Model Composition and Aggregation in Macromolecular Regulatory Networks

Ranjit Randhawa

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

Doctor of Philosophy
in
Computer Science and Applications

Clifford A. Shaffer, Chairman
John J. Tyson
Narendran Ramakrishnan
Osman Balci
Jill C. Sible

April 4, 2008
Blacksburg, Virginia

Keywords: Systems Biology, Model Composition and Aggregation, Modeling & Simulation, Verification & Validation, Macromolecular Regulatory Networks, JigCell.

Model Composition and Aggregation in Macromolecular Regulatory Networks

Ranjit Randhawa

Abstract

Mathematical models of regulatory networks become more difficult to construct and understand as they grow in size and complexity. Large regulatory network models can be built up from smaller models, representing subsets of reactions within the larger network. This dissertation focuses on novel model construction techniques that extend the ability of biological modelers to construct larger models by supplying them with tools for decomposing models and using the resulting components to construct larger models.

Over the last 20 years, molecular biologists have amassed a great deal of information about the genes and proteins that carry out fundamental biological processes within living cells — processes such as growth and reproduction, movement, signal reception and response, and programmed cell death. The full complexity of these macromolecular regulatory networks is too great to tackle mathematically at the present time. Nonetheless, modelers have had success building dynamical models of restricted parts of the network. Systems biologists need tools now to support composing "submodels" into more comprehensive models of integrated regulatory networks.

We have identified and developed four novel processes (fusion, composition, flattening, and aggregation) whose purpose is to support the construction of larger models. Model Fusion combines two or more models in an irreversible manner. In fusion, the identities of the original (sub)models are lost. Beyond some size, fused models will become too complex to grasp and manage as single entities. In this case, it may be more useful to represent large models as compositions of distinct components. In Model Composition one thinks of models not as monolithic entities but rather as collections of smaller components (submodels) joined together. A composed model is built from two or more submodels by describing their redundancies and interactions.

While it is appealing in the short term to build larger models from pre-existing models, each developed independently for their own purposes, we believe that ultimately it will become necessary to build large models from components that have been designed for the purpose of combining them. We define Model Aggregation as a restricted form of composition that represents a collection of model elements as a single entity (a "module"). A module contains a definition of pre-determined input and output ports. The process of aggregation (connecting modules via their interface ports) allows modelers to create larger models in a controlled manner.

Model Flattening converts a composed or aggregated model with some hierarchy or connections to one without such connections. The relationships used to describe the interactions among the submodels are lost, as the composed or aggregated model is converted into a single large (flat) model. Flattening allows us to use existing simulation tools, which have no support for composition or aggregation.

Para Mónica

Acknowledgements

I would like to give a special thanks to my advisors, Dr. Clifford A. Shaffer and Dr. John J. Tyson for accepting me into their research group and their support, advice, and endless patience. During my time here, both Dr. Shaffer and Dr. Tyson have helped and encouraged me in my research and for this I will be forever grateful. Dr Tyson's lab has been a positive experience and one that I wont soon forget. I am also very thankful to my lab mates: Dr. Bill Baumann, Dr Yang Cao, Dr Kathy Chen, Dr Jason Zwolak, Dr Nick Allen, Dr Suman Banik, Tongli Zhang, Shengua Li, Pengyuan Wang, and Janani Ravi for their help, assistance and guidance in my research. And the rest of the lab: Paul, Oak, Rajat, Sandip, Debashis and Baris, for making the lab a fun place to work in.

I would like to thank my committee Dr. Naren Ramakrishnan, Dr Jill Sible and Dr. Osman Balci for being a valuable source of ideas and support during my Ph.D. program. They have always been accessible and willing to help me with advice and guidance. A big special thanks also goes out to the folks I have collaborated with over the last 4 years, Mike Hucka, Andrew Finney, Herbert Suaro, Stefan Hoops, Colin Gillespie, and the rest of the SBML gang for their help, insights, feedback and ideas. I will certainly miss the lively SBML meetings I have had the pleasure of attending all over the world.

To my family and friends, thank you for always being there whenever called upon. To Mom, Pa and Tara, I love you guys and thank you for your constant unconditional love, support during this journey, and for helping me become the man I am today. To Pa, I miss you and thank you for "my unconquerable soul", this one's for you. To Eileen and Julio and the rest of the Caparra Crew, thank you for adopting me into your family, for your love and support, and treating me like one of your own. To Raghu, Anurag, Pardha, and Sam to name a few for making the really long days in grad school pass quicker and making my Blacksburg chapter a memorable one. To Lola my labrador retriever, for being "Lola Number One", the best dog in the world and little Valetina for coming into our lives and converting us from dog people to dog *and* CH cat people. Finally, to my beautiful wife Mónica, *yo te amo mucho niña*. I would not have been able to do this without your love, laughter, understanding, and support, so from the bottom of my heart thank you for always being there for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The physiological properties of cells are governed by macromolecular regulatory networks of great complexity [31]. Understanding the dynamical properties of these networks is facilitated by mathematical modeling of the biochemical reactions [31, 43, 67, 72]. These models are often implemented deterministically, as sets of nonlinear differential equations, or probabilistically by Gillespie's stochastic simulation algorithm. In either case, the modeler is faced with the problem of specifying the reactions involved in a large complex network of interacting species, the rate laws describing each reaction, and numerical values for the rate constants involved in each rate law. Building regulatory network models is a little like putting together a complicated jigsaw puzzle with many interlocking pieces. This complex modeling challenge is best broken down into smaller components that can later be joined together into a larger whole.

The main research objective for the work described in this dissertation is to extend the ability of modelers to construct large models by supplying them with tools for decomposing models and using the resulting components to construct larger models. This will be accomplished by identifying novel modeling processes (called model fusion, composition, flattening, and aggregation), identifying language extensions for Systems Biology Markup Language (SBML) [25], and implementing the proposed language changes into the SBML language to support the construction of models of increasing complexity and size. We shall first construct large models from existing toy models, then move up to realistic models of biological systems, such as the Tyson and Novak 2001 cell cycle model [74], and finally trying to combine large-scale biological models of increasing complexity such as the morphogenesis checkpoint model [12] and the budding yeast cell cycle model [11].

SBML was created to support the modeling of biochemical reaction networks, but the present version (Level 2) does not support any notion of model composition or aggregation. The language extensions proposed in this dissertation were ratified at meetings about model composition; the Third SBML Hackathon (2005) in Tokyo, Japan; the Fifth Annual SBML Hackathon (2007) in Newcastle, UK; the SBML Composition Workshop (2007) at the University of Connecticut Health Center; the Sixth SBML Hackathon (2008) in Stellenbosch, South Africa; and private communications with Mike Hucka, Andrew Finney, Martin Ginkel,

and Stefan Hoops among others. We will provide a concrete proposal for composition for SBML Level3. Current SBML parsers must also be extended to deal with model composition and aggregation. The open source JigCell SBML Parser Library was developed in-house at Virginia Tech and has been extended to enable model composition and aggregation.

Building accurate biological models is a painstaking task of collecting experimental data from published papers and fitting the model to the experimental data. In this approach, there is a correlation between the size of a model and the amount of biological information it represents. Therefore, the ability to construct large biological models provides the potential for better insights into the workings of a cell under investigation. Models that exist today are small compared to the amount of information known about a particular organism or cellular pathway/process. Previous modeling work focused on building small models that dealt with a restricted view of the overall physiology of a cell. Modelers work on individual pieces (cellular processes or certain pathways) that are easy to construct and manage. Their ultimate goal is to put these pieces together to construct a more complete picture of the underlying molecular machinery of the organism under investigation. Merging the pieces together will provide researchers with more complete and biologically accurate models with which to perform simulations. Currently this merging step is an error-prone process as it is done manually. The level of complexity is difficult to deal with as the number of models and their sizes increase. When making large models it is better to start from existing models in order to reuse information from smaller models, rather than from scratch. This is analogous to adding features to an existing computer program rather than having to completely re-write it to incorporate new functionality.

Modeling tools help modelers construct their models by providing a computational environment or framework that minimizes the amount of human error during the construction step. While modelers are currently able to construct large models by hand, the process is simplified by using computational tools which not only decrease the time taken to input a model but also ensure that the modeler does not make mistakes while inputting the model.

1.1 Overview

Chapter 2 introduces the building blocks from which biological modelers construct their models. The models described in this dissertation are in terms of chemical reactions and chemical reaction kinetics. Chemical reactions describe the structure of the biological processes that take place within a cell. Chemical reaction kinetics describe the rate at which the chemical reactions occur. The corresponding primitive building block for mathematical models is the differential equation. A differential equation defines a family of functions in terms of their rate of change over time.

Chapters 3, 4, and 5 apply the new modeling processes to develop tools that biological modelers can use. The Fusion/Composition Wizard, the Flattening Algorithm, and the Aggregation Connector are additions to the JigCell modeling environment, which is a suite of applications, programming libraries, and utility programs that focuses on the production, execution, and analysis of models of biochemical reaction

networks. The modeling tools developed are open source and employ documented standards for interoperable communication with other tools and applications. Efforts are underway to prepare and present an abridged version of this dissertation as a formal proposal in order to include these language features into the next version of SBML.

Chapter 6 presents a case study applying the processes, tools, and frameworks presented in this dissertation to combine together a biological model for cell cycle control and a biological model morphogenesis checkpoint in budding yeast.

Chapter 7 contains the conclusions, summarizes the contributions made, and presents future avenues for research to extend and expand the ideas in this dissertation. This chapter also discusses the ongoing interaction with the SBML community to propose and implement support for composition within the language.

Connections to other works

Much of the material in this dissertation previously appeared in part in publications [56–58, 66] or online [27]. In most cases, these publications contain abridged versions of the material presented here. This dissertation supersedes the earlier publications by providing a more thorough treatment of the material, up-to-date accounts of the software, methods, and results, and corrections to the errors found subsequent to publication.

1.2 Modeling and Simulation

A model is a theoretical construct that reproduces the set of variables, behaviors and properties of a natural system. There are many different types and forms of models, such as physical, logical, or mathematical models. However, our focus is on mathematical models of biological systems. Modelers attempt to capture the properties of their biological system with mathematical equations. After performing experiments on the model, a modeler then relates the results of the model experiment to an equivalent experiment in the biological system.

Simulation is a method for studying the evolution of model behaviors over time. Simulation does not have to be exact and when dealing with mathematical models of biological systems, some level of abstraction exists. Given existing tools, standards, and resources, it is not likely that modelers could replicate the complex behaviors of a cell without error or abstraction.

Biologists employ modeling and simulation to test their hypotheses about a biochemical reaction network or develop hypotheses they test by experimental validation. The cost of building a mathematical model of the biological system and performing experiments on that model is usually less than the cost (both monetary and time) of laboratory experimentation. It is crucial that models are credible and reliable, and that biologists understand the limitations of their modeling and simulation efforts. Developers, modelers, decision makers, and those impacted by the outcome of the model are all concerned with whether the model is correct [61].

Model verification, validation, and testing activities assess the accuracy of a model. Performing these activities also improves the credibility and reliability of a model. The practice of model verification, validation,

and testing is essential to the consistent production of models that are useful and correct. Comprehensive overviews of model verification, validation, and testing already exist and are provided for reference [4, 61].

Model verification is the process of certifying that transformations of the model from one form to another maintain the fidelity of a model. Model verification ensures that the modeler transforms the model as they intended and that the modeler preserves the accuracy of the model over time. Modeling tools that support model verification allow the modeler to check that the tool operates in the manner that the modeler assumes. Model verification checks that the process of building the model is correct. We try to address verification, both in terms of the models generated and the processes used.

Model validation is the process of determining whether a model sufficiently approximates the real system. The definition of the term "sufficiently" depends upon the purpose of the model. Increasing the validity of a model has cost, it is therefore most efficient to evaluate the model with respect to its intended application [6]. The purpose of the model dictates the important aspects to validate and the standards that should apply. Different modelers may have different intended purposes for a model. Therefore, it is possible for a model to pass the criteria for acceptability for one modeler but not for another modeler.

