmc:refName="ksx"
      jcmc:pType="Input"
      jcmc:vType="Module Quantity"/>
    </annotation>
  </comp:port>
  <comp:port comp:idRef="Metabolite_67" comp:id="X" comp:name="X">
    <annotation>
      <jcmc:portInfo
       xmlns:jcmc="http://www.copasi.org/Projects/JigCell_Model_Connector"
       jcmc:refName="X"
       jcmc:pType="Equivalence"
       jcmc:vType="Species"/>
    </annotation>
  </comp:port>
</comp:listOfPorts>
```

The annotation stores the port type, variable type, and variable name.

The *layout* package does not have a specific glyph to represent ports. It does have 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. Below is an example of the graphical information for two ports using the *layout* package.

```
<layout:listOfSubGlyphs>
  <layout:graphicalObject layout:id="ksx_PortGlyph">
    <annotation>
      <jcmc:portInfo
      xmlns:jcmc="http://www.copasi.org/Projects/JigCell_Model_Connector"
      jcmc:refName="ksx"
      jcmc:id="ksx"
      jcmc:pType="Input"
```

```
      jcmc:vType="Module Quantity"
      jcmc:parentMod="Cdh1TT"/>
    </annotation>
    <layout:boundingBox>
      <layout:position layout:x="0.7898753894081" layout:y="1"/>
      <layout:dimensions layout:width="0" layout:height="0"/>
    </layout:boundingBox>
  </layout:graphicalObject>
  <layout:graphicalObject layout:id="X_PortGlyph">
    <annotation>
      <jcmc:portInfo
      xmlns:jcmc="http://www.copasi.org/Projects/JigCell_Model_Connector"
      jcmc:refName="X"
      jcmc:id="X"
      jcmc:pType="Equivalence"
      jcmc:vType="Species"
      jcmc:parentMod="Cdh1TT"/>
    </annotation>
    <layout:boundingBox>
      <layout:position layout:x="0.291666666666667" layout:y="0"/>
      <layout:dimensions layout:width="0" layout:height="0"/>
    </layout:boundingBox>
  </layout:graphicalObject>
</layout:listOfSubGlyphs>
```

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 where the port is located.

4.5.2 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. Below is an example of a visible variable node $ClbS$ and an equivalence node $ClbM$.

```
<listOfSpecies>
  <species id="Metabolite_21" name="ClbS" compartment="cell"
  initialConcentration="0" substanceUnits="mole"
  hasOnlySubstanceUnits="false"
  boundaryCondition="false" constant="false">
```

```
    <annotation>
      <jcmc:speciesInfo
      xmlns:jcmc="http://www.copasi.org/Projects/JigCell_Model_Connector"
      jcmc:refName="ClbS"
      jcmc:type="VisibleVariable"/>
    </annotation>
  </species>
  <species id="Metabolite_23" name="ClbM" compartment="cell"
  initialConcentration="0" substanceUnits="mole"
  hasOnlySubstanceUnits="false"
  boundaryCondition="false" constant="false">
    <annotation>
      <jcmc:speciesInfo
       xmlns:jcmc="http://www.copasi.org/Projects/JigCell_Model_Connector"
       jcmc:refName="ClbM"
       jcmc:type="Equivalence"/>
    </annotation>
  </species>
</listOfSpecies>
```

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. Below is an example of the graphical information for visible variable node *ClbS* and equivalence node *ClbM* using the *layout* package.

```
<layout:listOfSubGlyphs>
  <layout:graphicalObject layout:id="ClbS_VisibleVariableNodeGlyph">
    <annotation>
      <jcmc:VisibleVariableNodeInfo
      xmlns:jcmc="http://www.copasi.org/Projects/JigCell_Model_Connector"
      jcmc:name="ClbS"
      jcmc:vType="Species"/>
    </annotation>
    <layout:boundingBox>
      <layout:position layout:x="475" layout:y="75"/>
      <layout:dimensions layout:width="0" layout:height="0"/>
    </layout:boundingBox>
  </layout:graphicalObject>
  <layout:graphicalObject layout:id="ClbM_EquivalenceNodeGlyph">
```

```
    <annotation>
      <jcmc:EquivalenceNodeInfo
      xmlns:jcmc="http://www.copasi.org/Projects/JigCell_Model_Connector"
      jcmc:name="ClbM"
      jcmc:vType="Species"/>
    </annotation>
    <layout:boundingBox>
      <layout:position layout:x="315" layout:y="175"/>
      <layout:dimensions layout:width="0" layout:height="0"/>
    </layout:boundingBox>
  </layout:graphicalObject>
</layout:listOfSubGlyphs>
```

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

Chapter 5

Case Study

In this chapter, we will demonstrate the features of JCMC by building a complex biological model. Barik et al. [3] 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.

5.1 Biological Model

Figure 5.1 shows a wiring diagram of the biological model from [3]. Species are represented by the various labeled shapes. Chemical reactions are represented by solid arrows, enzymatic activities are represented by dotted arrows, and multisite phosphorylation chains are represented by dashed arrows. Reversible binding reactions are represented by T-shaped arrows with balls on the cross bars. For clarification purposes, Figure 5.1 only displays 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 et al. [3] 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 concentration of each species within are evenly split between the two resulting daughter cells. 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.

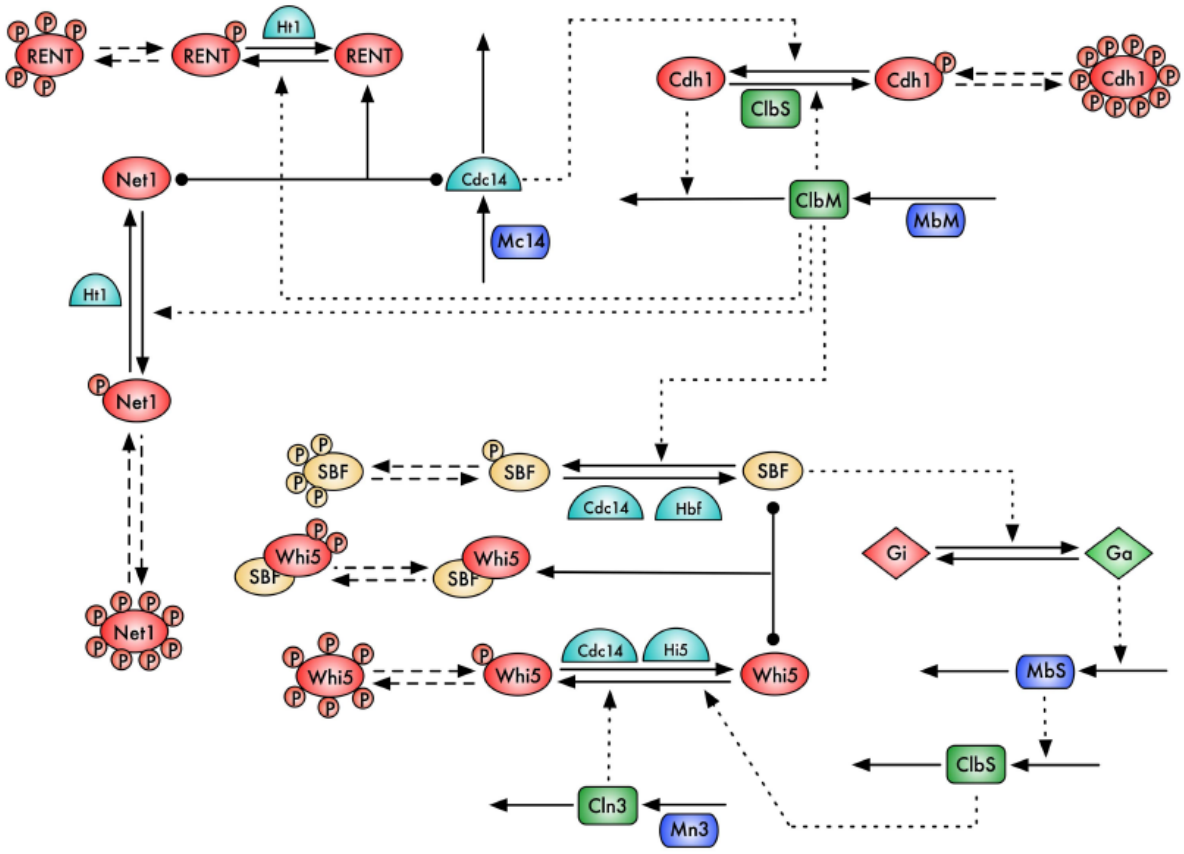


Figure 5.1: Model of cell-cycle control in budding yeast [3]

5.2 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 5.2) 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.

5.2.1 Transcription and Translation Coupling

From Figure 5.1 we can see that some of the regulatory functions in the model are similar. The mechanisms regulating *ClbM*, *ClbS*, and *Cln*3 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

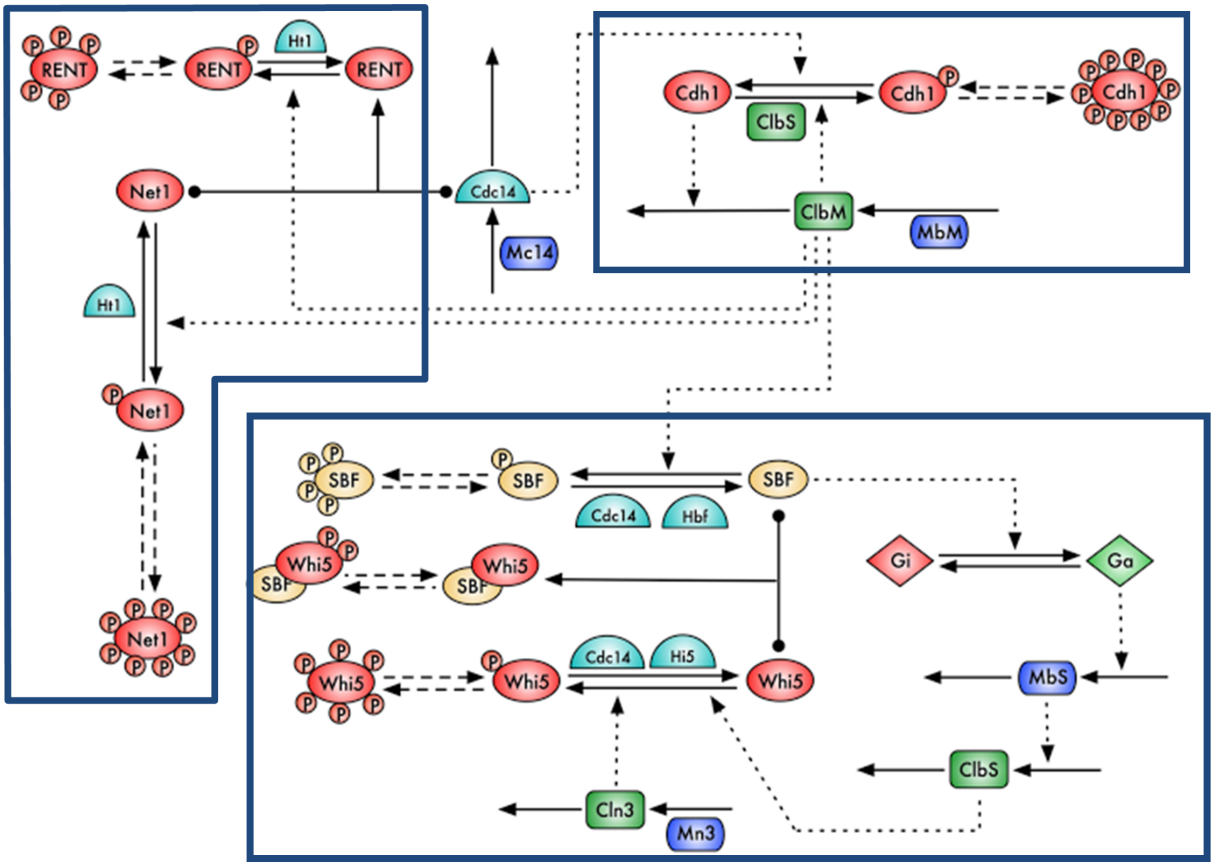


Figure 5.2: Modular model of cell-cycle control in budding yeast

build it as a reusable, generic module.

Figure 5.3 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 ksm, gdm, ksx, and gdx 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 $ClbM$ and determines if the cell needs to divide. If the cell needs to divide then the species numbers within the module are reassigned appropriately. An event

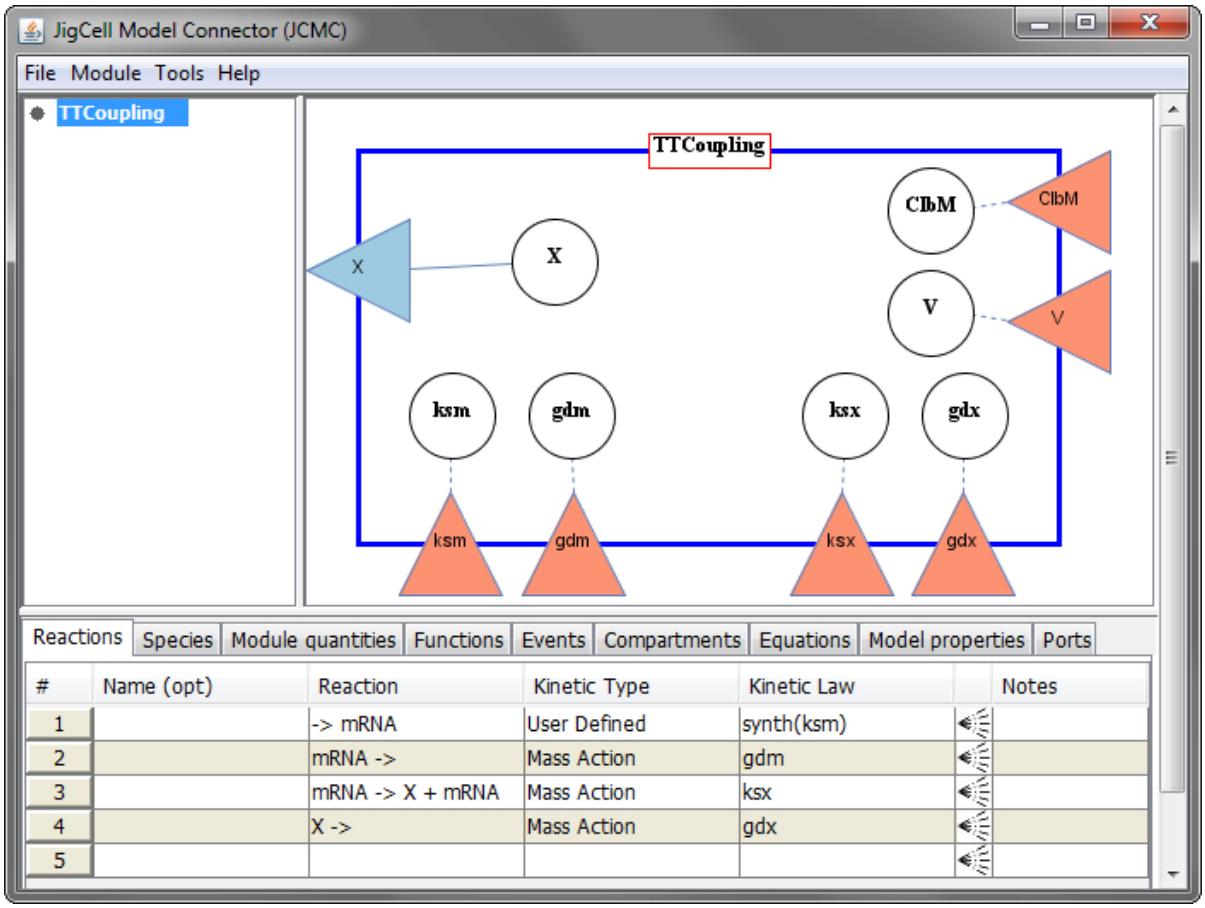


Figure 5.3: Transcription and translation coupling module TTCoupling

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 half of their current values.

5.2.2 *ClbM* Regulation

Step 1 of the model will consist of the interactions shown in Figure 5.4.

The regulation of $Cdh1$ and $ClbM$ play big roles in step 1. $Cdh1$ has an unphosphorylated state as well as ten phosphorylated states, for a total of eleven phosphorylation states. In this model, only the unphosphorylated state of $Cdh1$ is active. $ClbM$ affects many parts of this model. In step 1, $ClbM$ and the active form of $Cdh1$ down-regulate each other. To better understand these interactions, let's start with $Cdh1$.