Model testing is the process of checking for errors in the model. Model testing determines if the model is functioning properly by subjecting the model to controlled inputs. The modeler designs a model test to perform model verification, model validation, or both activities. The model experiments that the modeler uses for testing determine the domain of acceptability for the model. Although one form of model experimentation is to compare the model with historically-collected or newly designed laboratory experiments, the modeler can use other applicable model verification and validation techniques. The intended application of the modeler determines the acceptable range of results from the model tests. Preferably, the modeler fixes the acceptable range before the development and testing of the model, and then works to make the model acceptable. One part of model accreditation, the certification that the model is acceptable for a particular application, comes from the documentation of model verification and validation that the modeler generates from these activities [61].

Chapter 2

Modeling Regulatory Networks

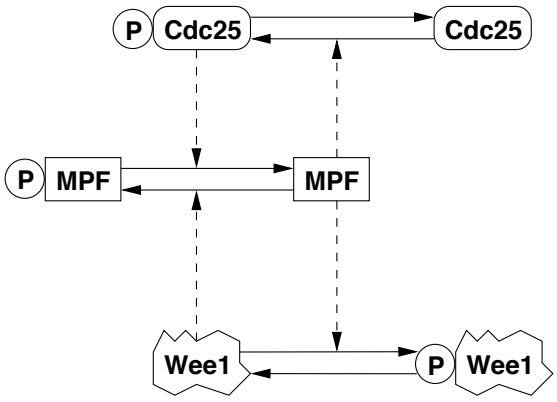
The ultimate goal of computational molecular biology is to understand the physiology of a cell in terms of the underlying molecular machinery. While experimental observation can provide snapshots of a cell's physiology under defined sets of conditions, it does not provide the complete picture. Computational molecular biology aims to take these snapshots and create a better understanding of the temporal dynamics of a cell. The focus is on the networks of interacting proteins within a cell that govern its physiology. The aim is to derive the physiological properties of a cell from the wiring diagrams of its underlying molecular regulatory circuits [74].

This chapter provides a background for this dissertation, by explaining models of regulatory networks using simple examples and how they are constructed, tested, simulated, and validated. Section 2.1 describes the process of modeling with a basic example of the frog egg model. Section 2.2 describes the overall modeling process. Section 2.3 describes the need for extending current modeling frameworks in order to build larger models than previously possible. Finally, Section 2.4 describes previous efforts and related work that motivated our approaches.

Contents

2.1 Regulatory Network Modeling

Mathematical models of gene-protein regulatory networks play key roles in archiving and advancing our understanding of the molecular basis of cell physiology. Models provide rigorous connections between the



Species	Description	Phosphorylated
M_a	Active MPF	no
M_i	Inactive MPF	yes
C_a	Active Cdc25	yes
C_i	Inactive Cdc25	no
W_a	Active Wee1	no
W_i	Inactive Wee1	yes

$$\frac{dM_a}{dt} = (v'_c \cdot C_i + v''_c \cdot C_a) \cdot M_i - (v'_w \cdot W_i + v''_w \cdot W_a) \cdot M_a$$
$$\frac{dC_a}{dt} = \frac{v_c \cdot M_a \cdot C_i}{K_{mc} + C_i} - \frac{v'''_c \cdot v_c \cdot C_a}{K_{mcr} + C_a}$$
$$\frac{dW_a}{dt} = -\frac{v_w \cdot M_a \cdot W_a}{K_{mw} + W_a} + \frac{v'''_w \cdot v_w \cdot W_i}{K_{mwr} + W_i}$$

Figure 2.1: Network diagram, mapping of species names, and the corresponding set of ordinary differential equations for a model of the mitotic regulatory system in frog eggs. The regulation of MPF (Mitosis Promoting Factor) by Wee1 (kinase) and Cdc25 (phosphatase) controls when the cell enters mitosis. Notice the two positive feedback loops whereby MPF actives Cdc25 (MPF's activator) and inactivates Wee1 (MPF's inactivator). The active forms (M_a, C_a, and W_a) have associated differential equations. The total amounts of MPF (M_T), Wee1 (W_T) and Cdc25 (C_T) are conserved (i.e., remain constant throughout the process). $M_i + M_a = M_T$, $W_i + W_a = W_T$, and $C_i + C_a = C_T$. Therefore, the inactive forms (M_i, C_i, and W_i) do not have differential equations because they can be calculated from these conservation relationships.

physiological properties of a cell and the molecular wiring diagrams of its control systems. A simple example is the set of reactions controlling the activity of MPF (mitosis promoting factor) in *Xenopus* oocytes [38], which we refer to herein as the frog egg model. Such networks are often represented as graphs. In the diagram of this network (Figure 2.1), vertices represent substrates and products (collectively referred to as species), solid directed edges represent biochemical reactions, and dashed directed edges represent regulatory signals.

Collectively, these biochemical reactions cause the concentrations of the chemical species (S_i) to change

in time according to a set of differential equations (one for each species)

$$\frac{dS_i}{dt} = \sum_{j=1}^{R} b_{ij} v_j, i = 1, \ldots, N,$$

where R is the number of reactions, N is the number of species, v_j is the velocity of the j^{th} reaction in the network, and b_{ij} is the stoichiometric coefficient of species i in reaction j ($b_{ij} < 0$ for substrates, $b_{ij} > 0$ for products, $b_{ij} = 0$ if species i takes no part in reaction j). Figure 2.1 shows differential equations derived from the reactions in the network diagram. The set of rate equations and associated parameter values is a mathematical representation of the temporal behavior of the regulatory network.

Since the purpose of these models is to codify a systems-level understanding of the control of some aspect of cell physiology, it is necessary to validate a proposed model against observed behavior of the reference system. In most cases it is essential to model the behavior of not only the wild-type form of the organism, but also of many mutant or manipulated forms (where each mutant form typically represents one or two variations in the genetic specification of the control system). For example, if we are modeling the cell cycle of an organism, then we would wish to know features such as the cell size at division, the time required for various phases of the cell cycle (G1, S, G2, M), and the viability or point of failure for each mutation. Measurements of the amounts for various control species within the cell over time would also be valuable information. In the case of a thoroughly studied an d genetically tractable organism such as *Saccharomyces cerevisiae* (budding yeast), a model can be compared against many dozens of mutants defective in the regulatory network.

A realistic model of the budding yeast cell cycle consists of over 30 differential equations and 100 rate constants and is tested against the phenotypes of over 150 mutants [11]. A model of this complexity represents the upper limit of what a dedicated modeler can produce "by hand" with nothing but a good numerical integrator like LSODE [24]. Beyond this size, we begin to lose our ability even to meaningfully display the wiring diagram that represents the model, let alone comprehend the information it contains, or parameter estimation becomes problematic. To adequately describe fundamental physiological processes (such as the control of cell division) in mammalian cells will require models of 100-1000 equations. Efforts such as the DARPA BioSPICE initiative [7] and the DOE Genomes to Life project [46] aspire to support models at least one order of magnitude larger than are currently used. To handle this next generation of dynamical models will require sophisticated software to automate the modeling process: network specification, equation generation, simulation and data management, and parameter estimation.

There are a number of distinct approaches to simulation. Deterministic models usually represent the system of chemical reactions with ordinary differential equations [2,39,62]. In some cases, partial differential equations are used to account for spatial effects [63]. Stochastic modeling or biochemical signaling networks is in its infancy, and most often is done by some variation of Gillespie's algorithm [10, 16, 36]. For the remainder of this dissertation, we will consider only deterministic simulation by ODEs. Creating a model that mimics the observed behavior of a living organism is a difficult task. This process involves a combination of biological insight, persistence, and support by good modeling tools.

2.2 The Modeling Process

Successful modeling of macromolecular regulatory networks can be aided by software tools based on a well-defined modeling process. Such tools should support the line of thought followed by modelers as they approach a problem. Mid-sized models of macromolecular regulatory networks track reactions among tens of species and are tested against hundreds of experimental observations. Thus, modelers need tools that help to organize the relevant information and automate as many steps of the process as possible. Figure 2.2 shows our conception of the modeling process. The modeler starts with an idea about an organism and a regulatory system to model. Next, the modeler gathers information (from the literature and from their own experiments) related to the regulatory system of the organism. During the literature search the modeler builds a hypothesis from information already published, continuously checking the hypothesis against the existing literature. Once the modeler has a testable hypothesis about the regulatory system, the hypothesis can be codified into four types of technical information:

- Experimental Data: The information that will be used to validate the model. This information might come as time series data of the concentrations of certain regulatory chemical species, as other observables such as the average size of cells at division, or as qualitative properties such as the viability or inviability of a mutant.
- Simulation Runs: Specifications for the simulations that will be made to model the experimental data. For example, each simulation might relate to a specific mutation of the organism. The specification will define the distinct conditions necessary to simulate that mutation, such as differences in rate constants from the wild-type values.
- Reaction network: The chemical equations that describe the regulatory processes.
- Rate constants: The parameters that govern the reaction rates.

Typically, the experimental data and simulation run descriptions are part of the problem definition and are not subject to frequent modifications. Nor are they considered to be "right" or "wrong" in the same way as the reaction network and rate constant values typically will be. The network and rate constants together define the mathematical model that will be simulated, compared to experimental observations, and judged "acceptable" or "unacceptable."

One simulation run of an ODE model takes only a fraction of a second on a typical desktop computer in 2007. As described above, a complete model actually involves a large collection of simulations, to be compared against a collection of experimental results. This entire set of runs might take a second or so for a smaller model such as our frog egg example on a desktop computer for one choice of rate constants, and about a minute or two for a larger model needed to describe the budding yeast cell cycle.

Once an initial specification of these four types of information has been made, the next phase of the process begins. This is a simulation-compare-update loop, whereby simulation results are compared to the experimental data. In some way, either a human or a computer will make a judgement as to the quality of the relationship between the two. At that point, since the model is typically judged unsatisfactory, the modeler

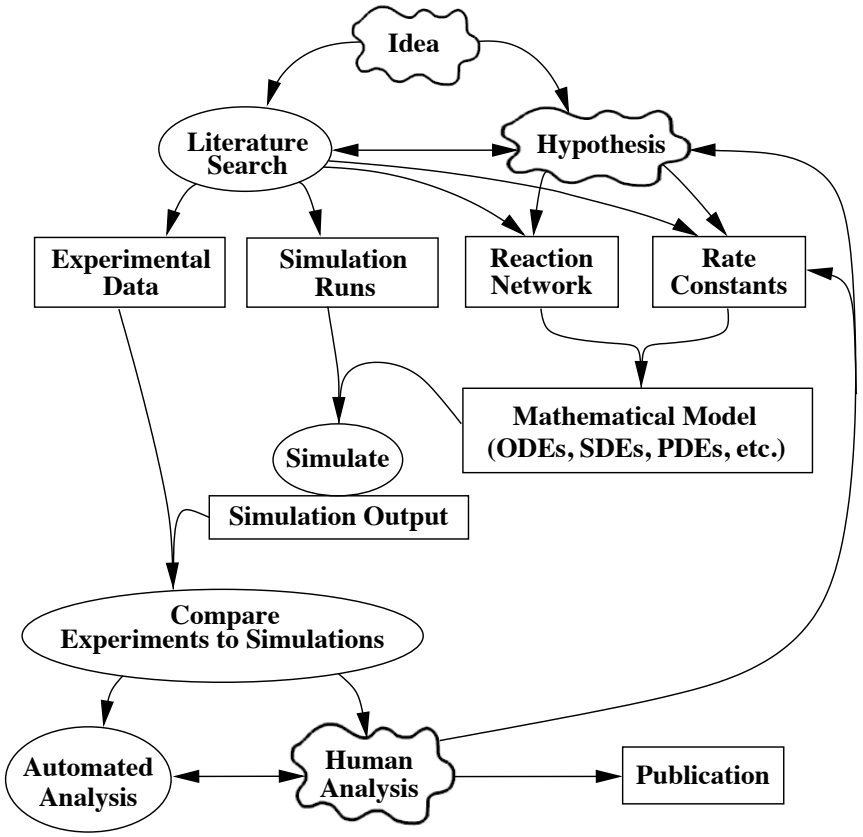


Figure 2.2: The modeling process. Once the modeler has generated a testable hypothesis about the organism, he or she must assemble the four necessary collections of information (experimental data, simulation runs, reaction network, and rate constants). This defines both the mathematical model and the behavior that the model must reproduce. The modeler then will repeatedly simulate and update the model, perhaps with the aid of automated analysis tools, until an acceptable result is obtained.

will make adjustments and repeat the cycle. We prefer to view this as a double loop, in that changes to rate constant values are made much more frequently than changes to the reaction network. That is, the modeler will typically "twiddle" the rate constants so long as progress is being made in matching simulation output to experimental data. When changes to the rate constants appear no longer to improve the match, then the modeler will attempt to improve the model by changing the reaction network, which in turn will trigger another round of changes to the rate constants. The process is continued until the model is judged satisfactory or totally hopeless.

Modelers often try to assign values to rate constants by a time-consuming process of "parameter twiddling" and visual comparison of simulation results to experimental data. A better approach is automated parameter estimation (once the modeler is confident that the basic structure of the reaction network is sound enough). To fit a model to experimental data by automated optimization algorithms requires thousands to millions of repetitions of the full calculations.

The process of comparing real-world observation (experimental data) with the mathematical model (time-series output from a simulation) is called model validation. Model validation is closely related to automated (or manual) parameter estimation, because both require that some measure of the quality of the model can be made. In the case of automated parameter estimation, we need a way to take the experimental data and the output from a simulation run, and create a single number as a measure of the quality of the fit.

This can be extremely difficult. The simulation data (usually in the form of time series plots) often appears in a form different from the experimental data (often qualitative information such as whether a cell is viable or not). In general, some complex computation must be done to relate the two. The function that does this computation is called a *transform*. Then, while it might be a simple judgement to measure the goodness of fit between one simulation and one experiment, it is often difficult to judge the goodness of fit of an entire ensemble of runs. Improvements in matching some experiments might come at the cost of worse fits for others, so their relative merits must be weighted [49].

2.3 Building Larger Models

Over the last 20 years, molecular biologists have amassed a great deal of information about the genes and proteins that carry out fundamental biological processes within living cells — processes such as growth and reproduction, movement, signal reception and response, and programmed cell death. The full complexity of these macromolecular regulatory networks is too great to tackle mathematically at the present time. Nonetheless, modelers have had success building dynamical models of restricted parts of the network. For example, for budding yeast cells there have been recent successful efforts to model the cell cycle [11], the pheromone signaling pathway [30], the response to osmotic shock [29], and the morphogenesis checkpoint [12]. Systems biologists need tools now to support composing "submodels" (like these) into more comprehensive models of integrated regulatory networks. Another motivation for our desire to create and implement concrete frameworks for composition are making a model suitable for stochastic simulation which increases the number of reactions by a factor of 3-5 [76].

We assume that each of the submodels used in creating the larger model can be a validated model itself, with experimental data that fixes its parameters. The main motivation for creating a larger model is that there exists experimental information on the interaction of the subsystems for which submodels cannot account for. By aggregating validated submodels, we mitigate the problem of searching through large parameter spaces. The parameter estimation problem is now to ensure that the aggregated model is consistent with the original data used to validate the submodels (for which we already have good initial guesses, inherited from the submodels) and also the new data relevant to the interactions of the subsystems (which are governed by the new parameters describing how the submodels fit together).

Our prior work has identified several distinct processes (called model fusion, composition, flattening, and aggregation) whose purpose is to support the construction of larger models [56–58, 66]. Figure 2.3 shows the relationship between these four processes. In the figure, both composition and aggregation are

labelled as reversible processes as the integrity of the submodels is maintained and can be derived from the resulting composed or aggregated model. This is not the case in fusion, where submodel identity and the model developmental history of how the larger model is created is lost. Flattening converts composed and aggregated models into their flattened forms which are essentially equivalent to the fused form of the model. The fused (or flattened) model can now be simulated in order to generate results.

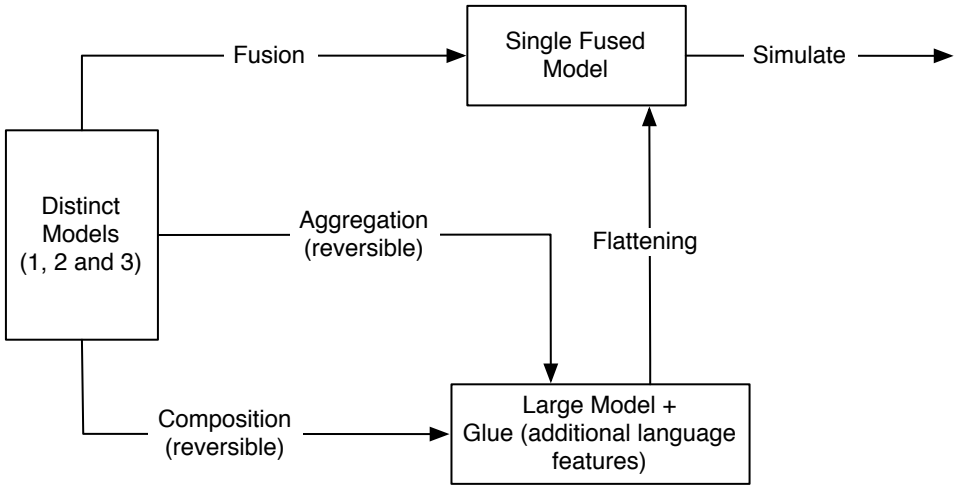


Figure 2.3: Relationship between Fusion, Composition, Aggregation and Flattening.

Model Fusion (Chapter 3) is an iterative process to combine two or more models in an irreversible manner. The goal of fusion is to combine submodels into a single unified model containing the combined information (without redundancies) across the original collection. In fusion, the identities of the original (sub)models being combined are lost. The result of fusion is a model in the same language as the submodels (in our case standard SBML [25]), meaning that the same simulation analysis tools can be applied. Beyond some size, fused models will become too complex to grasp and manage as single entities. In this case, it may be more useful to represent large models as compositions of distinct components. Thus, while model fusion is a useful tool for manipulating small to mid-sized models, it does not seem to be a viable solution in the long run.

Model Composition (Chapter 4) provides a potential solution to our goal to build models of large reaction networks. With composition, one thinks of models not as monolithic entities but rather as collections of smaller components (submodels) joined together. A composed model is built from two or more submodels by describing their redundancies and interactions. Composition is a reversible process, in that removing the inter-model interaction description that holds the composed model together recovers the individual submodels.

While it is appealing in the short term to build larger models from pre-existing models, each developed independently for their own purposes, we believe that ultimately it will become necessary to build large models from components that have been designed for the purpose of combining them. We distinguish this approach from model composition as described in [57, 66]. We define *Model Aggregation* [58] (Chapter 5) as a restricted form of composition that represents a collection of model elements as a single entity (a "module").

A module contains a definition of pre-determined input and output ports. We distinguish between the terms module and submodel by defining a module to be a submodel with ports. These ports link to internal species and parameters and enable them to be accessed/referenced outside the model in which they occur. They define the module's interface, which provides restricted access to the components within the module. The process of aggregation (connecting modules via their interface ports) allows modelers to create larger models in a controlled manner.

Model Flattening (Section 4.5) converts a composed or aggregated model with some hierarchy or connections (discussed later) to one without such connections. The result of flattening is equivalent to fusing the submodels. The relationship information provided by the composition or aggregation process must be sufficient to allow the flattening to take place without any further human intervention. The relationships used to describe the interactions among the submodels are lost, as the composed or aggregated model is converted into a single large (flat) model. Flattening a model allows us to use existing simulation tools, which have no support for composition or aggregation.

In order to better understand the relationships between the four processes described above, consider the timeline we followed to develop these processes and tools. Fusion was originally conceived as modelers in John Tyson's Group at Virginia Tech desired some computational assistance in creating and combining large models of macromolecular regulatory networks. Our first step in solving the model composition problem was to provide a fusion tool to enable modelers to quickly, easily, and efficiently combine models together. as this was previously done manually and thus was an error-prone process. The only requirement that we adhered to while developing the fusion process and tool was to use existing standards, such as SBML Level2, to ensure that it would be viable. After developing the fusion tool we realized that it was not an ultimate solution. Fusion turned out to be a somewhat temporary measure as ultimately even fused models would become too large and complex to manage and maintain. Developing fusion first, led us naturally to model composition, which we learnt was very similar to the fusion process, both in terms of the information needed in developing a composed model and the step-by-step process used to facilitate this information from the modeler. Once both fusion and composition were completed, we decided to look at the fundamental problem of model composition differently which led to model aggregation. In aggregation we would only try to combine models that were meant to be combined together. During this time we also developed flattening (for composed models and later, for aggregated models) as a way to use existing software (and simulator) packages.

2.4 Context and Prior Work

This review covers a combination of works from the fields of software engineering, modeling and simulation (M&S) and artificial intelligence (AI). We surveyed several papers on modeling, composition and reuse from a purely theoretical M&S perspective [3, 47, 48, 51–54, 78–80]. We also surveyed several papers on model composition in AI [1, 17, 26, 33, 41, 42, 59, 60].

Snoep et al. [68] show it is possible to construct a large model in a bottom-up manner by manually linking together smaller modules. They demonstrate this by combining a glycolysis pathway model with a glycoxylate pathway model. Bulatewicz et al. [9] suggest an interface for model coupling and provide a number of solutions, from a brute force technique to using frameworks specifically designed to support coupling. A number of authors from domains both within [23, 32] and outside of [14, 19, 28, 37, 69] systems biology find that successful composition (or model "reuse") requires components that are specifically designed for the purpose, and their experiences served as motivation for our approaches to model composition and aggregation.

Spiegel et al [69] provide a case study of model context for composability and reuse. They identify a taxonomy of validation constraints and found that this was a harder task than previously believed. The authors conclude that a library of components designed for composition is needed for successful composition/reuse. Having a library of composable elements will enable modelers to rapidly construct large models and might be an area of future research. Kasputis and Ng [28] describe a composable simulation framework and state the composition is made easier through standardized component descriptions. Standardized descriptions allow modelers to classify models/modules into different classes in order to represent a model using its description. Using standardized descriptions could potentially decrease the development time for modelers who need to understand their models before connecting them together and might be yet another area of future study for our work. Our approach allows composing models together using a number of different techniques:

1. By merging/fusing submodels together.
2. By manually connecting/composing submodels together using new language features that fully explains the interactions between components within a composition.
3. By modifying submodels before composition.

Bulatewicz et al [9] and Liang and Paredis [34] describe different approaches to the connections within a model composition. Bulatewicz et al [9] suggest using a potential coupling interface for model coupling and provide a number of solutions, from a brute force technique to using frameworks designed to support coupling. Bulatewicz et al [9] identify different types of coupling and provide a detailed description of their coupling interface. Liang and Paredis [34] describe a port ontology for automated model composition. While automating composition is outside the scope of this dissertation, the ontology for representing ports is useful in detailing the different roles and functions port structures can take. Liang and Paredis [34] state that the ports provide and specify interaction constraints between components and argue that an ontology is needed to resolve ambiguity of components within a composition. The two approaches to support unambiguous representations are labels and metadata. This dissertation uses unique names for port structures to resolve the ambiguity as these labels are easy to create and understand by modelers. The metadata approach suggested by Liang and Paredis [34] assigns primitive and compound attributes to the terms used to define concepts and is not relevant as the port structures in this dissertation can only be of a certain type. They stress that the benefit of aggregation over composition is that there is no duplication of information that is already captured within the submodels.

Proposals have been made within the SBML community [18, 20, 64, 77] that describe the mechanics of composition (or aggregation) through additional SBML language features (referred to herein as "glue"), as we will do. The common idea among these proposals is to support the composition of larger models from smaller ones (submodels). In all these proposals a model can contain:

1. Submodels: Models can be contained within an SBML document or can be externally referenced.
2. Instances: Models may contain one or more instances or (copies) of submodels. Composed models represent a hierarchy of submodel instances connected together
3. Links: Models may contain directional links between objects (SBML components).
4. References: Components within a model can be referenced from another model.

However, none of these proposals have been published in the peer-reviewed literature, nor to our knowledge have any been implemented. While some commercial tools might have more or less support for various forms of composition, we are unaware of any non-proprietary implementations for model composition (or aggregation) in this application domain, or any publications describing proprietary features in commercial applications. Composition and aggregation for pathway models remains very much an open problem.