Figure 5.5 displays module Cdh1PD, short for Cdh1 Phosphorylation/Dephosphorylation. In the ModelBuilder panel, the phosphorlation and dephosphorylation reactions are shown. Degradation reactions for the phosphorylated states are also present. The rate constants

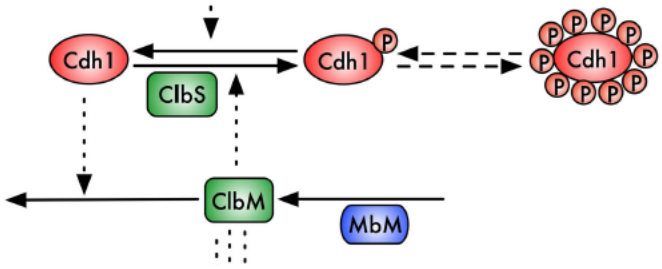
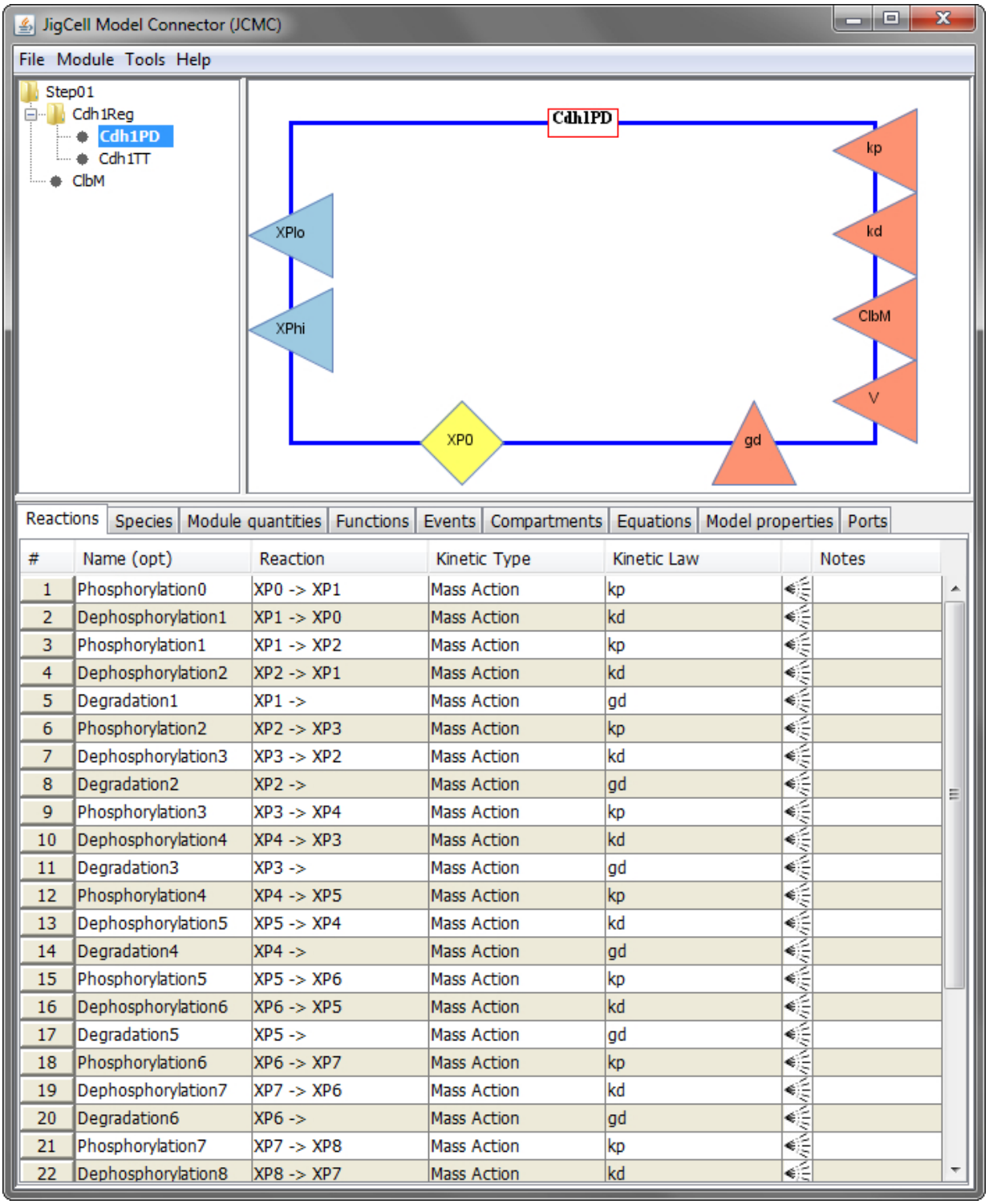


Figure 5.4: Step 1 of the model

for the reactions (kp, kd, and gd) are linked to input ports. The species $XPlo$ and $XPhi$ are both linked to output ports. $XPhi$ is a summation from $XP1$ to $XP10$ and $XPlo$ is equivalent to the remaining state, $XP0$. The two remaining input ports link to species V and $ClbM$, which are used to calculate when the cell divides.



Reactions | Species | Module quantities | Functions | Events | Compartments | Equations | Model properties | Ports

#	Name (opt)	Reaction	Kinetic Type	Kinetic Law	Notes
1	Phosphorylation0	XP0 -> XP1	Mass Action	kp	
2	Dephosphorylation1	XP1 -> XP0	Mass Action	kd	
3	Phosphorylation1	XP1 -> XP2	Mass Action	kp	
4	Dephosphorylation2	XP2 -> XP1	Mass Action	kd	
5	Degradation1	XP1 ->	Mass Action	gd	
6	Phosphorylation2	XP2 -> XP3	Mass Action	kp	
7	Dephosphorylation3	XP3 -> XP2	Mass Action	kd	
8	Degradation2	XP2 ->	Mass Action	gd	
9	Phosphorylation3	XP3 -> XP4	Mass Action	kp	
10	Dephosphorylation4	XP4 -> XP3	Mass Action	kd	
11	Degradation3	XP3 ->	Mass Action	gd	
12	Phosphorylation4	XP4 -> XP5	Mass Action	kp	
13	Dephosphorylation5	XP5 -> XP4	Mass Action	kd	
14	Degradation4	XP4 ->	Mass Action	gd	
15	Phosphorylation5	XP5 -> XP6	Mass Action	kp	
16	Dephosphorylation6	XP6 -> XP5	Mass Action	kd	
17	Degradation5	XP5 ->	Mass Action	gd	
18	Phosphorylation6	XP6 -> XP7	Mass Action	kp	
19	Dephosphorylation7	XP7 -> XP6	Mass Action	kd	
20	Degradation6	XP6 ->	Mass Action	gd	
21	Phosphorylation7	XP7 -> XP8	Mass Action	kp	
22	Dephosphorylation8	XP8 -> XP7	Mass Action	kd	

Figure 5.5: Cdh1 phosphorylation and dephosphorylation module

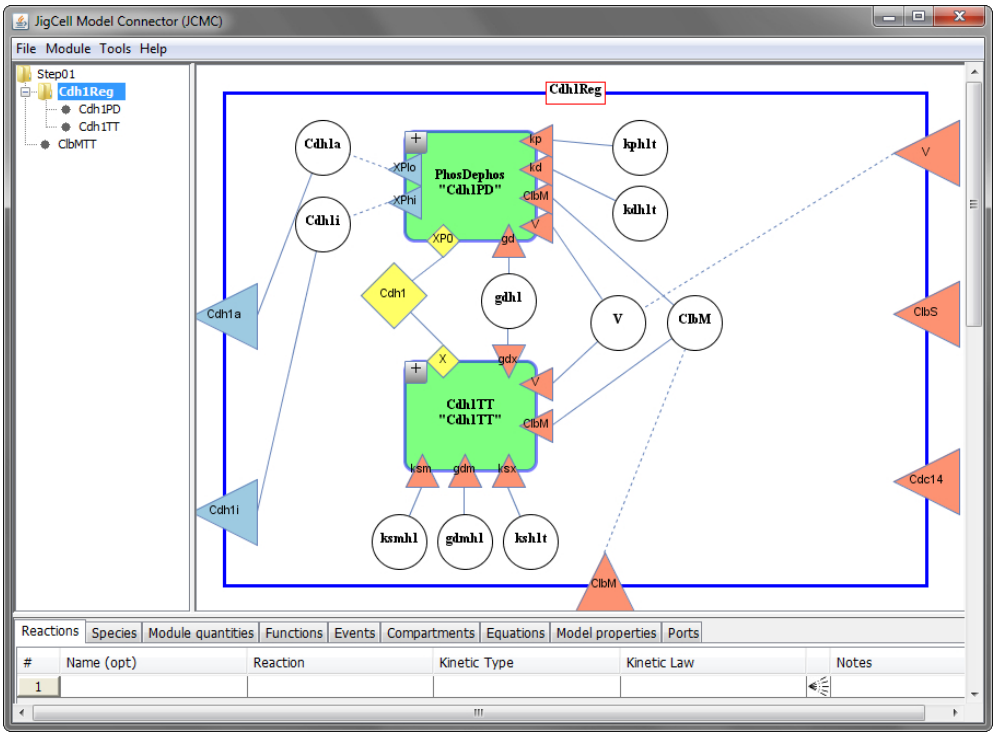


Figure 5.6: Cdh1 regulation module

Figure 5.6 displays module Cdh1Reg, short for Cdh1 Regulation. Cdh1Reg contains submodules Cdh1PD and Cdh1TT. Visible variable nodes $kph1t$, $kdh1t$, and $gdh1$ connect the module quantities to Cdh1PD's input ports kp, kd, and gd. Those ports link to Cdh1PD's phosphoryation, dephosphorylation, and degradation reaction rates. Cdh1TT, short for Cdh1 Transcription and Translation Coupling, is similar to module TTCoupling in Figure 5.3. The only difference is that Cdh1TT has an equivalence port linked to internal species X. In Cdh1Reg, equivalence node $Cdh1$ connects the species to equivalence port X on submodule Cdh1TT and $XP0$ on submodule Cdh1PD. $Cdh1$ is synthesized in Cdh1TT and phosphorylated in Cdh1PD. Since $Cdh1$ is modified in both submodules, an equivalence node is necessary. Species $Cdh1a$ and $Cdh1i$ represent the active and inactive forms of Cdh1. They receive incoming connections from output ports on submodule Cdh1PD and have outgoing connections to output ports on module Cdh1Reg. Input ports $ClbS$ and $Cdc14$ link to species used to calculate rate constants $kph1t$ and $kdh1t$.

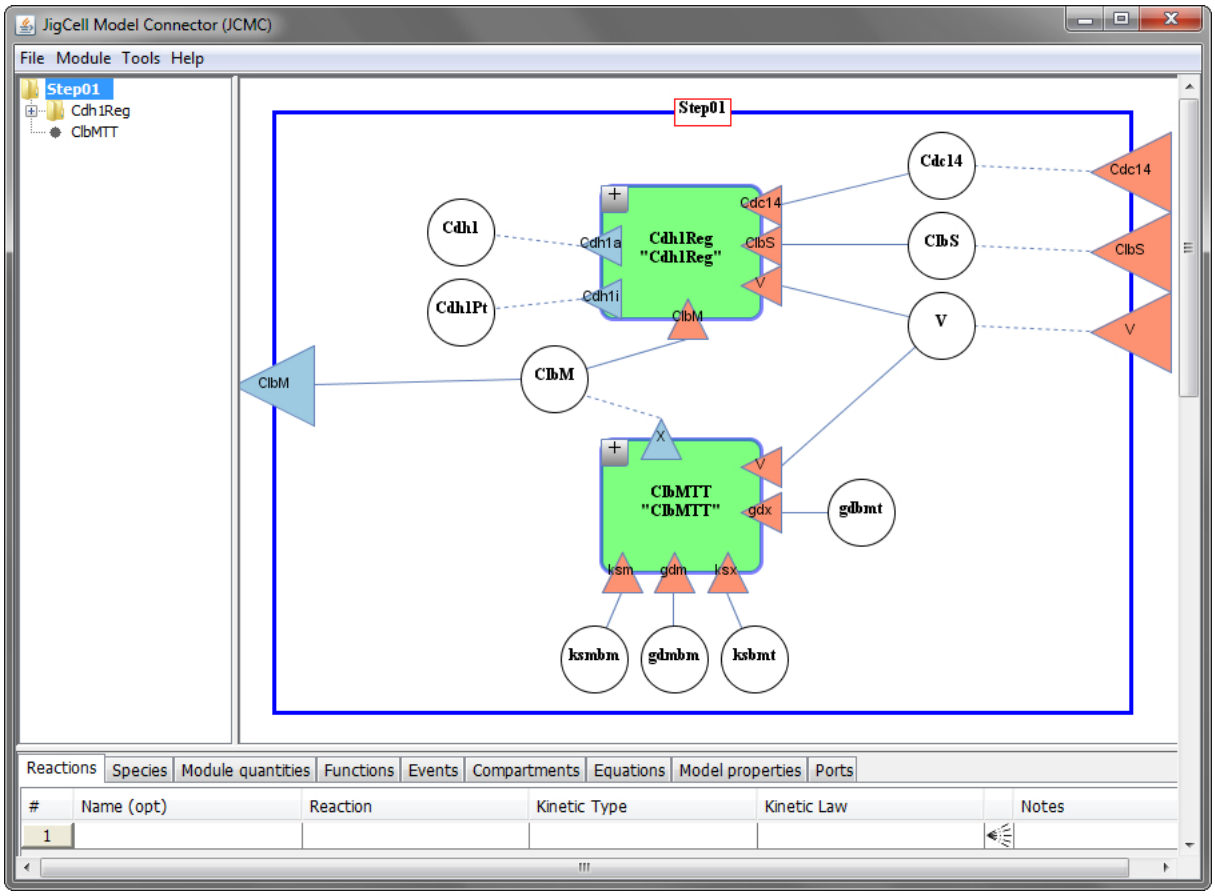


Figure 5.7: Step01 module in JCMC

Figure 5.7 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 $gdbmt$. Visible variable nodes $ksmbm$, $gdmbm$, $ksbmt$, and $gdbmt$ connect the module quantities to ClbMTT's input ports ksm, gdm, ksx, 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. This concludes building module Step01.

5.2.3 *Cdc14* Regulation

Step 2 of the model will consist of the interactions shown in Figure 5.8. 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.