In Finney's Model Composition proposal [18], a model can be composed of instances of submodels, which themselves are full-fledged models. Other language features discussed in Finney's proposal include <submodel> and <instance> structures. An <Instance> structure refers to submodels and represents a copy of that model within the current model. Finney lists three ways of describing composition in SBML by creating connections between components in different models using links, ports or direct links. The <Link> structure connects two components together directly. Restrictions are needed for linkages, but it is not clear how far this should be defined as part of the language.

Finney's <Port> structure creates interfaces of components within a model. It should be noted that the <port> structures described by Finney are different that those presented in Chapter 5. Ports in [18] are interfaces for components within a model and not for the model itself. There are a number of arguments for and against the use of interfaces. Interfaces provide a contract between a submodel and the model composed using it. Several different submodels can implement a contract with the same interface, as these submodels may use different simulation packages and/or encode different hypothesis for modeled phenomena. An interface facilitates the documentation of the functions of the submodel from the perspective of a modeler wanting to reuse a submodel. The argument against the use of interfaces is that while ports have no biological analogy, nonetheless biology built in modules, with limited interconnections, is only useful to structure human knowledge of those networks (they have no biological analogy) and that ports add an additional and unnecessary overhead. .

The <Direct Link> structure enables direct access to components within a submodel without having to define it beforehand. Direct links can be used in cases where models are not easy to construct using the above architecture. An example is composing of two models together by creating a reaction between them. Finney's approach duplicates everything in one model within another. Alternatively, modifying the existing SBML element <simpleSpeciesReference> can make this connection, to allow referencing a component

directly. In an object-oriented analogy, submodels are analogous to class definitions and instances to object declarations. Two types of connections are possible, one is analogous to pointers (direct links) and the other to overloading parameters (links).

Weimar's method of modularization [64] considers SBML <modules> and their dependencies. Modules are defined as the smallest part of the SBML definition, which can be removed independently of other modules. There are two types of dependencies that exist, syntactic and semantic. The syntactic dependencies exist due to the XSD schema, while the semantic dependencies exist due to variations in intuition in English text. dependencies Tools for modularization should create XML schema for any subset of modules, create UML modules for a subset and analyze SBML files and determine module usage. More information is needed to provide an in-depth analysis, as there is no proposal currently available for this approach.

Ginkel's proposal [20] highlights the advantages of modularity and describes what is necessary for creating modular models. These include: modules (encapsulated logical/physical submodels) with the same set of elements as the SBML model; namespaces (hierarchical names) to access and specify parts of modules; interfaces to integrate modules into a larger model; model assembly consisting of model instantiation and connection; and parameterization to adjust initial and parameter values and compartments of model instances. Ginkel considers separating the interface from the implementation of a model by creating terminals representing inner species to the outside or outer species to the inside. Links establish connections between terminals of model instances, which contain attribute information for species and reactions. An SBML extension proposed by Ginkel would have <listOfTerminals> composed of a list of <terminal> elements that define model interfaces. The scope of the interfaces and what to create instances of is not well-defined as yet. <ListOfLinks> contains lists of terminals and species that should be connected together. The links establish mathematical equations, and <speciesSpecification> and <reactionSpecification> allow changing attributes, but prevent the addition of new parts to the model instances.

A number of questions exist with Ginkel's approach. Areas of uncertainty include how to deal with units and whether they are separate from the SBML document/model. The distinction between a module and a submodel remains ambiguous. Ginkel describes a submodel as a separated part in an SBML model that contains a namespace for its elements but does not define an interface. For connections to the parent model or other submodels you can establish links to all inner parts of the submodel by (possibly hierarchical) model references. A module is also a separate part (as is the submodel), but defines an accessible interface. The parts in an interface can be safely linked to other modules. Everything else is hidden and can be changed by the modeler without notice. This allows for different implementations or model improvements without breaking other "applications" [20] of the module. Some of the other questions we have that remain unanswered in Ginkel's proposal are whether separate namespaces are needed for all terminals, whether additional specifications are possible and if modelers could/should override the <isA> attribute for <modelInstance> in <listOfModelInstances>.

Webb [77] proposes language constructs that enable composition by specifying links to models and model components from other model/composition files. Webb argues that new syntax is needed in SBML to

represent the distinction between an actual element (a Species element) and a reference to an element (used in the definition of a component). Webb takes a document view, which encompasses the use of models as a presentation device in line with how XML is handled. The reasoning behind this approach is that XML comes from a document-centric view of a collection of data. Since SBML is based on XML, it makes sense to use the facilities of XML rather than trying to invent a representation that looks more like a programming language. There are two parts to identifying a component, the instance of the component to reference and the element in the component being referenced in a particular element of the referencing model. References to the component definition are made through an XLink attribute. <listOfComponentInstances> contains the list for defining the components and associated element references used in the composed model. References to model elements are given by the element <elementReference> and model component instances are defined by the element <componentInstance>. Connections between components are defined by the element <link>.

2.4.1 Related Tools

A small number of tools exist that provide limited support for composition (in the context of pathway models) in some form or another. However, none of the tools reviewed provides any support for composition in SBML. SBMLmerge [65] is a tool for building large models from smaller components, but does not support model composition or aggregation. Pathway Builder [50] skirts the issue of model composition. While a user can arrange elements hierarchically in the diagram, the actual model is kept flat.

ProMoT (Process Modeling Tool) [21,55] is an equation-based process modeling tool where encapsulated modules are ordered in an object-oriented form of inheritance. Terminals within models act as interfaces and enable variables to be exported for use outside the model/namespace they are in. Links exist connecting different terminals and provide a rudimentary ability for model composition in this manner. Modules in ProMoT are logical, encapsulated groupings of modeling elements that represent compartments which contain reactions, species and special signaling parts. ProMoT provides support for modularity and hierarchical modeling. It uses object-oriented models, composed from modules, and has its origins in process engineering. It provides support for importing/exporting standard SBML (Level2).

E-cell System [70] is an object-oriented software suite for modeling, simulation, and analysis of large-scale complex systems. Models are developed using EML (E-cell modeling language) which is XML based. E-cell provides multiple simulators and has weak support for SBML. Support for modularity and model composition is under development and is in the early stages. E-Cell uses an architecture where the complete model may be modularized through compartments. In this sense, modules must have some physical border and are not only logical or functional groupings but represent an object in the physical topology of the cell. E-cell has three fundamental object classes: Variable (represent state variables), Process (represent discrete or continuous changes in values of the Variable objects), and System (container of these classes of objects and can contain other Systems). The System class has three subclasses, LogicalSystem (list of Variables, Processes, and sub-Systems, but no volume), VirtualSystem (list of Processes and sub-Systems)

and CompartmentSystem (represent a compartment).

Teranode Design Suite [71] does not directly provide any support for composition but it does provide some support for grouping models together. A more appropriate term for model composition in Teranode is model collection. The Teranode Design Suite (using its own file format and not SBML) is able to build a collection of models. Models can interact with each other and components within a model are able to be linked across models. Models in Teranode Design Suite are analogous to compartments in SBML. Model collection is therefore analogous to being able to link species in different compartments to each other.

Chapter 3

Model Fusion

Model Fusion is a step-by-step process to build large models by merging two or more submodels. Unlike composition or aggregation (where submodels are connected but not modified), fusion irreversibly changes the submodels in the process of combining them. In fusion there is no "glue" (additional SBML language constructs) to describe how submodels are combined or connected. The goal of fusion is to combine submodels into a single unified model containing the aggregated information (without redundancies) across the original collection. Our approach to fusion is to provide a tool that aids modelers attempting to perform the fusion process. Fusion involves two steps, known as name resolution and automatic merging. During resolution, naming conflicts between SBML components (e.g. species, reactions, etc) are resolved, thus removing redundancies and expressing equivalences. During merging the fused model is created based on information supplied during resolution. The fusion process is discussed in more detail below using the following example: two models, m_1 and m_2, each containing two chemical species (*A* and *B* in m_1, *A* and *D* in m_2) will be fused together to produce the fused model (m_f).

This chapter introduces the process by which models are fused together. Section 3.1 describes how to resolve naming conflicts and identify equivalences between components within models. Section 3.2 describes the prototype software developed to implement the ideas of this chapter. The terms *Wizard* [40] (used in Windows based operating systems) and *Assistant* (used in Mac OS X) are synonymous and describe a user interface where the user is led through a sequence of dialogs in order to perform a task. The fusion prototype, based on the wizard interface paradigm, provides a step-by-step guide that walks users through the fusion process. Finally, Section 3.3 describes the fusion process with an example that models the protein interaction network controlling cell division.

Contents

3.1 Mapping Tables

Fusion provides a systematic approach to combine models together using mapping tables. Mapping tables resolve naming conflicts of SBML components between submodels. During fusion, the modeler produces a mapping table for each of the the various SBML component types (compartments, species, reactions, etc.). The mapping tables enable a modeler to express equivalences between components using their names.

3.1.1 Layout and Setup

The mapping tables for fusion have a predefined layout. Each column in a mapping table represents a model, and each row represents an SBML component in that model. Duplicate names within a model are not allowed. For example, a species name may occur only once per column. The first column in the mapping table is reserved for the fused model (m_f for the above example). The two actions available to the modeler during fusion are:

1. define two or more SBML components to be equivalent
2. remove the equivalence definition between two or more SBML components (which have previously been incorrectly equivalenced).

The modeler can decide how the submodel data is arranged and displayed in the mapping tables. The first option is to place all components with the same name on the same row (Table 3.1). The second option is to place every component on a different row in the mapping table (Table 3.2). Another useful feature to minimize the amount of work done by the modeler is to automatically fill in the names of components in the fused model column on rows where there are no naming conflicts (Table 3.3). After manual manipulation of the mapping table by the modeler, if the species name in each cell is different on a single row, the cell in m_f is left empty to enable the modeler to either define its name or select a name from a list of options. This occurs when a species exists in all the submodels, but is named differently in one or more of the submodels. Automatically filling in the name of components for the fused model is based on a component's name/id matching other components, and not whether the component represents the same biological entity. In this dissertation the options selected for the mapping tables is to place components with the same name on the same row and enable the mapping tables to partially complete the fused model column. Components with the same name will be placed on the same row and it is up to the modeler to distinguish when components that represent different biological entities share the same name.

	m_f	m_1	m_2
1		A	A
2		B	
3			D

Table 3.1: Fusion mapping table where species with the same name are on the same row

	m_f	m_1	m_2
1		A	
2		B	
3			A
3			D

Table 3.2: Fusion mapping table where each species is on a different row

3.1.2 Component Dependencies

Forcing a fixed order during name resolution resolves dependencies across SBML component types, which exist as some components are referenced in other components. For examples, we must resolve the identities of compartments (which represent the bounded space in which species are located) before species, since each species stores a reference to its containing compartment in terms of a compartment identifier. Fortunately, the following ordering for the eight SBML component types has no conflicting dependencies:

1. Compartments
2. Species
3. Function Definitions
4. Rules
5. Events
6. Unit Definitions
7. Reactions
8. Parameters.

In other words, by fusing the component types in this order, we resolve all dependencies before they are encountered.

3.1.3 Construction

The construction of the species mapping table using the example models m_1 and m_2 is shown below; the other mapping tables are constructed using the same process. Every column (c) in the species mapping table corresponds to a particular model ($m_1 ... m_M$), where M is the number of models. Every cell in a column is either empty or contains a distinct species name. Duplicate names within a model are not allowed, therefore a species name will only occur once in any particular column. The total number of columns is $i + 1$, as the first column (c_0) is reserved for the fused model (m_f).

Each row (r) in the species mapping table corresponds to a distinct species in some submodel. If s_i is the number of species in model m_i, the maximum number of rows is in the mapping table is $\sum_{i=1}^{M} s_i$. A cell in column c_1 contains a species from the list of species in model m_1. The modeler is able to change the name of a species in m_f, but is unable to change the name of species in any of the other columns/models.