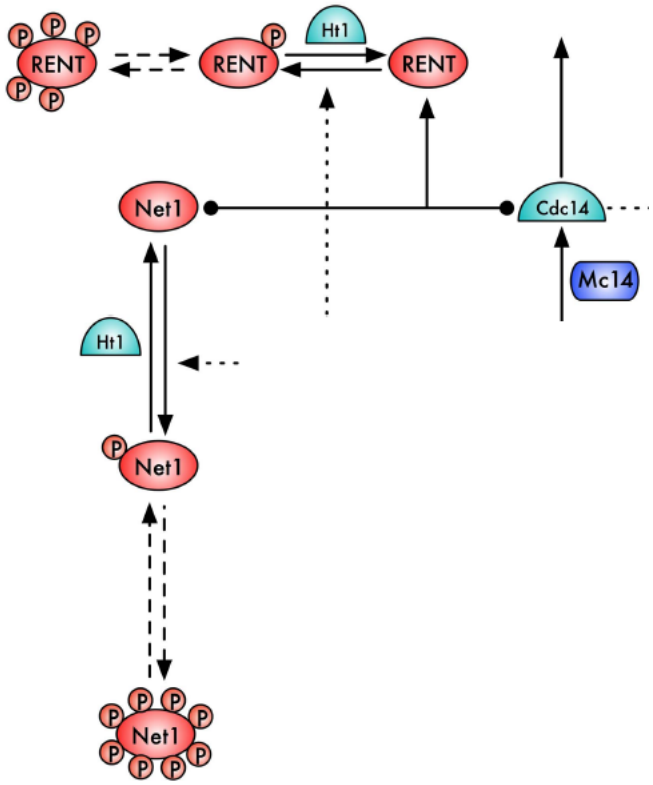


Figure 5.8: Step 2 of the model

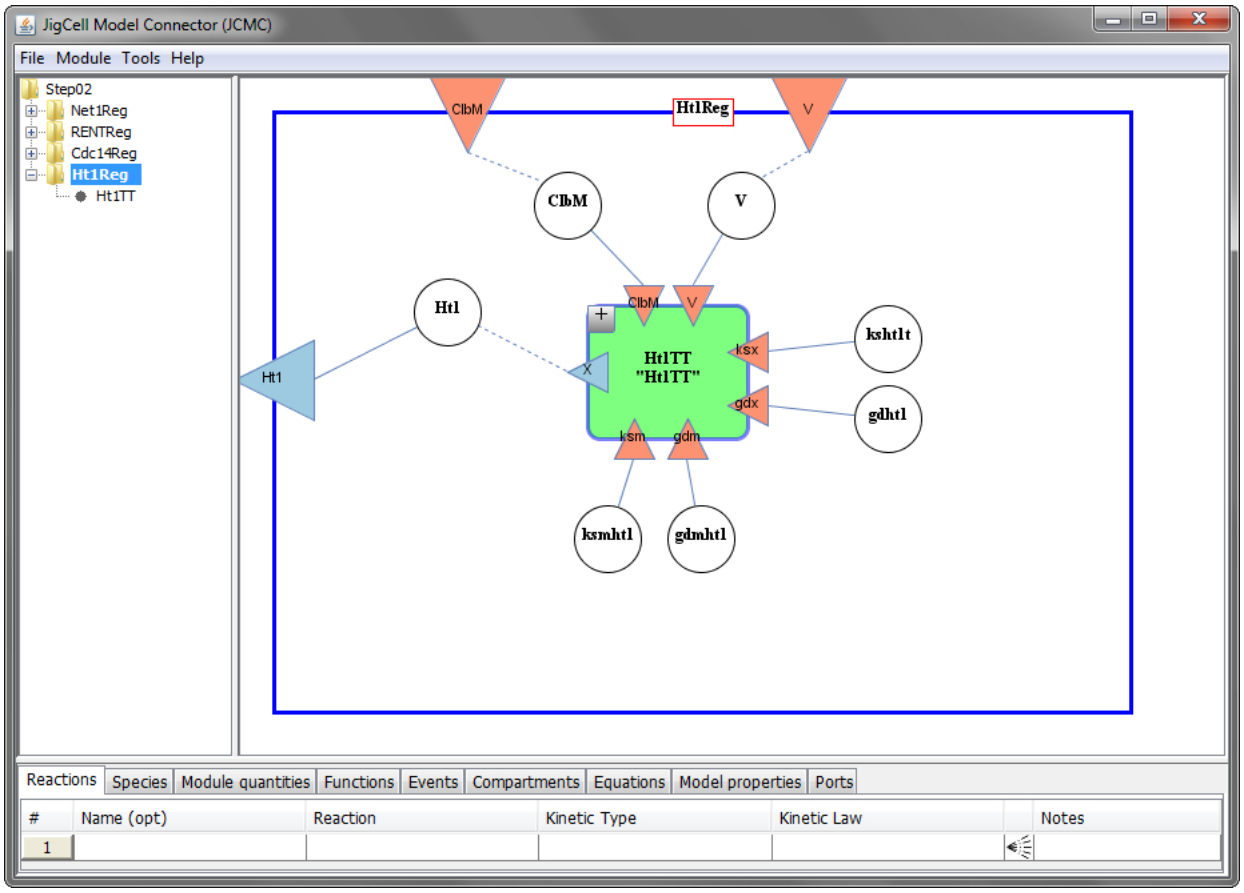


Figure 5.9: Ht1 regulation module

Figure 5.9 displays module Ht1Reg. Ht1Reg contains the submodule Ht1TT. Ht1TT is similar to module TTCoupling in Figure 5.3. Visible variable node *Ht*1 connects an output port on submodule Ht1TT to an output port on module Ht1Reg, so it can be referenced outside of the module.

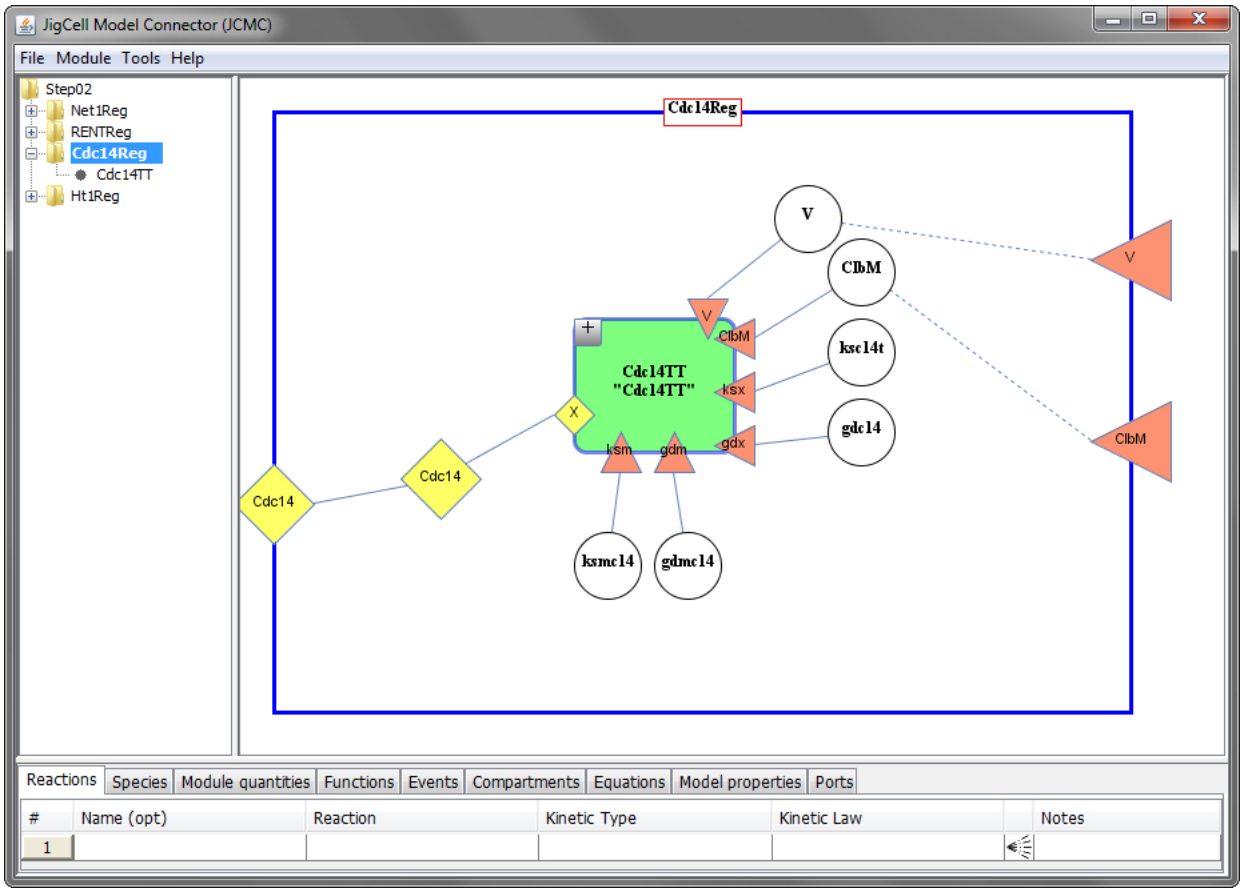


Figure 5.10: Cdc14 regulation module

Figure 5.10 displays module Cdc14Reg, which follows the same pattern as Ht1Reg. Cdc14Reg contains the submodule Cdc14TT. Cdc14TT is slightly different than module TTCoupling in Figure 5.3 because Cdc14TT has an equivalence port. Equivalence node *Cdc*14 connects the equivalence port on submodule Cdc14TT to the equivalence port on module Cdc14Reg, so it can be modified outside of the module.

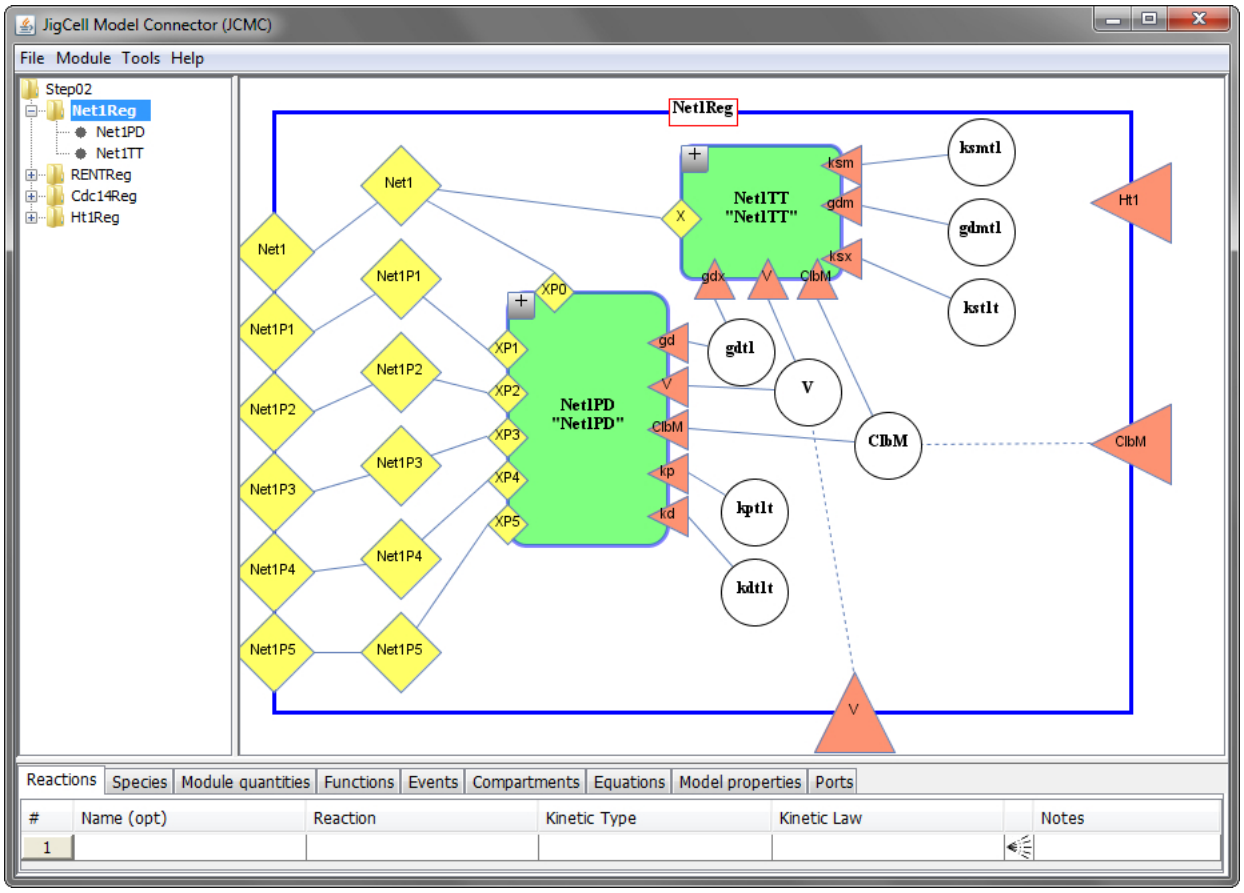
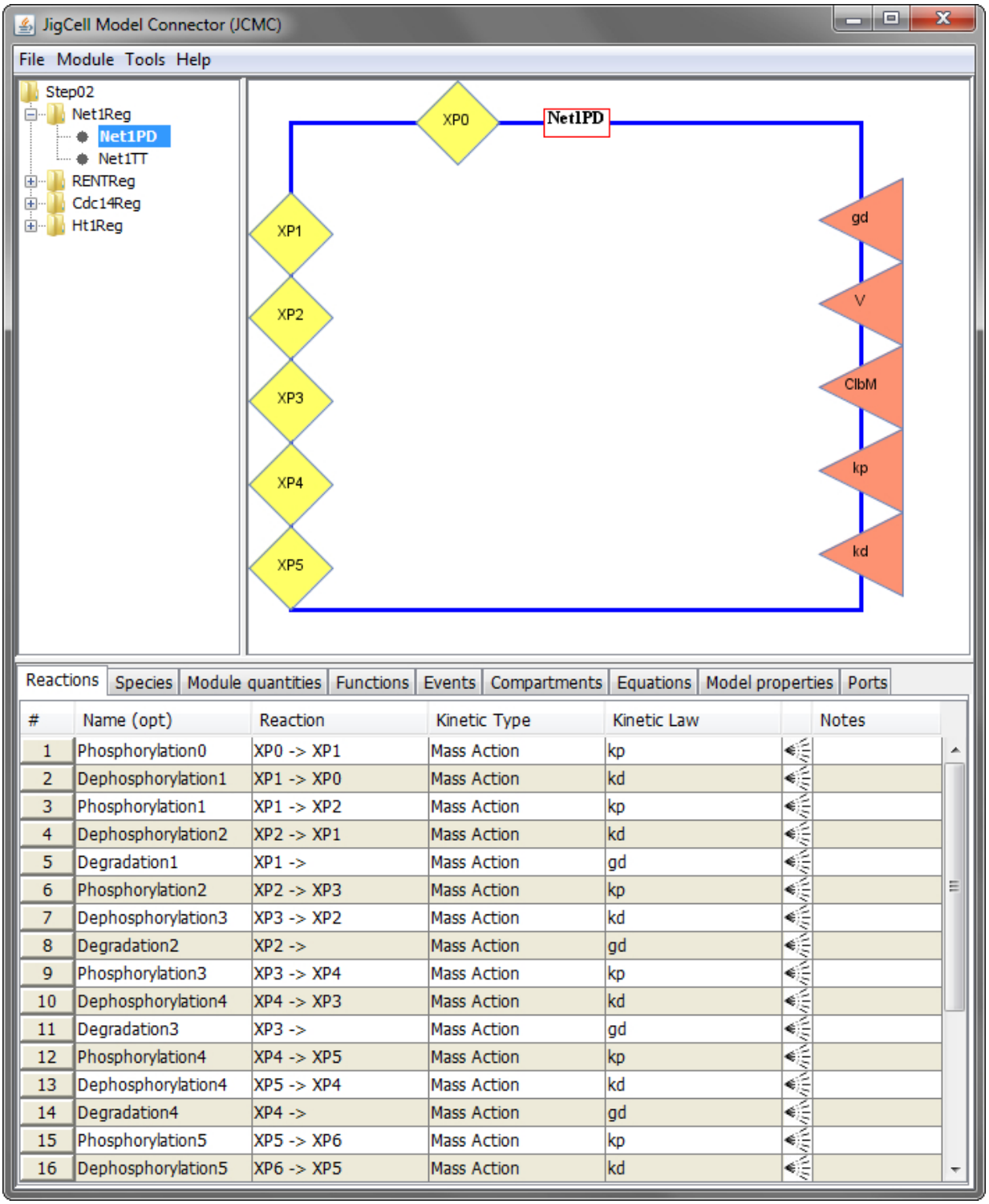


Figure 5.11: Net1 regulation module

Figure 5.11 displays module Net1Reg. Net1Reg contains submodules Net1TT and Net1PD. Net1TT is similar to module TTCoupling in Figure 5.3. Net1PD has phosphorylation and dephosphorylation reactions for the different Net1 states. Details are shown in Figure 5.12. In Net1Reg, equivalence node $Net1$ connects the species to both submodules. $Net1$ is translated in Net1TT and phosphorylated in Net1PD. Nodes $Net1$, $Net1P1$, $Net1P2$, $Net1P3$, $Net1P4$, and $Net1P5$ are all connected to equivalence ports on module Net1Reg, so they can be modified outside of the module.



Reactions | Species | Module quantities | Functions | Events | Compartments | Equations | Model properties | Ports

#	Name (opt)	Reaction	Kinetic Type	Kinetic Law	Notes
1	Phosphorylation0	XP0 -> XP1	Mass Action	kp	
2	Dephosphorylation1	XP1 -> XP0	Mass Action	kd	
3	Phosphorylation1	XP1 -> XP2	Mass Action	kp	
4	Dephosphorylation2	XP2 -> XP1	Mass Action	kd	
5	Degradation1	XP1 ->	Mass Action	gd	
6	Phosphorylation2	XP2 -> XP3	Mass Action	kp	
7	Dephosphorylation3	XP3 -> XP2	Mass Action	kd	
8	Degradation2	XP2 ->	Mass Action	gd	
9	Phosphorylation3	XP3 -> XP4	Mass Action	kp	
10	Dephosphorylation4	XP4 -> XP3	Mass Action	kd	
11	Degradation3	XP3 ->	Mass Action	gd	
12	Phosphorylation4	XP4 -> XP5	Mass Action	kp	
13	Dephosphorylation4	XP5 -> XP4	Mass Action	kd	
14	Degradation4	XP4 ->	Mass Action	gd	
15	Phosphorylation5	XP5 -> XP6	Mass Action	kp	
16	Dephosphorylation5	XP6 -> XP5	Mass Action	kd	

Figure 5.12: Net1 phosphorylation and dephosphorylation module

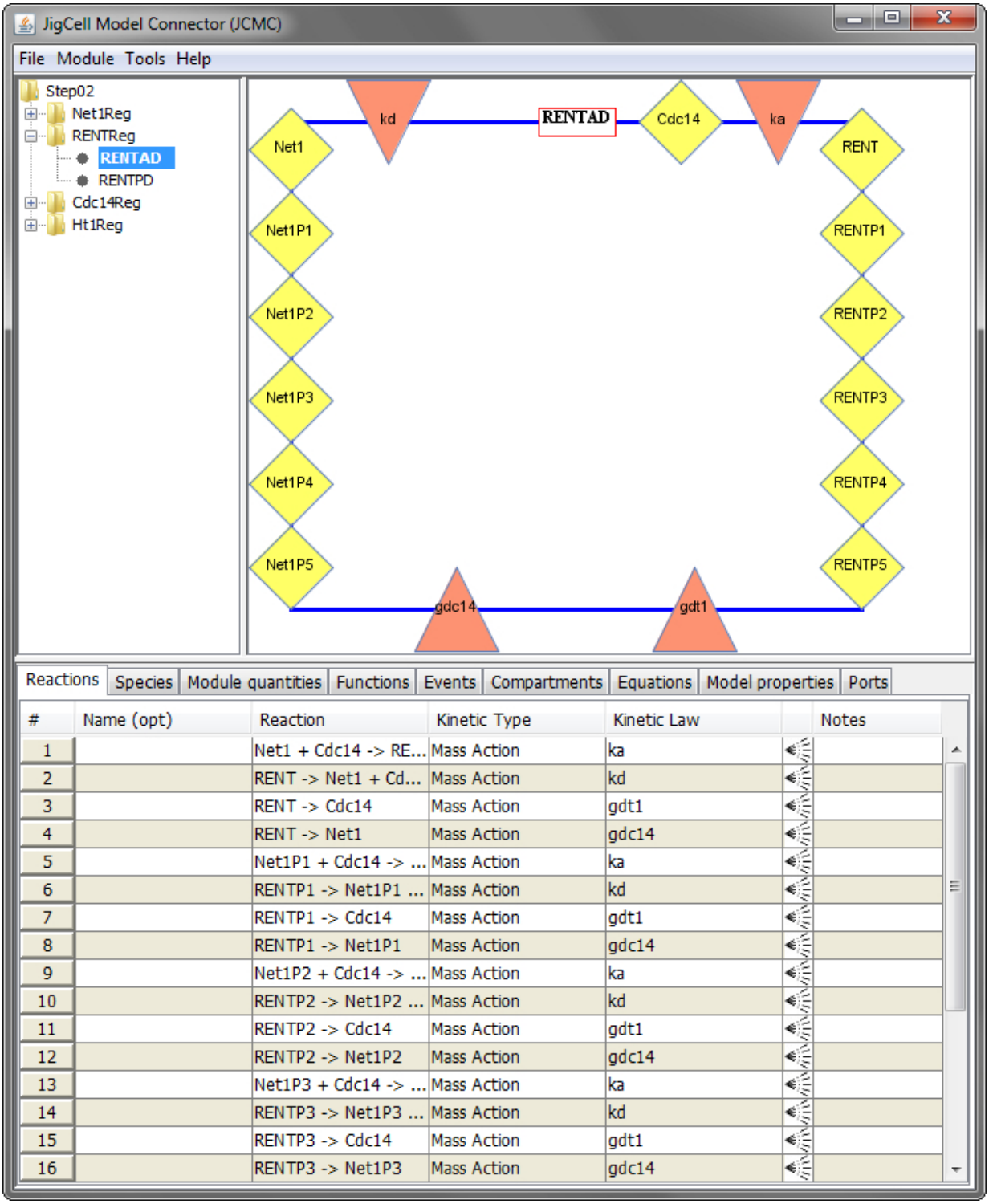


Figure 5.13: RENT association and dissociation module

Figure 5.13 displays module RENTAD, short for RENT Association/Dissociation. Different phosphorylation states of $Net1$ combine with $Cdc14$ to form different phosphorylation states of $RENT$. $Cdc14$, each $Net1$ state, and each $RENT$ state are linked to equivalence ports, so they can be modified inside and outside the module.

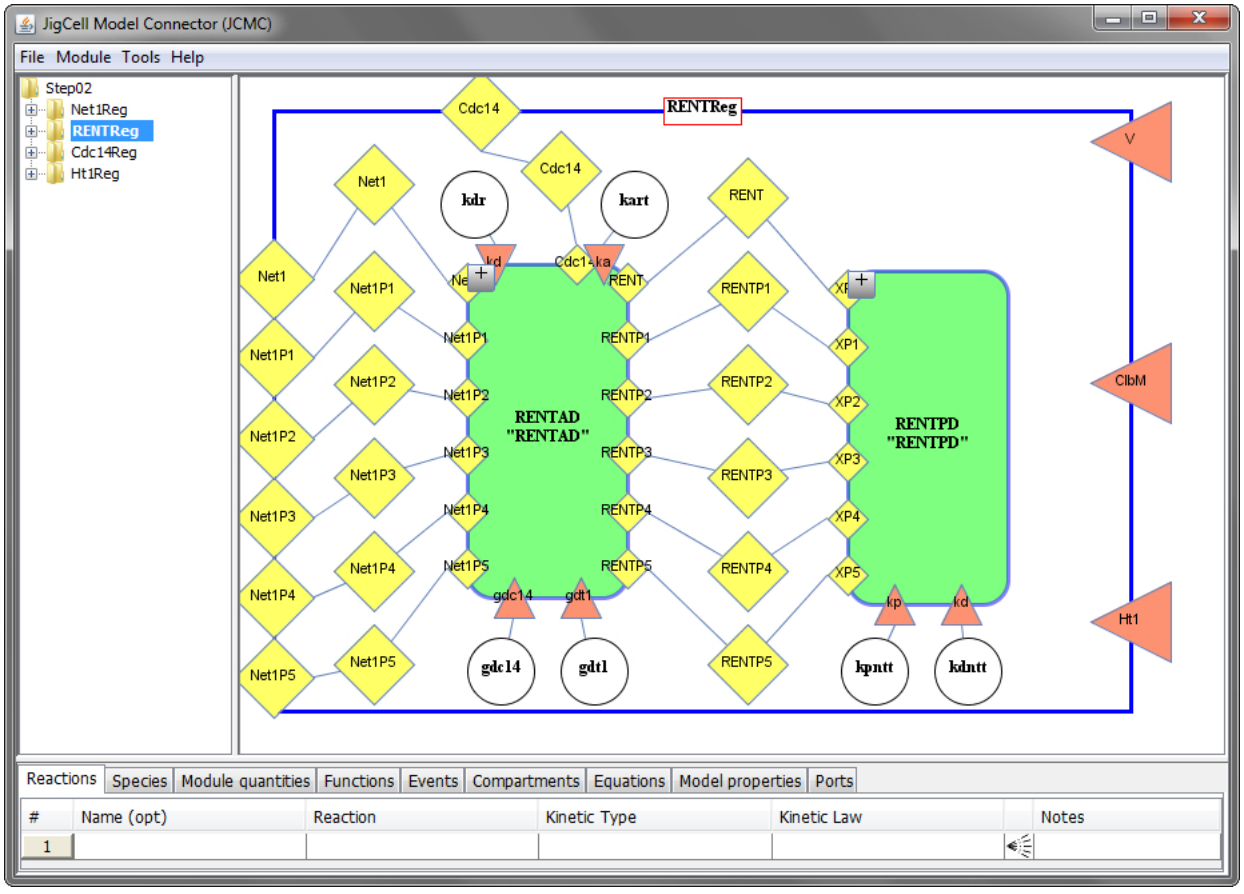
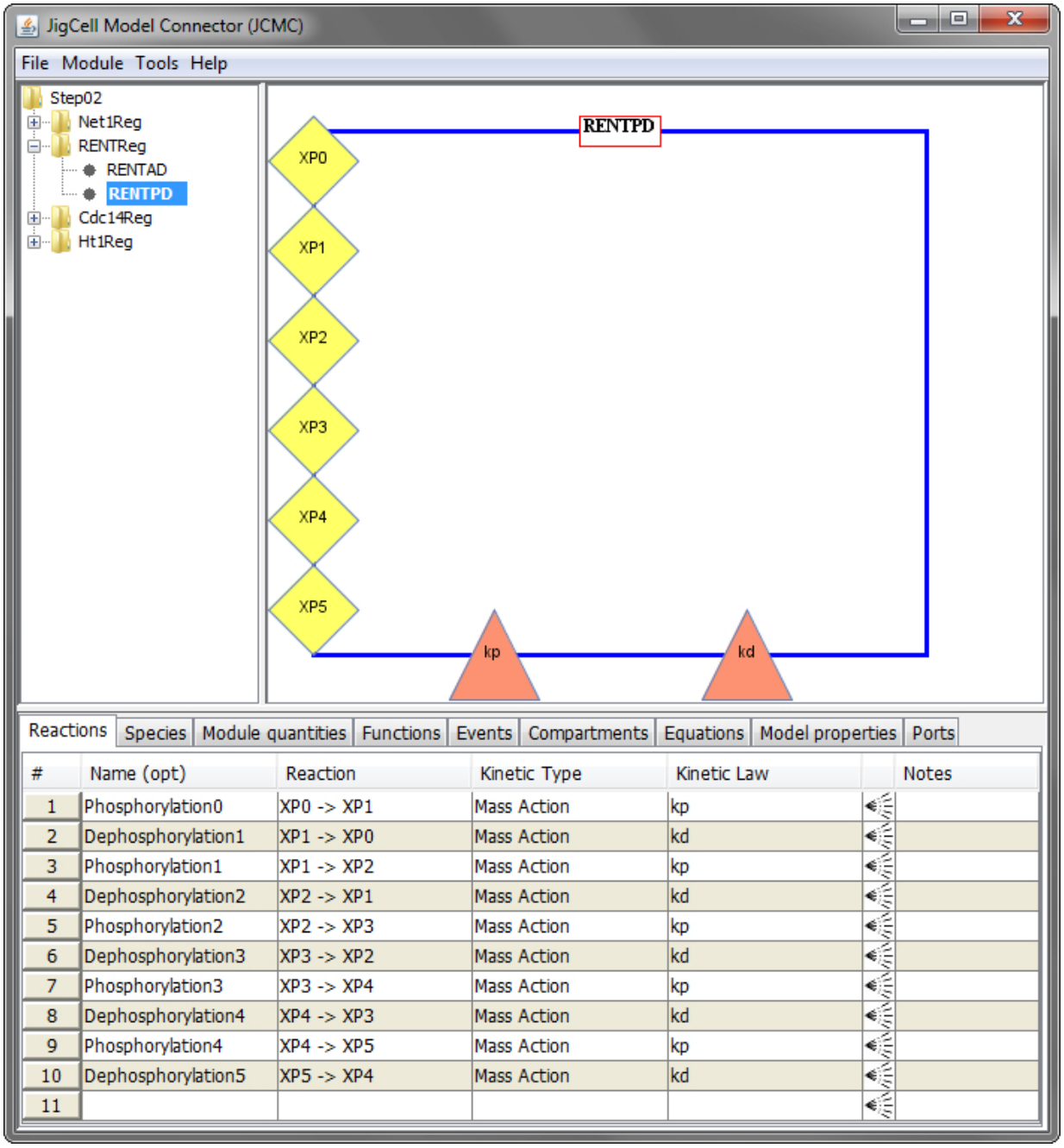


Figure 5.14: RENT regulation module

Figure 5.14 displays module RENTReg. RENTReg contains submodules RENTPD and RENTAD. RENTPD has the phosphorylation and dephosphorylation reactions for the different $RENT$ states. Details are shown in Figure 5.15. In RENTReg, equivalence nodes for each of the $RENT$ phosphorylation states connect to both submodules. This is necessary because the $RENT$ states are modified in both submodules. Equivalence nodes for $Cdc14$ and each of the $Net1$ states connect ports on RENTAD to equivalence ports on RENTReg, so they can be modified outside of the module.



#	Name (opt)	Reaction	Kinetic Type	Kinetic Law	Notes
1	Phosphorylation0	XP0 -> XP1	Mass Action	kp	
2	Dephosphorylation1	XP1 -> XP0	Mass Action	kd	
3	Phosphorylation1	XP1 -> XP2	Mass Action	kp	
4	Dephosphorylation2	XP2 -> XP1	Mass Action	kd	
5	Phosphorylation2	XP2 -> XP3	Mass Action	kp	
6	Dephosphorylation3	XP3 -> XP2	Mass Action	kd	
7	Phosphorylation3	XP3 -> XP4	Mass Action	kp	
8	Dephosphorylation4	XP4 -> XP3	Mass Action	kd	
9	Phosphorylation4	XP4 -> XP5	Mass Action	kp	
10	Dephosphorylation5	XP5 -> XP4	Mass Action	kd	
11					

Figure 5.15: RENT phosphorylation and dephosphorylation module

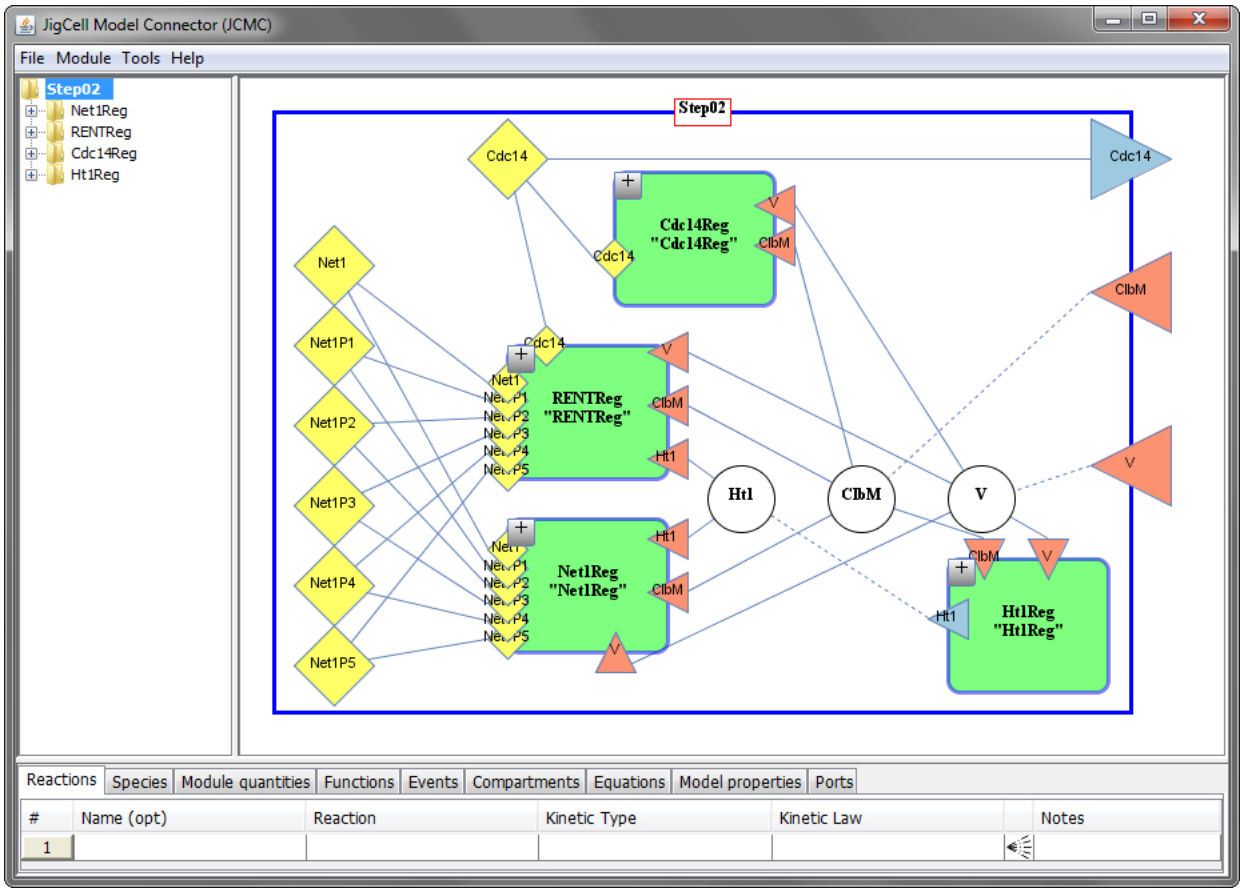


Figure 5.16: Step02 module in JCMC

Figure 5.16 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. This concludes building module Step02.

5.2.4 *SBF* Regulation

Step 3 of the model will consist of the interactions shown in Figure 5.17. 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.