Suppose species *A* represents the same biological entity in both models (m_1 and m_2), then m_f also gets a species with the same name initially (Table 3.3: row 1). It should be noted that the name of the species in m_f can be changed by the modeler at anytime. There are no equivalent species in either model for the species *B* and *D*. The initial species mapping table is shown in Table 3.3.

	m_f	m_1	m_2
1	A	A	A
2	B	B	
3	D		D

Table 3.3: Initial Species Map based on options selected by modeler.

	m_f	m_1	m_2
1	A	A	A
2		B	D
3			

Table 3.4: Intermediate mapping table to show what happens when a modeler assigns two species (species D and species B in row 2) with different names to be equivalent.

	m_f	m_1	m_2
1	A	A	A
2	C	B	D

Table 3.5: Intermediate mapping table where modeler chooses what species to place in fused model. The empty row is automatically removed from the mapping table and the modeler enters a species name for the fused model for Row 2.

When a modeler defines two species with different names to be equivalent to each other, a number of events occur. In Table 3.4, the modeler defines species *D* in m_2 to be equivalent to species *B* in m_1, by "picking up" and moving species *D* from row 3 to row 2. At this point, the previous occurrence of species *D* is deleted (in row 3, column m_2 of Table 3.4) to ensure that there are no duplicate names in any single column (or model). The empty row is automatically deleted (to reach the status shown in Table 3.5). The modeler can select which name to give this species in the fused model from the following options: (1) B; (2) D; (3) some user defined name. If a new species name *C* is specified for row 2 in the fused model, the resulting m_f contains the species *A* and *C* (Table 3.5).

If two species (say species *A* in m_1 and species *A* in m_2) were incorrectly identified as being equivalent to each other (Table 3.3), the reverse process occurs. The modeler must redefine what species are equivalent to each other by setting the other cells in that row to empty, as is the case for the species in row 1, column m_2 in

Table 3.6. When a species name is replaced by a empty cell, a new row is created for it in the mapping table. The next step is to allow the modeler to fill the cells in column m_f with new names as required (Table 3.7).

	m_f	m_1	m_2
1	*	A	
2	B	B	
3	D		D
4			A

Table 3.6: Mapping table showing what happens when two species, with the same name (A), are declared unequivalent. Note that the cell marked with an * can either be set to A or something the modeler specifies.

	m_f	m_1	m_2
1	A1	A	
2	B	B	
3	D		D
4	A2		A

Table 3.7: A completed mapping table, produced by assigning new species names in Table 3.6.

3.2 Fusion Wizard

The fusion prototype follows a wizard interface paradigm, which means that information is solicited from the modeler in a step-by-step process. A modeler can navigate across the different panels of the wizard in both directions (forward and backward) using the "Back" and "Next" buttons on the lower panel of the application (Figure 3.1). A major goal of the Fusion Wizard is to minimize the time and errors associated with combining models by hand. The model generated by the Fusion Wizard can be used with other SBML-compliant software to simulate the response of the model to given conditions. The Fusion Wizard proceeds in three main stages: setup, resolution, and merging. During setup the modeler is guided through various steps that initialize the Fusion Wizard. Once the environment has been initialized, the modeler starts resolving the naming conflicts in the various submodels. Finally, when the SBML component mapping tables have been generated, the application uses this information to automatically merge the submodels together.

3.2.1 Setup

On start up, the Fusion Wizard displays a welcome panel (Figure 3.1) that provides some basic information about the application and the fusion process. The next panel enables the modeler to assign the name of the fused model and the type of the wizard (Figure 3.2). In earlier versions of the application the modeler was only able to fuse two or more models together, however over time more features and functionality were added. The Fusion Wizard became the Fusion/Composition Wizard capable of both fusion and composition.

A modeler is now able to choose the type of the wizard – Fusion Wizard or Composition Wizard – by selecting the appropriate option button. The button labelled “Open Composed Model” will be discussed in Section 4.3.1.

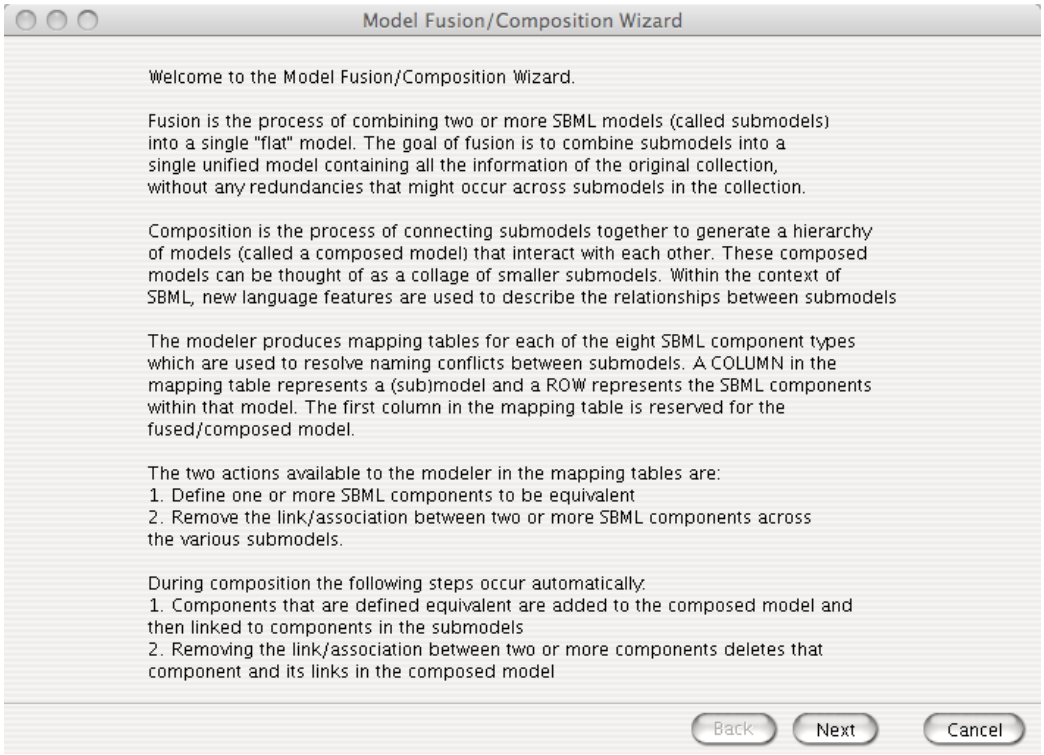


Figure 3.1: Fusion/Composition Wizard welcome panel.

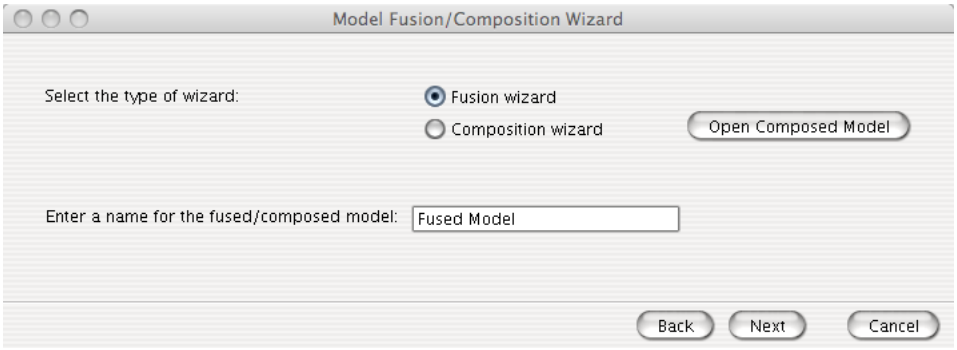


Figure 3.2: Fusion/Composition Wizard name and type panel enables a modeler to name the fused model and select the type of Wizard to use (Fusion or Composition).

Next, submodels are selected from the file chooser panel (Figure 3.1). The file chooser panel is divided into two unequal halves, on the left is a large regular file open dialog window that navigates the local filesystem, on the right is a small text area that lists the submodels already added. In between the two are two

buttons, a right arrow (→) that adds submodels from the file open dialog to the "Submodels Added" list (and thus the application) and a left arrow (←) that removes submodels from the "Submodels Added" list.

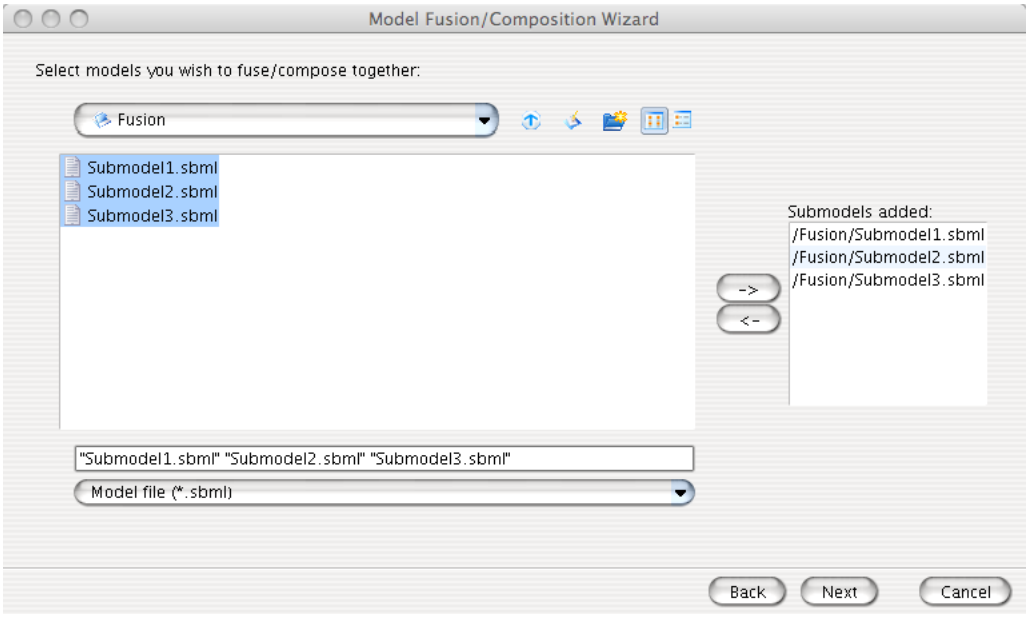


Figure 3.3: Fusion/Composition Wizard file chooser panel.

Once the submodels have been added, the Fusion Wizard updates all component IDs to ensure that there are no duplicate component IDs across the submodel collection. It should be noted that each SBML component contains both a $name$ and id attribute. The names within a model do not have to be unique whereas the IDs do according to the SBML schema. Therefore, to facilitate the modeler all the component IDs are automatically updated to ensure they are unique. The modeler is now ready to initialize and set up the mapping tables. The resolution panel (Figure 3.4) allows the modeler to select a control option for the mapping tables, the modeler directs the system to either:

1. place components of the same name on the same row or
2. place each component on a different row.

The auto-fill panel (Figure 3.5) attempts to minimize the modeler's work by filling up the fused column on rows where there are no naming conflicts. However, the modeler may decide during resolution whether to use these initial choices or to change them. At this stage setup is complete and the modeler can start resolving naming conflicts within the submodels using the mapping tables.

3.2.2 Resolution

Resolving names is vital in order to unambiguously identify all the distinct entities within a model and their relationships with each other. The compartment mapping table (Figure 3.6) is the first panel displayed during

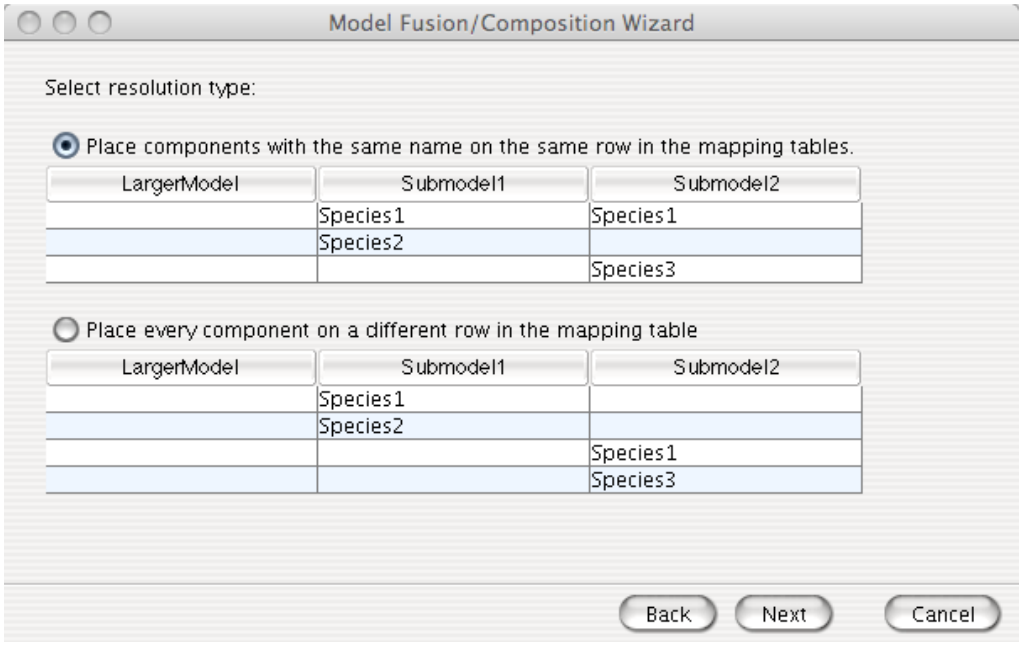


Figure 3.4: Fusion/Composition Wizard resolution type panel for mapping tables.