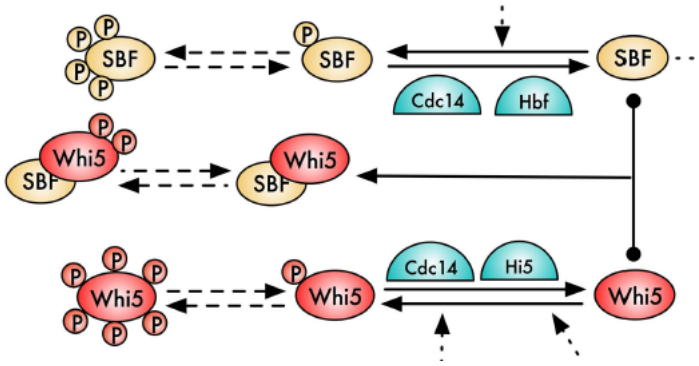


Figure 5.17: Step 3 of the model

Modules Hi5Reg and HbfReg both have structures similar to Ht1Reg from step 2. Each module contains a transcription and translation submodule that has input ports for its reaction rates. Visible variable nodes connect module quantities to the input ports on the submodule. Details for Hi5Reg and HbfReg are shown in Figures 5.18 and 5.19.

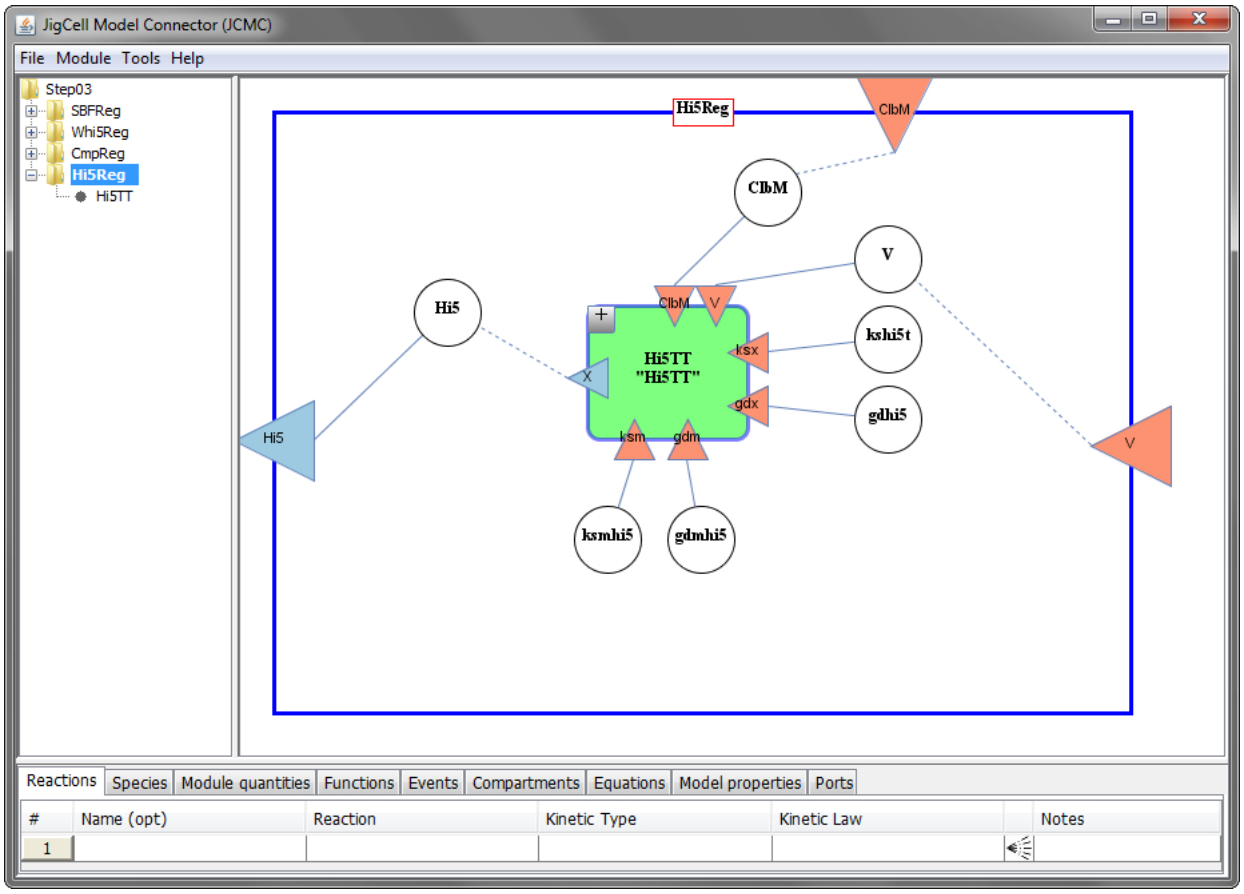


Figure 5.18: Hi5 regulation module

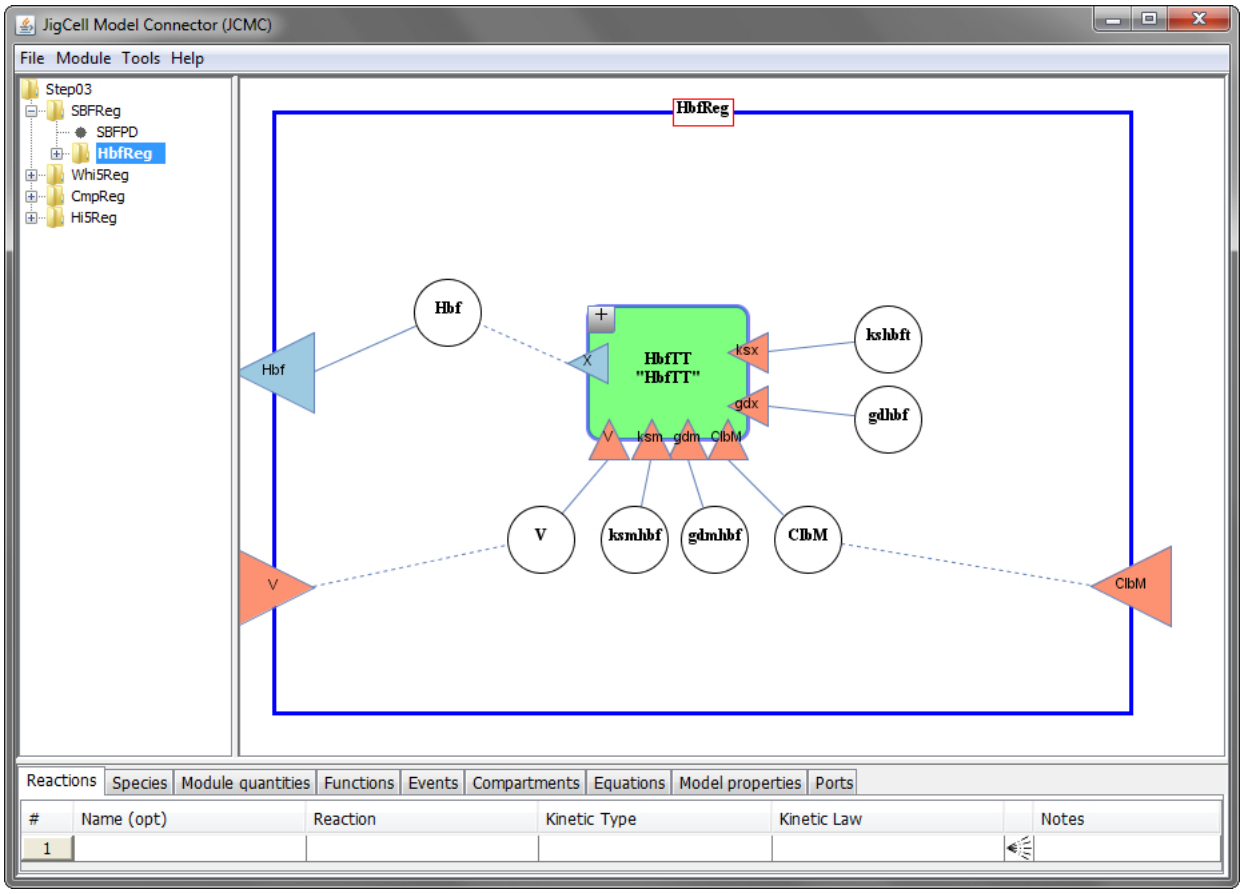


Figure 5.19: Hbf regulation module

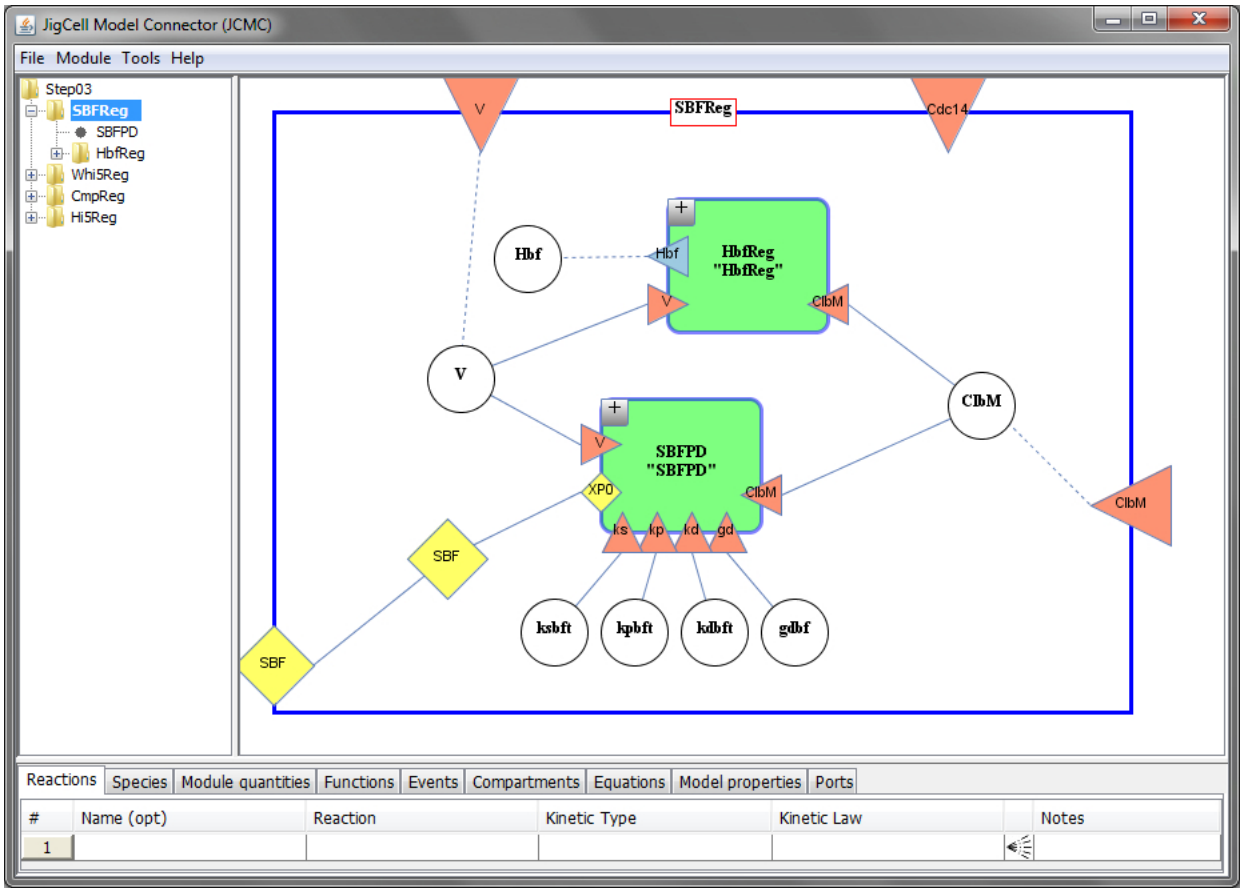
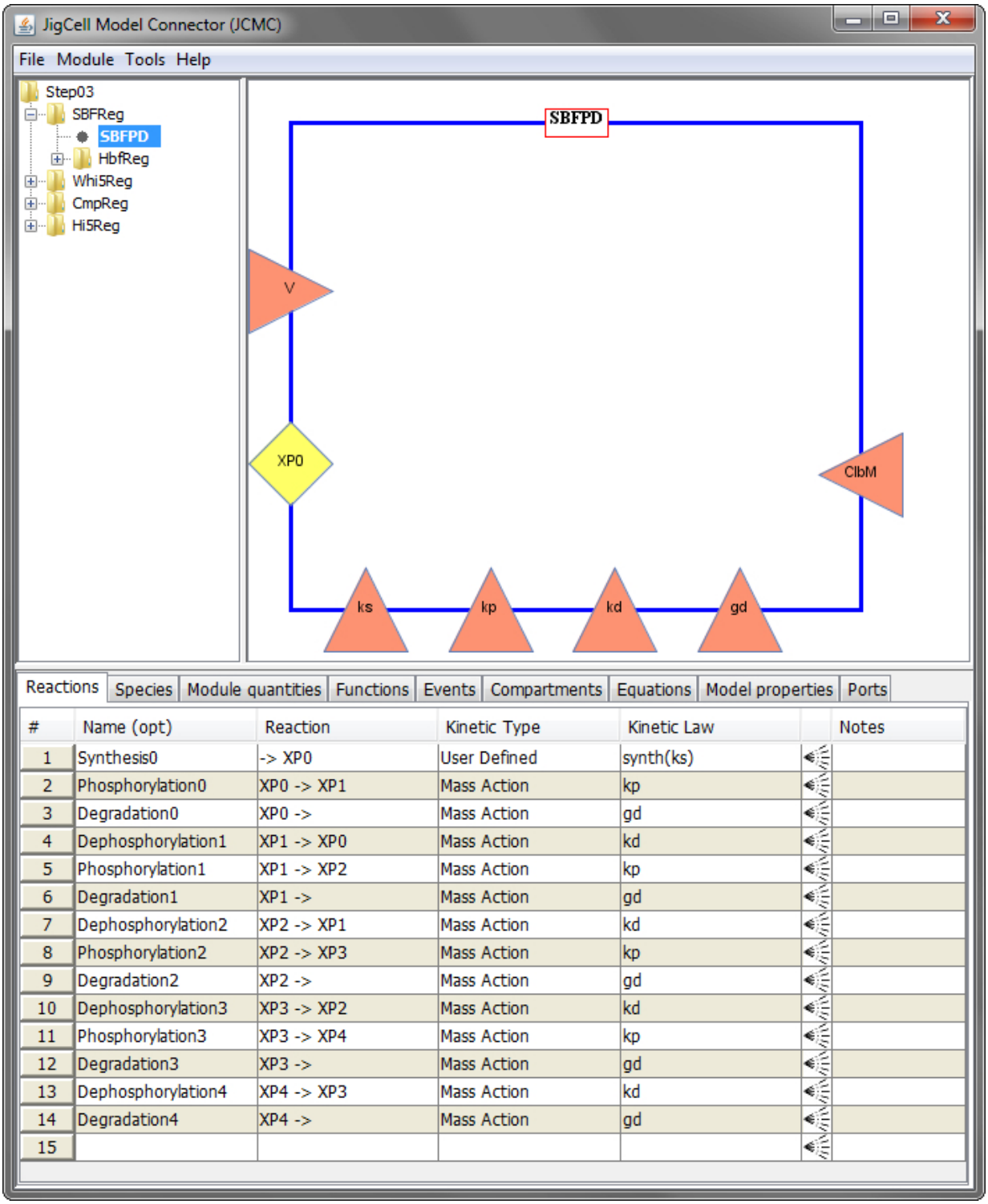


Figure 5.20: SBF regulation module

Figure 5.20 displays module SBFReg. SBFReg contains submodules HbfReg and SBFPD. Visible variable node Hbf connects to HbfReg's output port. Hbf receives a value from the port and is used to calculate module quantity $kdbft$. SBFPD holds the synthesis, degradation, phosphoryation, and dephosphorylation reactions for SBF. Details are shown in Figure 5.21. SBFPD has input ports that link to its reaction rates. In SBFReg, visible variables nodes $ksbft$, $kpbft$, $kdbft$, and $gdbf$ connect the module quantities to those input ports. Equivalence node SBF connects the species to submodule SBFPD. Node SBF also connects to an equivalence port on module SBFReg, so it can be modified outside of the module.