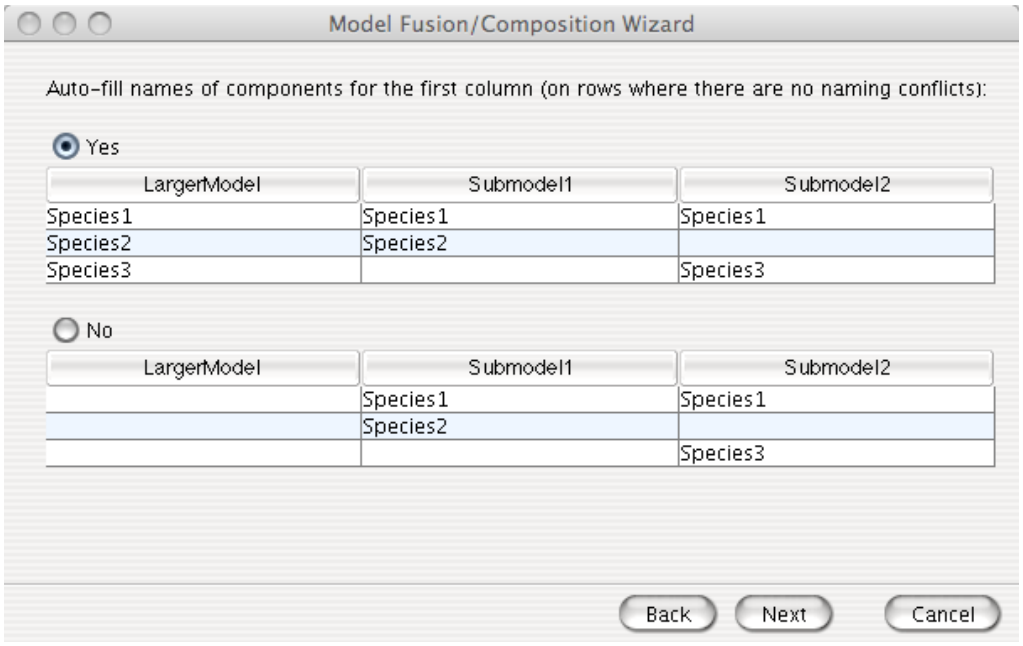


Figure 3.5: Fusion Wizard auto-fill panel for mapping tables.

resolution, followed by the species and function mapping tables (Figures 3.7 and 3.8 respectively). The mapping table panels are divided into two parts: a top part that displays the component map in the form of a mapping table and an optional bottom part that displays attribute information for the components on a

selected row.

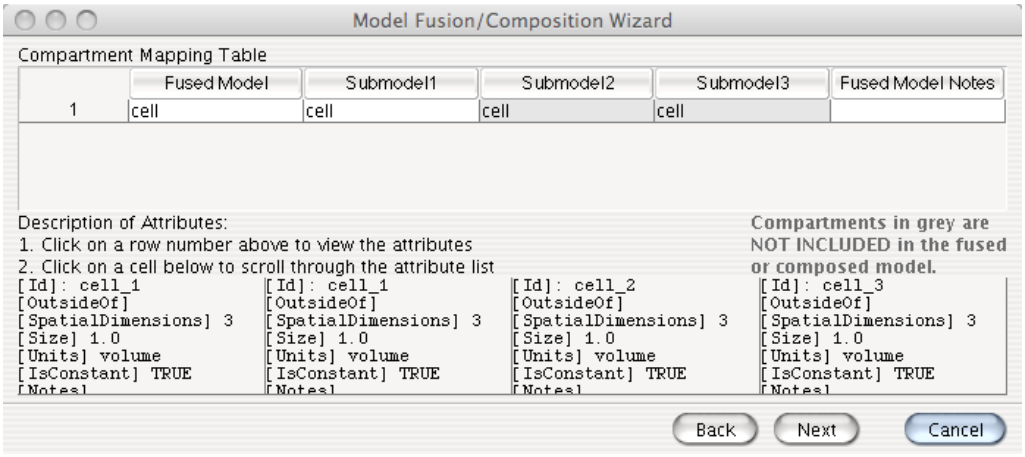


Figure 3.6: Fusion Wizard compartment mapping table.

The compartment component map in Figure 3.6 contains five columns. The row number is displayed on the left of each row and is provided for convenience when dealing with large models. The first column is reserved for the fused model and is named according to the information provided in Figure 3.2. The fused column is followed by the submodel columns, their number depends on the number of models selected in Figure 3.3. The last column (the right-most column) in the component map is reserved for "Notes". The $notes$ attribute in SBML is optional information about a component (stored in the SBML file) intended to be seen by humans. The "Notes" column is meant as an aid to enable modelers to quickly enter relevant information about an SBML component.

The attribute information is only displayed when a particular row in the mapping table is selected. The attribute information is displayed in column form where the first column is reserved for the fused model. It should be noted that there is one less column in the attribute information table than its corresponding mapping table. This is because the $notes$ attribute is already displayed in each cell in the attribute information table. Clicking on the first row in the compartment mapping table (Figure 3.6) displays the attribute information for each compartment called $cell$, such as its id, whether it contains an enclosing compartment ($OutsideOf$ attribute), its $SpatialDimentions$, its $Size$, its $Units$, whether its a constant ($Constant$ attribute), and finally any optional information in the $Notes$ attribute. The cells in the attribute information table are scrollable for viewing but uneditable. The only way to modify the attributes of a component of the model would be in the JigCell ModelBuilder.

Color is used in the component map to provide additional information:

1. a non-empty white cell in any column not reserved for the fused model or notes indicates that the component in that cell is included in the fused model
2. a non-empty gray cell in any column not reserved for the fused model or notes indicates that the component in that cell is excluded from the fused model

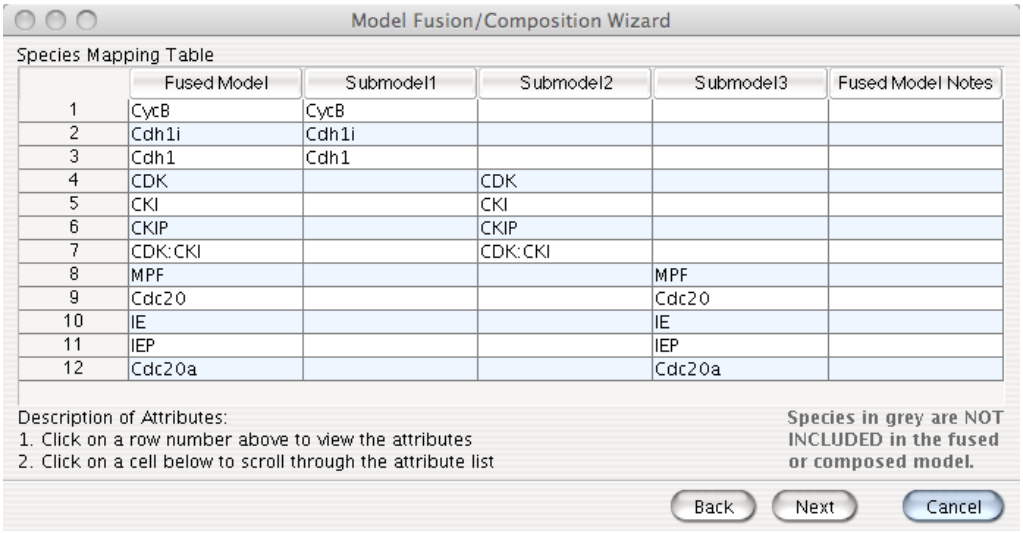


Figure 3.7: Fusion Wizard species mapping table.

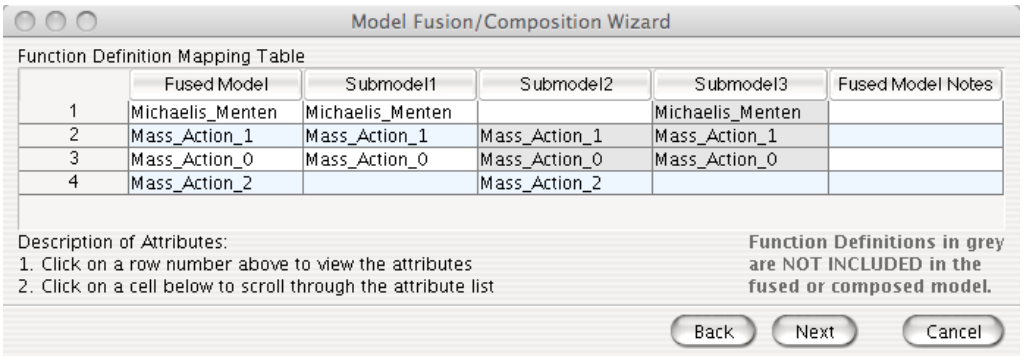


Figure 3.8: Fusion Wizard function mapping table.

3. a non-empty cell with blue text indicates that the component is actually being used in the fused model (and should therefore be included/added to the column reserved for the fused model).

The white and grey colors indicate the originating submodel for a component added to the fused model. The first row in the compartment component map in Figure 3.6 contains three white and two gray cells. The way the component map is set up in Figure 3.6 with three white and two gray cells, it is evident that the fused model will contain a compartment called *cell* from the second column ($Submodel1$). This is a trivial case as all three compartments in the submodels contain the same attribute information, but suppose the modeler wanted to use the compartment from the third column ($Submodel2$) instead of the second column. The following steps and screenshots demonstrate how a modeler would perform this action.

1. The modeler clicks on the cell in the fused model column which in turn displays a combo box (Figure 3.9) which is a combination of a drop-down list and a single-line textbox, that allows the modeler either to type a value directly into the control or choose from the list of existing options. The

fused model column combo box has the following options:

(a) Remove Component - removes the component from the fused model

(b) [Column Number] : Component Name (e.g. [1]: *cell* indicating the compartment named *cell* in $Submodel1$).

2. The modeler then selects the component *cell* from $Submodel2$ ([2] : *cell*)
3. The component map is updated to the state in Figure 3.10. Note that the cell in the $Submodel2$ column is now white while the two cells in the $Submodel1$ and $Submodel3$ columns are now gray.

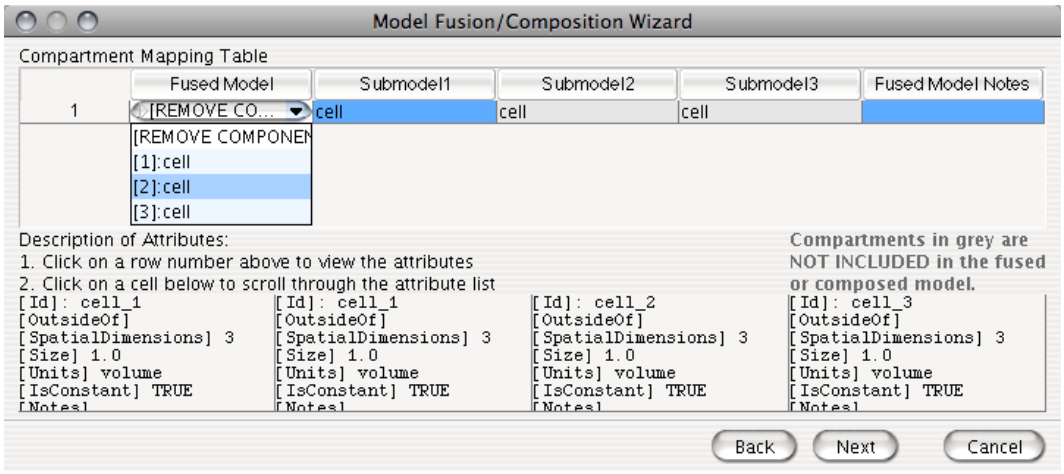


Figure 3.9: Steps needed to add a component from another submodel to the fused model.