#	Name (opt)	Reaction	Kinetic Type	Kinetic Law	Notes
1	Synthesis0	-> XP0	User Defined	synth(ks)	
2	Phosphorylation0	XP0 -> XP1	Mass Action	kp	
3	Degradation0	XP0 ->	Mass Action	gd	
4	Dephosphorylation1	XP1 -> XP0	Mass Action	kd	
5	Phosphorylation1	XP1 -> XP2	Mass Action	kp	
6	Degradation1	XP1 ->	Mass Action	gd	
7	Dephosphorylation2	XP2 -> XP1	Mass Action	kd	
8	Phosphorylation2	XP2 -> XP3	Mass Action	kp	
9	Degradation2	XP2 ->	Mass Action	gd	
10	Dephosphorylation3	XP3 -> XP2	Mass Action	kd	
11	Phosphorylation3	XP3 -> XP4	Mass Action	kp	
12	Degradation3	XP3 ->	Mass Action	gd	
13	Dephosphorylation4	XP4 -> XP3	Mass Action	kd	
14	Degradation4	XP4 ->	Mass Action	gd	
15					

Figure 5.21: SBF posphorylation and dephosphorylation module

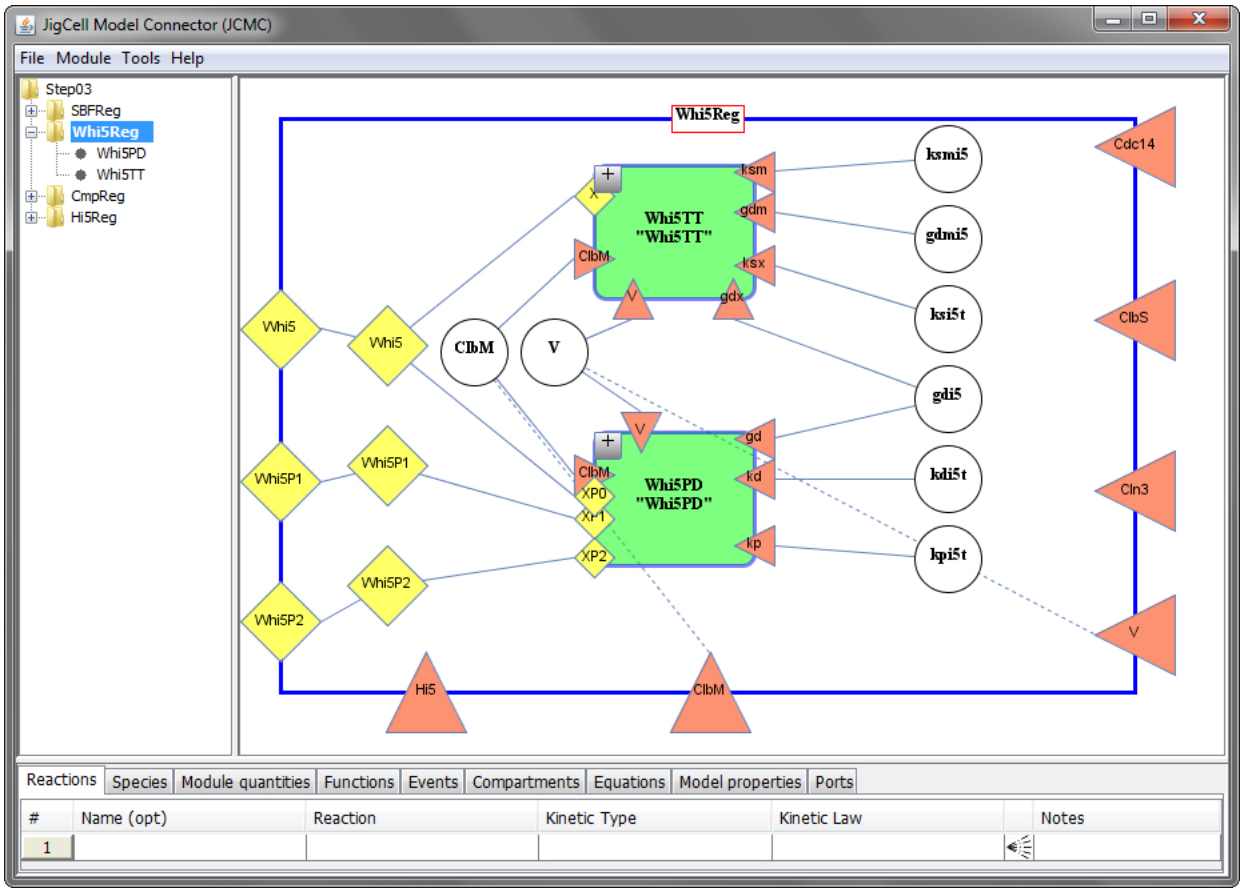
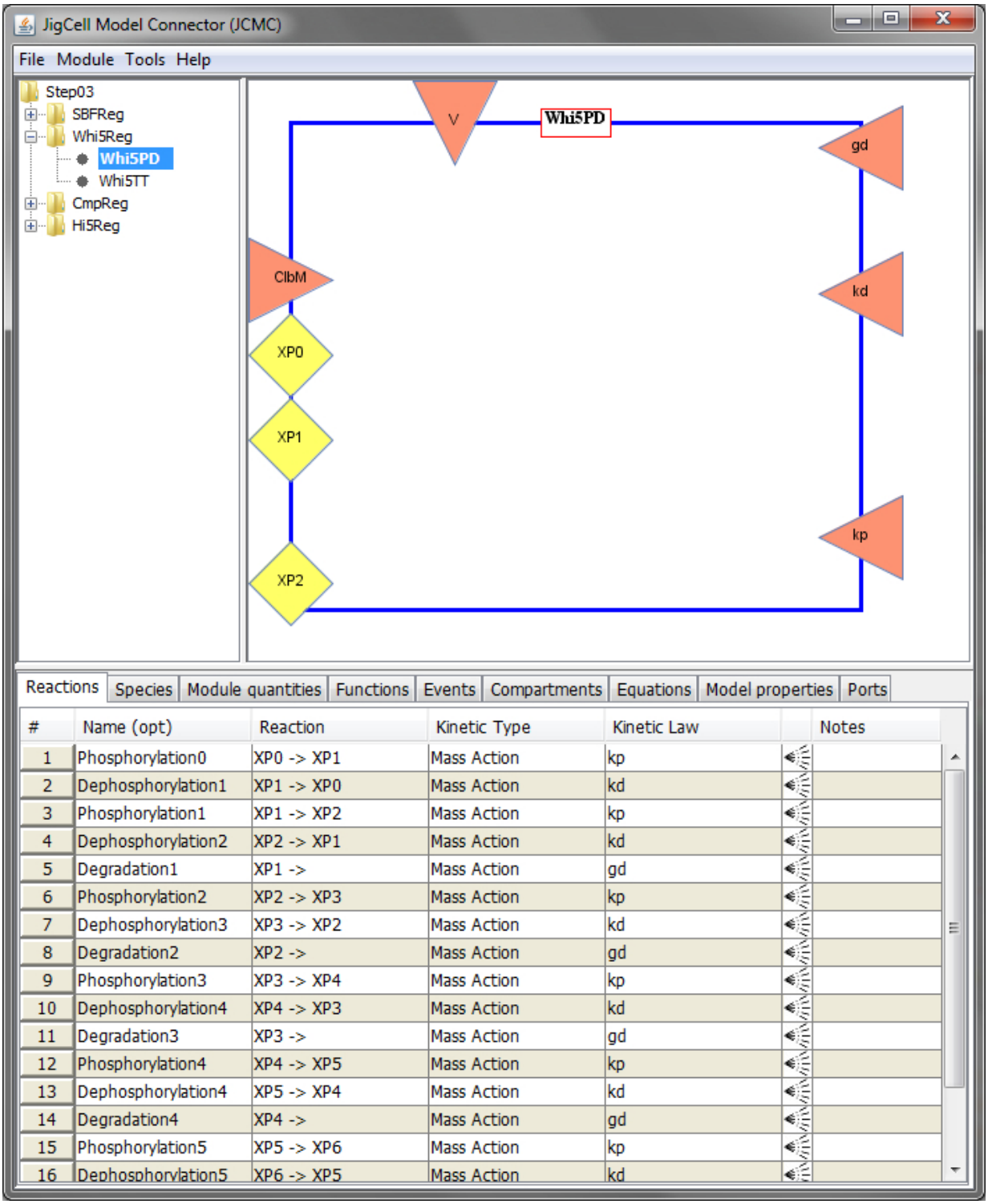


Figure 5.22: Whi5 regulation module

Module Whi5Reg has a structure similar to Net1Reg from step 2. Whi5Reg contains submodules Whi5TT and Whi5PD. It has input ports linked to species that are used to calculate module quantities. These module quantities connect to ports on Whi5TT and Whi5PD, which control their internal reaction rates. Details are shown in Figures 5.22 and 5.23. In Whi5Reg, equivalence nodes $Whi5$, $Whi5P1$, and $Whi5P2$ are all connected to equivalence ports on submodule Whi5PD. The nodes are also connected to equivalence ports on module Whi5Reg, so they can be modified outside of the module.



#	Name (opt)	Reaction	Kinetic Type	Kinetic Law	Notes
1	Phosphorylation0	XP0 -> XP1	Mass Action	kp	
2	Dephosphorylation1	XP1 -> XP0	Mass Action	kd	
3	Phosphorylation1	XP1 -> XP2	Mass Action	kp	
4	Dephosphorylation2	XP2 -> XP1	Mass Action	kd	
5	Degradation1	XP1 ->	Mass Action	gd	
6	Phosphorylation2	XP2 -> XP3	Mass Action	kp	
7	Dephosphorylation3	XP3 -> XP2	Mass Action	kd	
8	Degradation2	XP2 ->	Mass Action	gd	
9	Phosphorylation3	XP3 -> XP4	Mass Action	kp	
10	Dephosphorylation4	XP4 -> XP3	Mass Action	kd	
11	Degradation3	XP3 ->	Mass Action	gd	
12	Phosphorylation4	XP4 -> XP5	Mass Action	kp	
13	Dephosphorylation4	XP5 -> XP4	Mass Action	kd	
14	Degradation4	XP4 ->	Mass Action	gd	
15	Phosphorylation5	XP5 -> XP6	Mass Action	kp	
16	Dephosphorylation5	XP6 -> XP5	Mass Action	kd	

Figure 5.23: Whi5 posphorylation and dephosphorylation module

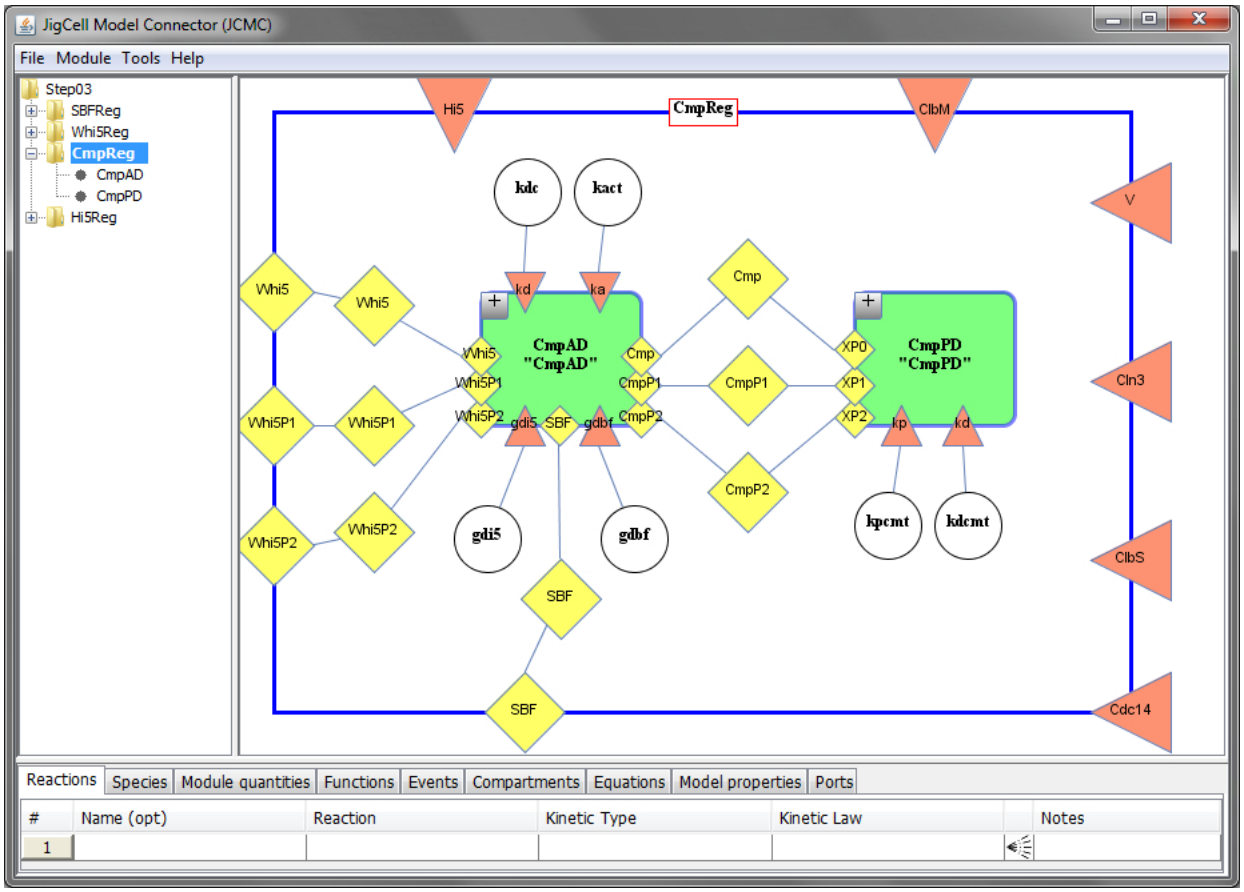


Figure 5.24: Cmp regulation module

Figure 5.24 displays module CmpReg, which has a structure similar to RENTReg from step 2. CmpReg contains submodules CmpPD and CmpAD. CmpPD holds the phosphorylation and dephosphorylation reactions for the different *Cmp* states. Details are shown in Figure 5.25. CmpAD holds the association and dissociation reactions for the different *Cmp* states. Different phosphorylation states of $Whi5$ combine with SBF to form different phosphorylation states of *Cmp*. Details are shown in Figure 5.26. Since the states of *Cmp* are modified in both CmpPD and CmpAD, equivalence nodes for each state connect the submodule ports in CmpReg. Equivalence nodes for SBF and each of the $Whi5$ states connect ports on CmpAD to equivalence ports on CmpReg, so they can be modified outside of the module.

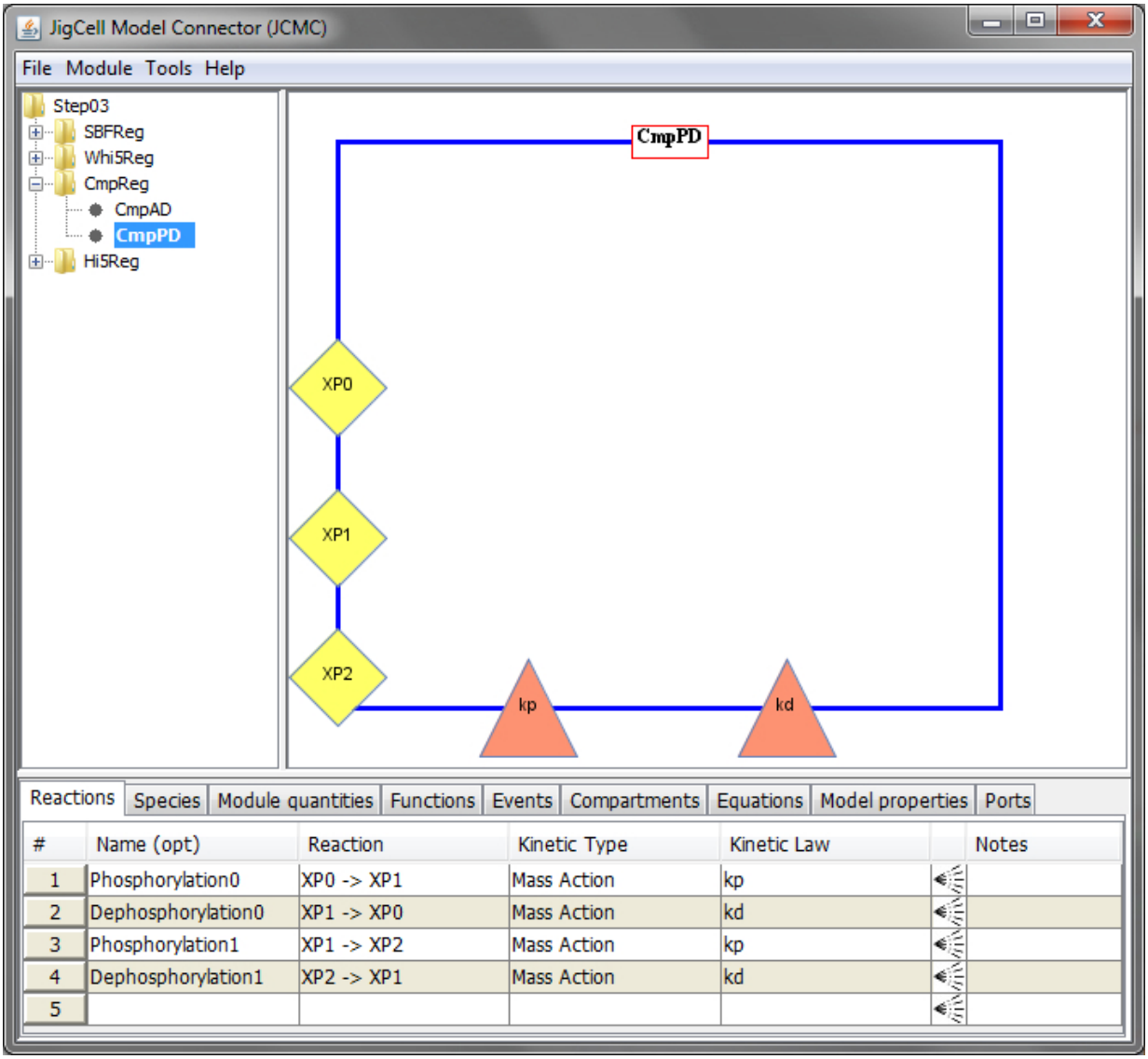
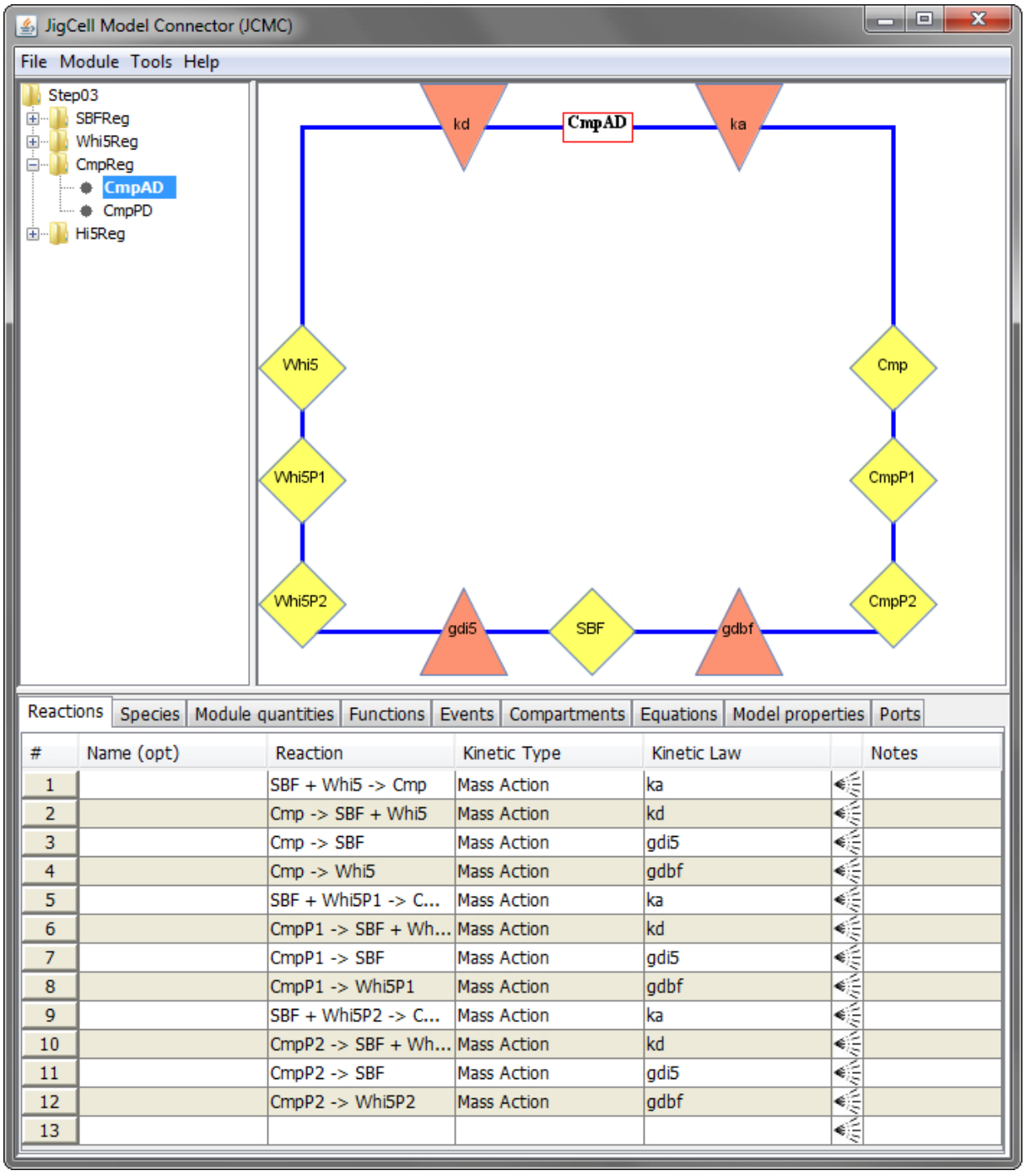


Figure 5.25: Cmp posphorylation and dephosphorylation module



#	Name (opt)	Reaction	Kinetic Type	Kinetic Law	Notes
1		SBF + Whi5 -> Cmp	Mass Action	ka	
2		Cmp -> SBF + Whi5	Mass Action	kd	
3		Cmp -> SBF	Mass Action	gdi5	
4		Cmp -> Whi5	Mass Action	gdbf	
5		SBF + Whi5P1 -> C...	Mass Action	ka	
6		CmpP1 -> SBF + Wh...	Mass Action	kd	
7		CmpP1 -> SBF	Mass Action	gdi5	
8		CmpP1 -> Whi5P1	Mass Action	gdbf	
9		SBF + Whi5P2 -> C...	Mass Action	ka	
10		CmpP2 -> SBF + Wh...	Mass Action	kd	
11		CmpP2 -> SBF	Mass Action	gdi5	
12		CmpP2 -> Whi5P2	Mass Action	gdbf	
13					

Figure 5.26: Cmp association and dissociation module

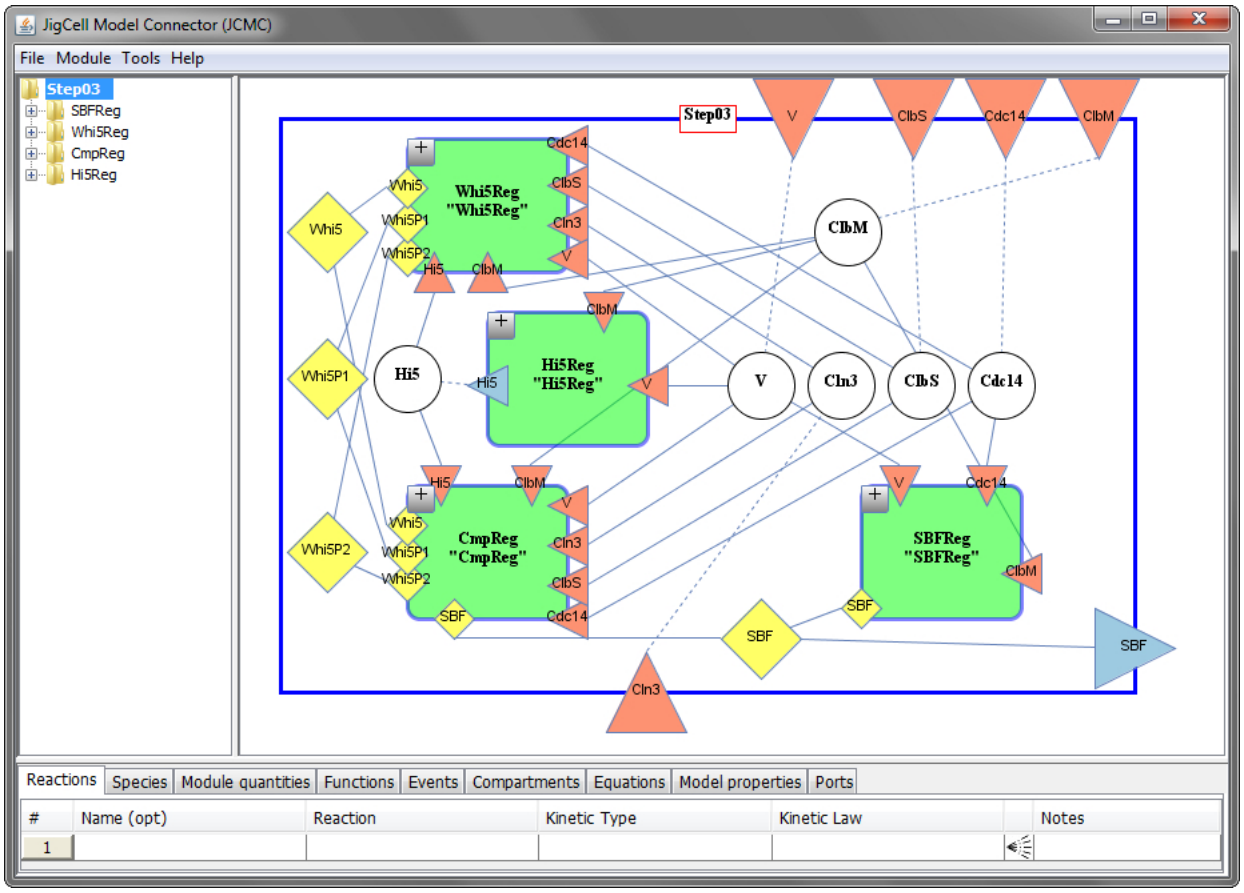


Figure 5.27: Step03 module in JCMC

Figure 5.27 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. This concludes building module Step03.

5.2.5 Last Components

Only a few more modules are required to complete the model. Module Cln3Reg has a structure similar to HT1Reg from step 2. Cln3Reg contains submodule Cln3TT. Visible variable node *Cln3* connects the output port on Cln3TT to an output port on module Cln3Reg, so it can be referenced outside of the module. Details are shown in Figure 5.28.

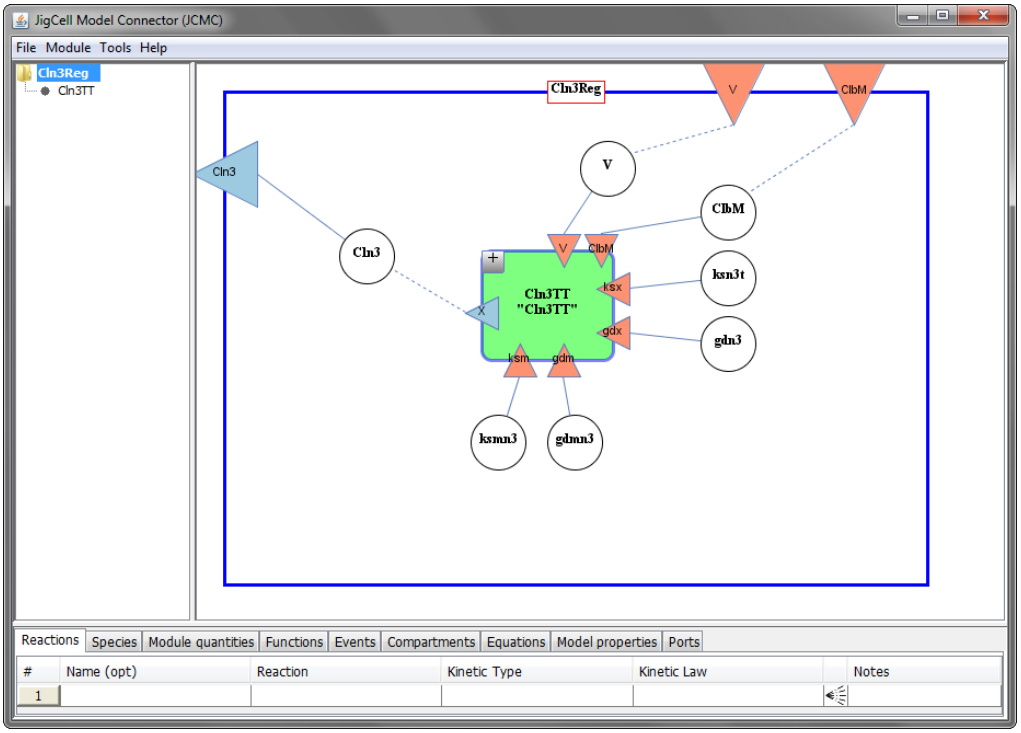


Figure 5.28: Cln3 regulation module

Figure 5.29 displays module ClbSReg. ClbSReg contains submodules ClbTT and GaReg. ClbTT has the same structure as TTCoupling in Figure 5.3. GaReg holds simple synthesis and degradation reactions with rates determined by quantities linked to its input ports. Details are shown in Figure 5.30. In ClbSReg, species *SBF* is linked to an input port and used to calculate module quantity *kagt*. A visible variable node connects the module quantity *kagt* to submodule GaReg's input port *ka*. GaReg's output port connects to visible variable node *Ga*, which is used to calculate module quantity *ksmbmt*. Visible variable node *ClbS* connects the output port on ClbSTT to an output port on module ClbSReg, so it can be referenced outside of the module.

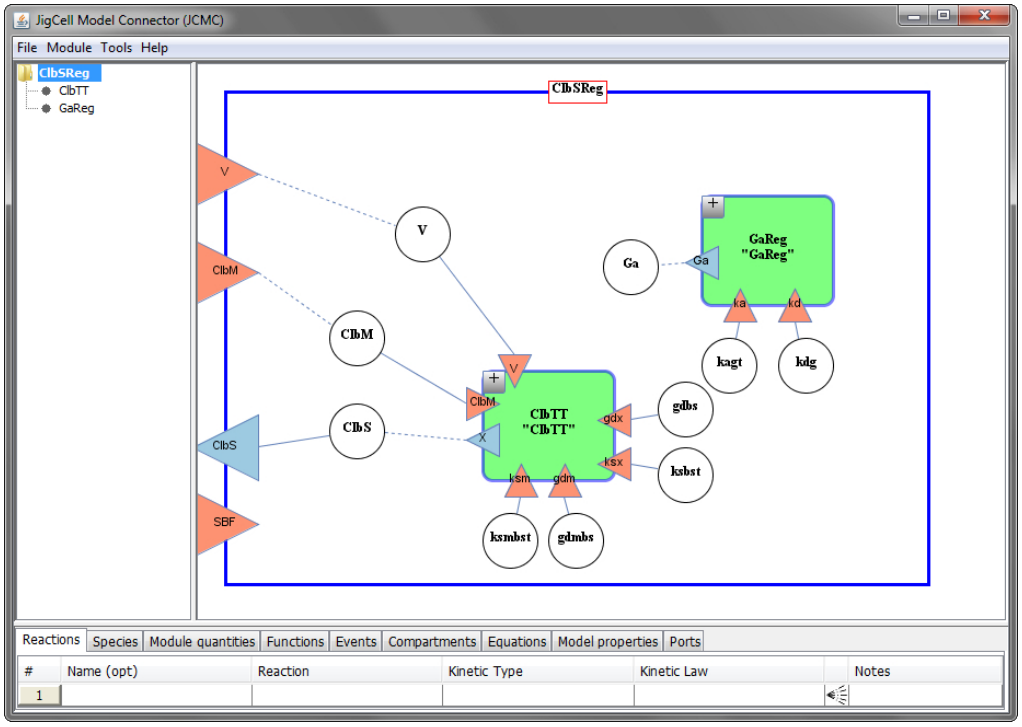


Figure 5.29: ClbS regulation module

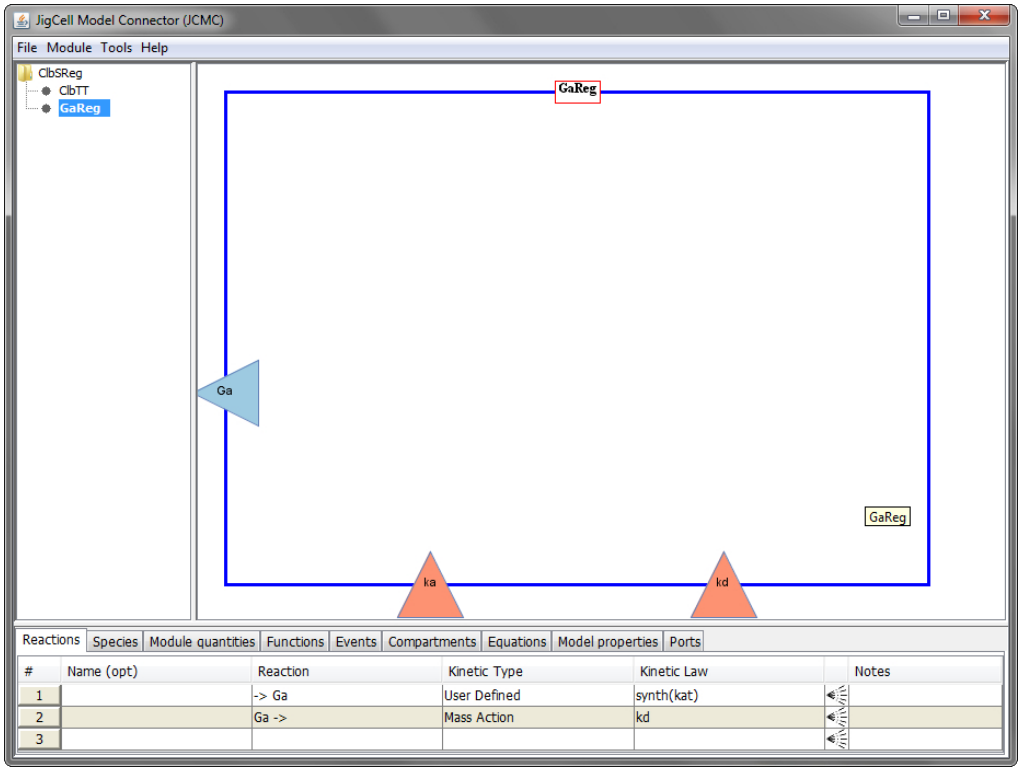


Figure 5.30: Ga regulation module

5.2.6 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. Initally, there are no connections, as shown in Figure 5.31.