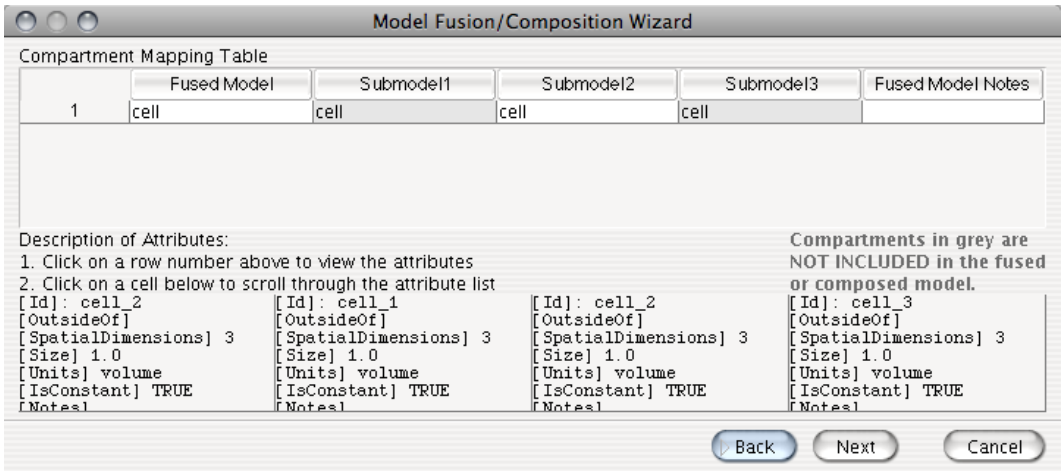


Figure 3.10: State of the component map after the selection from Figure 3.9 has been done.

Another option is to remove the component from the fused model altogether. Figures 3.11 and 3.12 demonstrate this by removing the component called *cell* from the fused model.

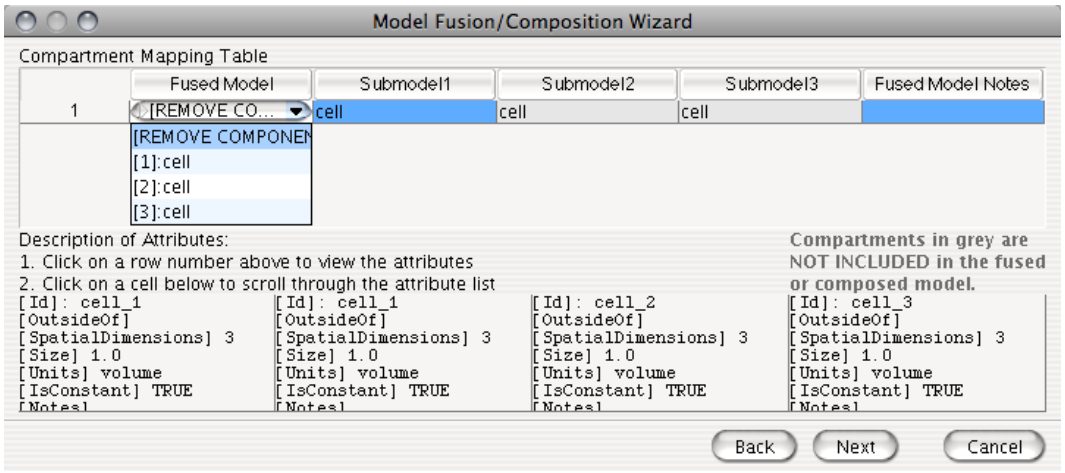


Figure 3.11: Selecting to remove a component from the fused model.

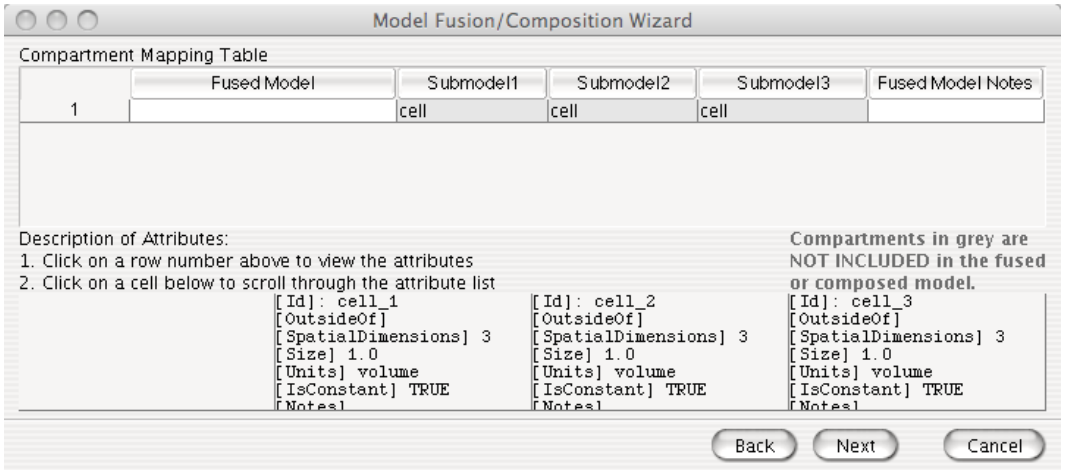


Figure 3.12: State of the component map after the selection from Figure 3.11 has been done.

The component map contains another type of combo box that is displayed when the modeler clicks on a cell in columns reserved for submodels. This combo box has a different function than the one described earlier, it is used to define or remove equivalences between components across the submodels. As an example suppose we wanted to include two components called $cell$ in the fused model, one from $Submodel2$ and the other from $Submodel3$. The following steps, along with accompanying screenshots, demonstrate how this is achieved.

1. The modeler clicks on the cell in the $Submodel3$ column which in turn displays a combo box (Figure 3.13). The submodel column combo box, differs from the fused model combo box and has the following options:
   (a) No Match - removes the link/association between components on the same row. This results in adding a new row to the bottom of the component map that contains the selected component.

(b) A list of all the components in (sub)model. For the compartment mapping table all the compartments will be displayed in this list.

2. The modeler then selects “No Match” (Figure 3.13).
3. The component map is updated to the state in Figure 3.14. Note that the fused model now contains two compartments named *cell* (with different IDs).

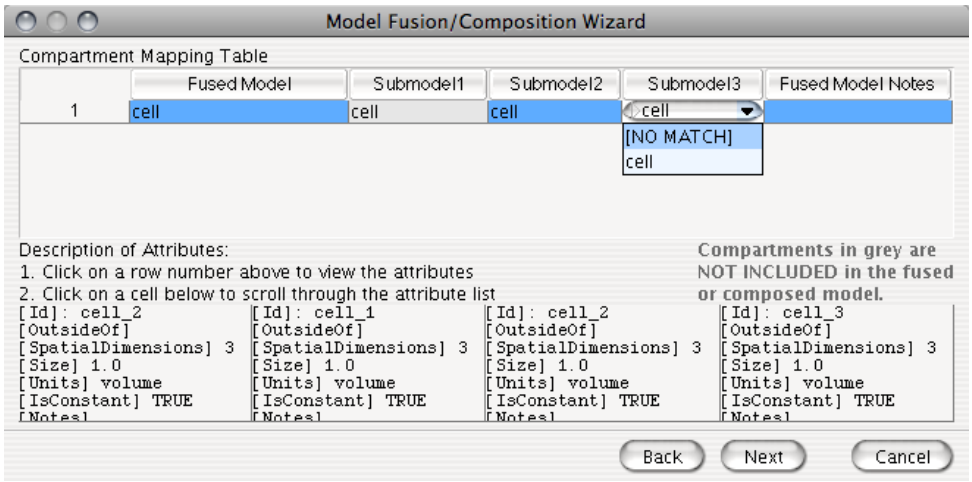


Figure 3.13: Selecting to remove the association between two components from the fused model.

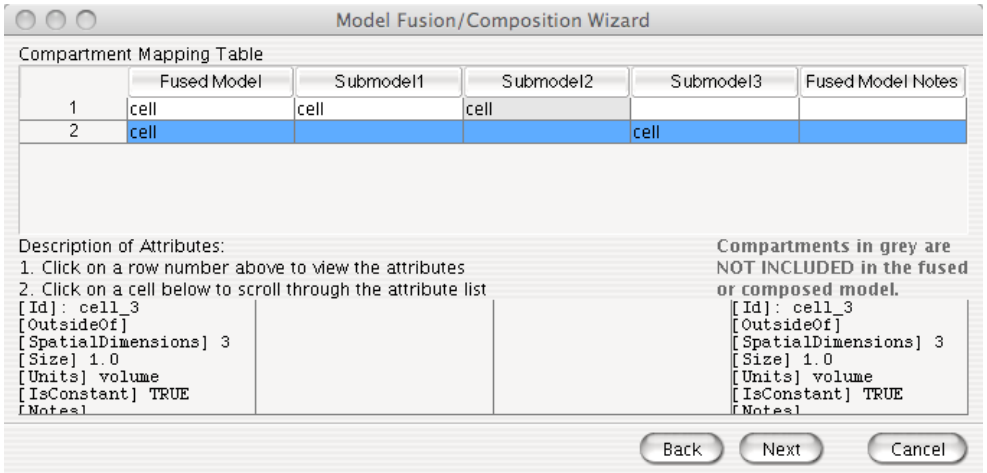


Figure 3.14: State of the component map after the selection from Figure 3.13 has been done.

Section 3.3 describes the fusion process in more detail using concrete biological models. However, the sample models do not have any rules, or events, as such their corresponding mapping tables are empty (Figure 3.15)

The last three mapping tables are for unit definitions, reactions and parameters (Figures 3.16, 3.17, and 3.18 respectively). As there are usually a large number of parameters in models, the parameter mapping

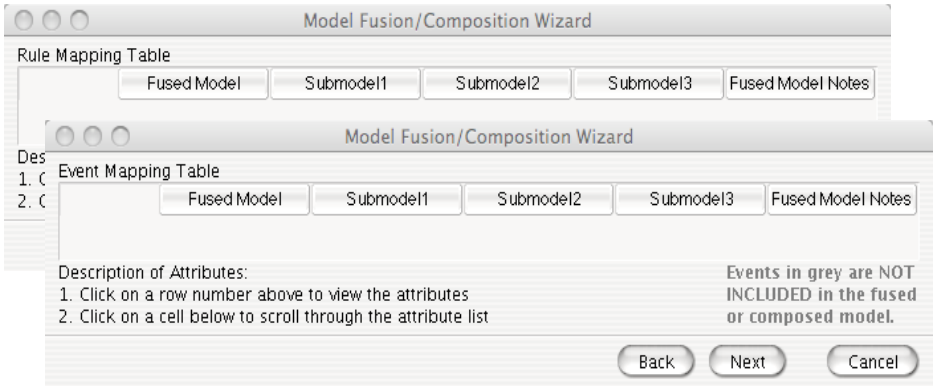


Figure 3.15: Fusion Wizard rule, and event mapping table.

table contains an additional aid using colored text (not found in the other mapping tables). The blue text, as previously mentioned, indicates that a parameter is actually being used in a reaction, function, rule or event in the fused model (and should therefore be included or added to the column reserved for the fused model).