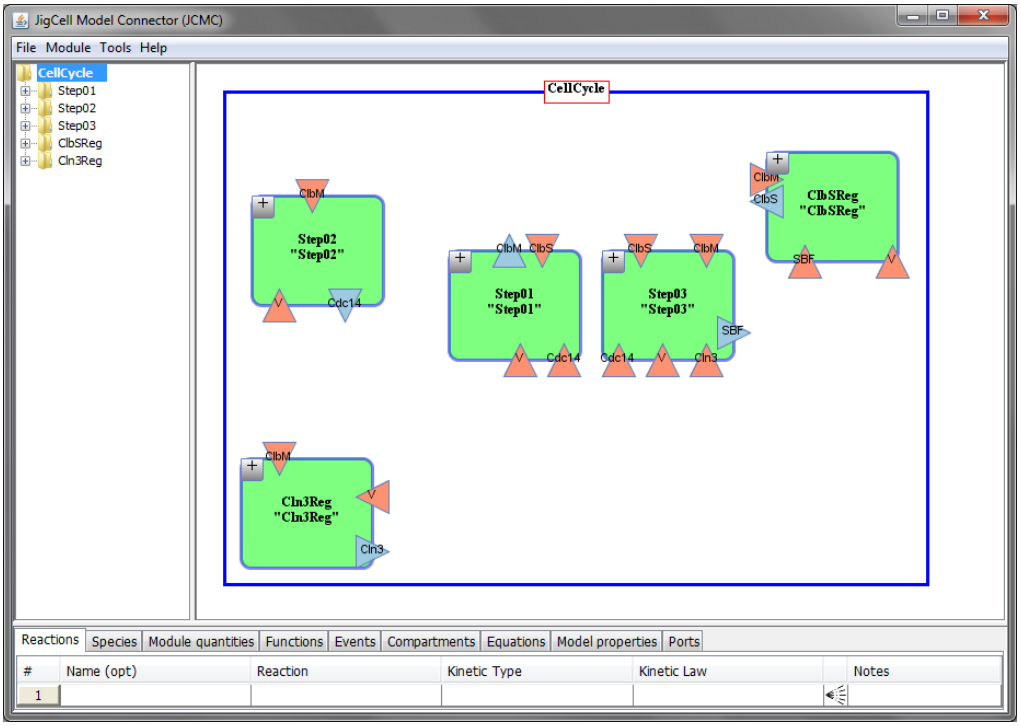


Figure 5.31: Initial CellCycle model in JCMC

Figure 5.32 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 *Cdc*14 connects the species to three submodules as well. Node *Cdc*14 receives a value from submodule Step02, where it is regulated and part of the *RENT* complex. Node *Cdc*14 sends its value to submodules Step01 and Step03, where it promotes dephosphorylation. Visible variable node *Cln*3 connects the species to two submodules. Node *Cln*3 receives a value from submodule Cln3Reg, where it is regulated. Node *Cln*3 sends its value to submodule 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 *ClbS* transcription. Visible variables nodes *ClbM* 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

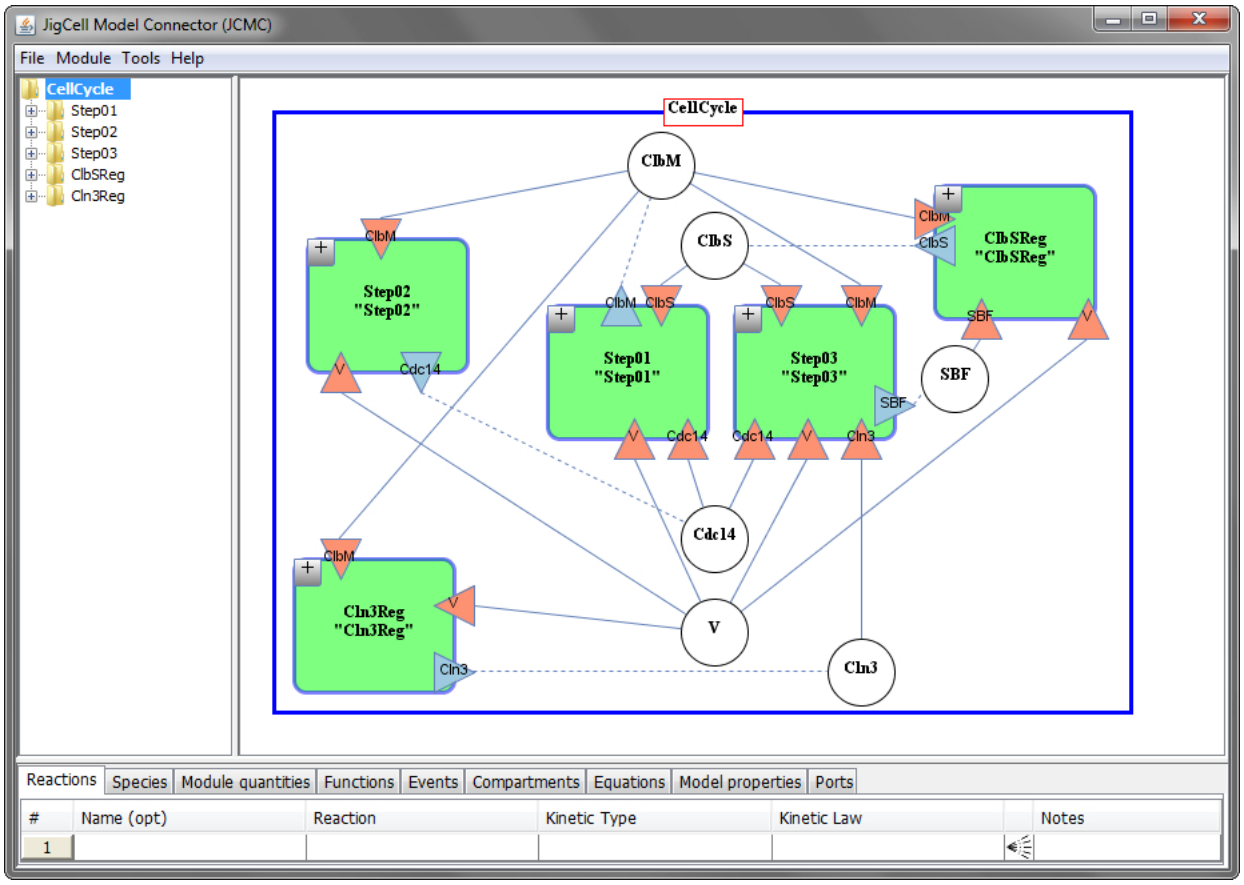
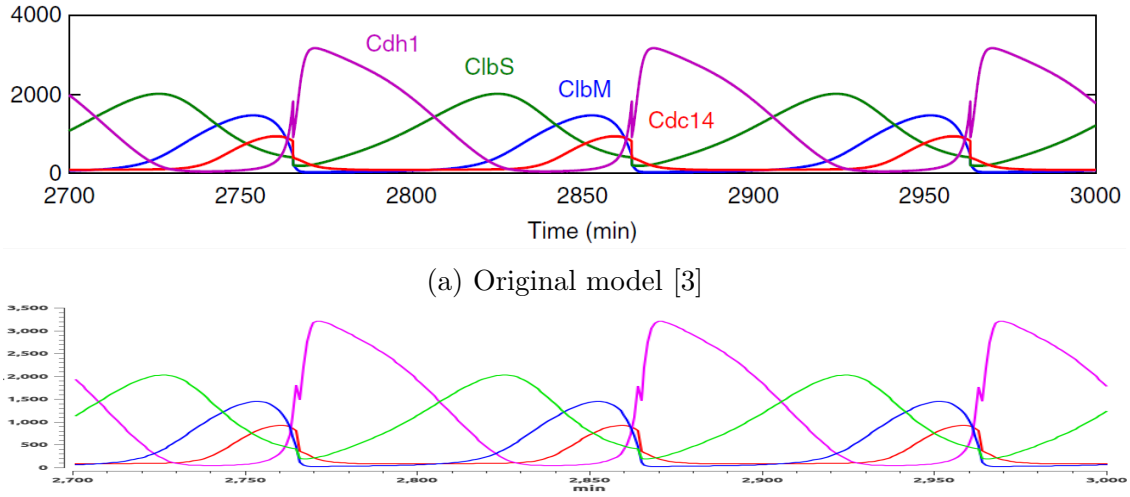


Figure 5.32: Final CellCycle model in JCMC

cell division occurs.

5.2.7 Simulation Results

To validate the hierarchical model, we compared its simulation results with results from the original model. The hierarchical model cannot be simulated in its current format because the interaction details of the modules must be used as instructions to flatten the model, as described in Chapter 2.



(a) Original model [3]

(b) Flattened hierarchical model

Figure 5.33: Time course simulation results

Figure 5.33 displays the two model simulations. We exported the hierarchical model into SBML format with the *comp* package using JCMC. We used the software tool COPASI [7] 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 in the paper by Barik et al.[3]. The simulation results from the flattened hierarchical model in Figure 5.33b match the results from the original model in Figure 5.33a. This was verified by confirming the time series simulation data from both models were identical.

Chapter 6

Conclusions and Future Work

Attempting to comprehend at once all the interactions in a complex molecular control network can be an overwhelming task. When building a new mathematical model, try to split the wiring diagram into small, manageable pieces. We did this in the case study in Chapter 5. We broke down the model displayed in Figure 5.1 into smaller sections shown in Figures 5.4, 5.8, and 5.17. Working on smaller parts of the model can make it easier to understand. Once the smaller modules are complete, they can be used to build up the final model.

When building a mathematical model, utilizing templates can also be beneficial. 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 5.3. 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 twelve 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 [20]. 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 model. As modules are created, their inner reaction structures can be verified using simulation tools such as COPASI [7]. 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 thesis, we have reviewed the improved modeling approaches involved with hierarchical model composition. We have reviewed software standards that support this modeling approach. We presented previous software tools that attempt to support hierarchical modeling.

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.

We implemented a new tool that supports hierarchical modeling, the JigCell Model Connector (JCMC). We described the JCMC interface and explained how the components of a hierarchical model are represented. We detailed how the new port and node constructs 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 JigCell Model Connector would be to incorporate multistate modeling. Currently, 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 JigCell Model Connector 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.

Bibliography

[1] N. A. Allen, L. Calzone, K. C. Chen, A. Ciliberto, N. Ramakrishnan, C. A. Shaffer, J. C. Sible, J. J. Tyson, M. T. Vass, L. T. Watson, and J. W. Zwolak. Modeling regulatory networks at Virginia Tech. *OMICS : A Journal of Integrative Biology*, 7(3):285–299, 2003.

[2] N. A. Allen, C. A. Shaffer, N. Ramakrishnan, M. T. Vass, and L. T. Watson. Improving the development process for eukaryotic cell cycle models with a modeling support environment. *Simulation*, 79(12):674–688, 2003.

[3] D. Barik, W. T. Baumann, M. R. Paul, B. Novák, and J. J. Tyson. A model of yeast cell-cycle regulation based on multisite phosphorylation. *Molecular Systems Biology*, 6(405):405, 2010.

[4] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[5] C. P. Fall and J. E. Keizer. Dynamic Phenomena in Cells. In *Computational Cell Biology*, chapter 1, pages 1–20. Springer-Verlag, New York, 2002.

[6] R. Gauges, U. Rost, S. Sahle, K. Wengler, and F. T. Bergmann. Layout, Version 1 Release 1. Available from COMBINE http://identifiers.org/combine.specifications/sbml.level-3.version-1.layout.version-1.release-1, 2013.

[7] S. Hoops, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI - A COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[8] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology

markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[9] M. Hucka, M. Hucka, F. Bergmann, S. Hoops, S. Keating, S. Sahle, J. Schaff, L. Smith, and D. Wilkinson. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core. *Nature Precedings*, Oct 2010.

[10] 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. *Bioinformatics*, 25(21):2848–2849, Nov 2009.

[11] A. Palmisano, S. Hoops, L. T. Watson, T. C. Jones Jr., J. J. Tyson, and C. A. Shaffer. Multistate Model Builder (MSMB): a flexible editor for compact biochemical models. *BMC Systems Biology*, 8(1):42, 2014.

[12] R. Randhawa, C. A. Shaffer, and J. J. Tyson. Fusing and composing macromolecular regulatory network models. In *Proceedings of the 2007 High Performance Computing Symposium*, volume 1, pages 337–344, 2007.

[13] R. Randhawa, C. A. Shaffer, and J. J. Tyson. Model aggregation: A building-block approach to creating large macromolecular regulatory networks. *Bioinformatics*, 25(24):3289–3295, Dec 2009.

[14] R. Randhawa, C. A. Shaffer, and J. J. Tyson. Model composition for macromolecular regulatory networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(2):278–287, 2010.

[15] C. A. Shaffer, R. Randhawa, and J. J. Tyson. The role of composition and aggregation in modeling macromolecular regulatory networks. In *Proceedings - Winter Simulation Conference*, pages 1628–1635, Monterey, CA, 2006.

[16] C. A. Shaffer, J. W. Zwolak, R. Randhawa, and J. J. Tyson. Modeling molecular regulatory networks with JigCell and PET. In *Methods in Molecular Biology*, volume 500, pages 81–111. 2009.

[17] J. C. Sible and J. J. Tyson. Mathematical modeling as a tool for investigating cell cycle control networks. *Methods*, 41(2):238–47, Feb 2007.

[18] L. P. Smith, F. T. Bergmann, D. Chandran, and H. M. Sauro. Antimony: A modular model definition language. *Bioinformatics*, 25(18):2452–2454, Sep 2009.

[19] L. P. Smith, M. Hucka, S. Hoops, A. Finney, M. Ginkel, C. J. Myers, I. Moraru, and W. Liebermeister. Hierarchical Model Composition, Version 1 Release 3. Available from COMBINE http://identifiers.org/combine.specifications/sbml.level-3.version-1.comp.version-1.release-3, 2013.

[20] I. Sommerville. *Software engineering*, volume 9th Ed. Pearson, 2011.

[21] J. J. Tyson. Bringing cartoons to life. *Nature*, 445(7130):823, Feb 2007.

[22] J. J. Tyson, K. C. Chen, and B. Novák. Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell. *Current Opinion in Cell Biology*, 15(2):221–231, Apr 2003.

[23] J. J. Tyson and B. Novák. Models in biology: lessons from modeling regulation of the eukaryotic cell cycle. *BMC Biology*, 13(1):46, Dec 2015.

[24] M. Vass, N. Allen, C. A. Shaffer, N. Ramakrishnan, L. T. Watson, and J. J. Tyson. The JigCell model builder and run manager. *Bioinformatics*, 20(18):3680–3681, Dec 2004.

[25] M. T. Vass, C. A. Shaffer, N. Ramakrishnan, L. T. Watson, and J. J. Tyson. The JigCell Model Builder: A spreadsheet interface for creating biochemical reaction network models. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(2):155–163, 2006.