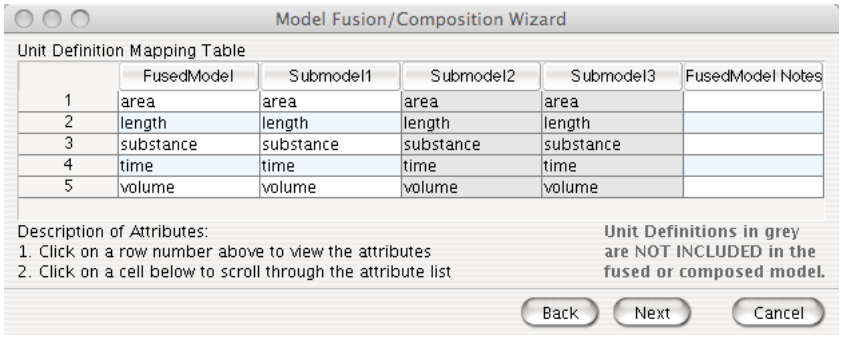


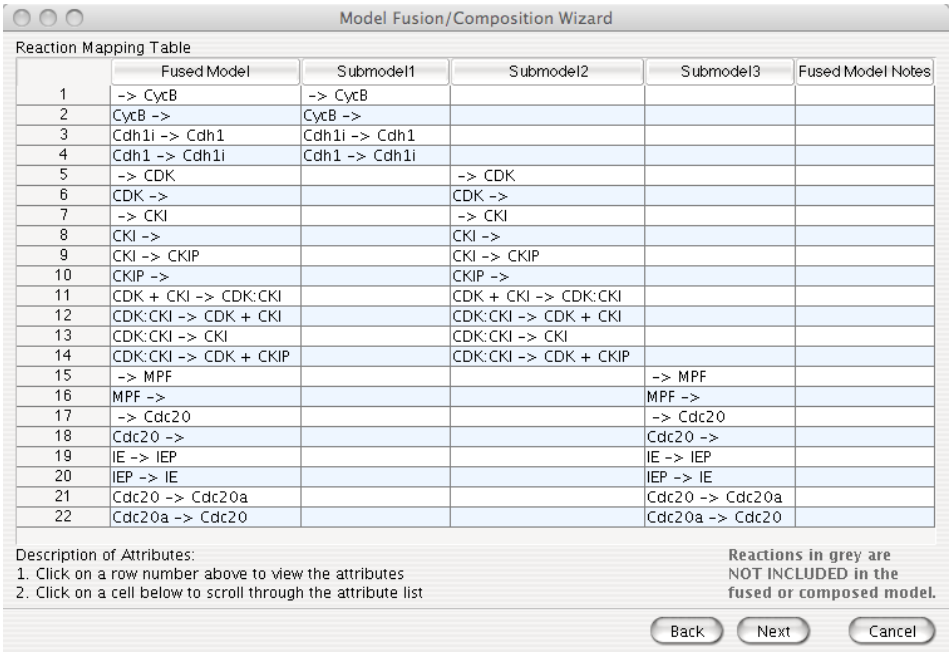
Figure 3.16: Fusion Wizard unit mapping table.

3.2.3 Merging

Once resolution has been completed and all the naming conflicts have been resolved, the modeler can select what to do with their results from a variety of options (Figure 3.19):

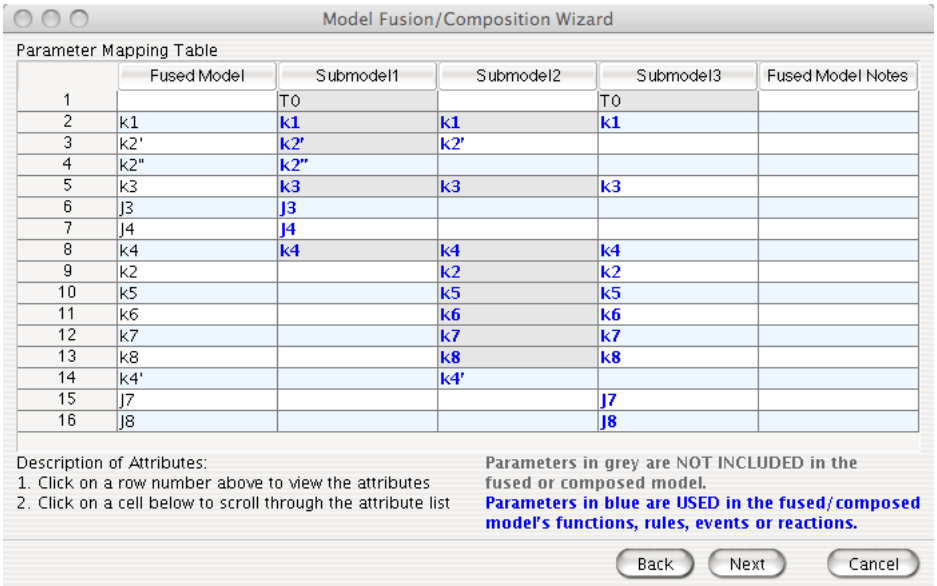
1. Save the fused model to file and exit the wizard
2. Save composed model to file (discussed in Chapter 4)
3. Save the fused model to file and launch the JigCell ModelBuilder
4. Launch the JigCell ModelBuilder with the fused model without saving the model to files
5. Restart the fusion process and reset all values.

If the modeler selects options 1-4 from above the fused model is generated from the reaction networks of the submodels and either written to file or loaded into the JigCell ModelBuilder.



	Fused Model	Submodel1	Submodel2	Submodel3	Fused Model Notes
1	-> CycB	-> CycB			
2	CycB ->	CycB ->			
3	Cdh1i -> Cdh1	Cdh1i -> Cdh1			
4	Cdh1 -> Cdh1i	Cdh1 -> Cdh1i			
5	-> CDK		-> CDK		
6	CDK ->		CDK ->		
7	-> CKI		-> CKI		
8	CKI ->		CKI ->		
9	CKI -> CKIP		CKI -> CKIP		
10	CKIP ->		CKIP ->		
11	CDK + CKI -> CDK:CKI		CDK + CKI -> CDK:CKI		
12	CDK:CKI -> CDK + CKI		CDK:CKI -> CDK + CKI		
13	CDK:CKI -> CKI		CDK:CKI -> CKI		
14	CDK:CKI -> CDK + CKIP		CDK:CKI -> CDK + CKIP		
15	-> MPF			-> MPF	
16	MPF ->			MPF ->	
17	-> Cdc20			-> Cdc20	
18	Cdc20 ->			Cdc20 ->	
19	IE -> IEP			IE -> IEP	
20	IEP -> IE			IEP -> IE	
21	Cdc20 -> Cdc20a			Cdc20 -> Cdc20a	
22	Cdc20a -> Cdc20			Cdc20a -> Cdc20	

Figure 3.17: Fusion Wizard reaction mapping table.



	Fused Model	Submodel1	Submodel2	Submodel3	Fused Model Notes
1		T0		T0	
2	k1	**k1**	**k1**	**k1**	
3	k2'	**k2'**	**k2'**		
4	k2"	**k2"**			
5	k3	**k3**	**k3**	**k3**	
6	J3	**J3**			
7	J4	**J4**			
8	k4	**k4**	**k4**	**k4**	
9	k2		**k2**	**k2**	
10	k5		**k5**	**k5**	
11	k6		**k6**	**k6**	
12	k7		**k7**	**k7**	
13	k8		**k8**	**k8**	
14	k4'		**k4'**		
15	J7			**J7**	
16	J8			**J8**	

Figure 3.18: Fusion Wizard parameter mapping table.

3.3 Biological Example

We illustrate the process of model fusion with an example that models the protein interaction network controlling cell division [13]. In eukaryotes, the cell cycle is controlled by a set of cyclin-dependent protein

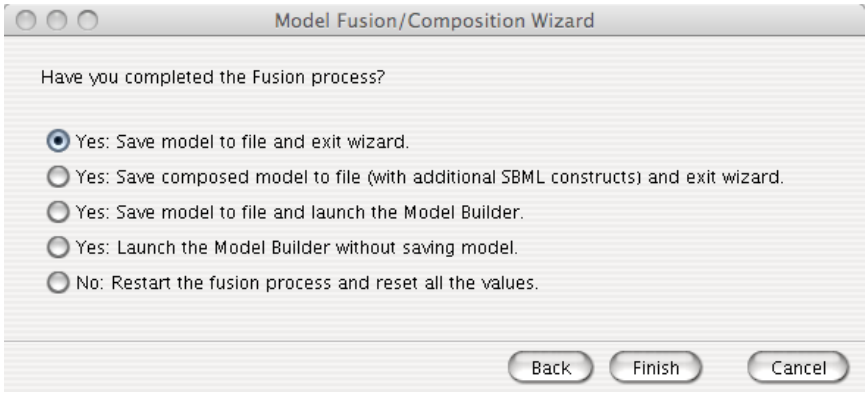


Figure 3.19: Final panel in the Fusion/Composition Wizard.

kinases (Cdks) that phosporylate specific target proteins and thereby initiate the events of DNA replication, mitosis and cell division. As their name suggests, Cdks require association with a cyclin partner (CycA, CycB, ...) in order to be active. The activity of a Cdk-Cyc dimer is controlled by interactions with a variety of regulatory proteins, including the anaphase promoting complex (APC), which, in combination with two auxiliary proteins (Cdc20 and Cdh1), degrades the cyclin component of the Cdk-Cyc dimer, and a suite of cyclin-dependent kinase inhibitors (CKIs), which bind to and inhibit the Cdk-Cyc dimer. A simple model of these interactions (Figure 3.20) is sufficient to reproduce (in simulation) many features of cell cycle regulation in budding yeast [74]. This model shows how progress through the cell cycle can be thought of as irreversible transitions (Start and Finish) between two stable states (G1 and S-G2-M) of the regulatory system.

To illustrate how fusion is implemented using the Fusion Wizard we follow the approach used by Tyson and Novak when building their basic model of cell cycle control in yeast cells [74]. They built their model in stages starting from a simple model and then adding new pieces until they obtained a satisfactory representation of the cell cycle control system. Their starting model (which we will call $Submodel\ I$) deals with the antagonistic interactions between cyclin B-dependent kinase ($CycB$) and a cyclin B-degrading factor ($Cdh1$), as shown in Figure 3.21. The next step was to create a model (which we call $Submodel\ II$) of the interaction between the cyclin-dependent kinase (now called CDK) and a cyclin-dependent kinase inhibitor (CKI), as shown in Figure 3.22. With $Submodel\ I$ and II operating, the cell can exit in one of the two alternating stable steady states - G1 where CKI is high, $Cdh1$ is on, and CDK activity is low or S/G2/M where the reverse is true (CKI is low, $Cdh1$ is off, and CDK activity is high). Finally, they built a second version of the simple model (which we call $Submodel\ III$) of the interaction between "mitosis promoting factor" (MPF) and a different form of the cyclin-degrading factor ($Cdc20$), as shown in Figure 3.23. This model describes the MPF induced activation of $Cdc20$ and is invoked when cells make the transition from the S/G2/M to G1 state. MPF induced $Cdc20$ activation, leads to MPF degradation, which allows CKI and $Cdh1$ to come back. Note that different names have been used here for the same cyclin B-dependent kinase in all three models ($CycB$, CDK and MPF) to better highlight implementation details of fusion.

The three submodels (Figures 3.21, 3.22, and 3.23) are combined using the Fusion Wizard to create an

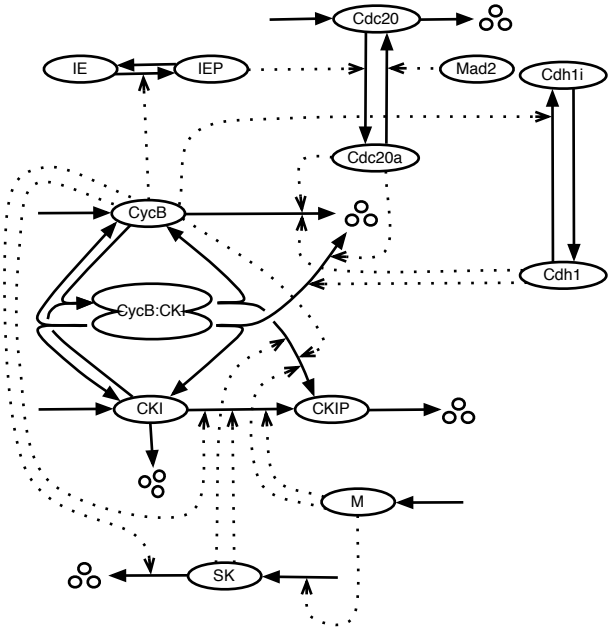


Figure 3.20: Reaction network for cell cycle control in yeast. Icons are proteins, solid arrows are chemical reactions, and dotted arrows represent enzymatic catalysis.

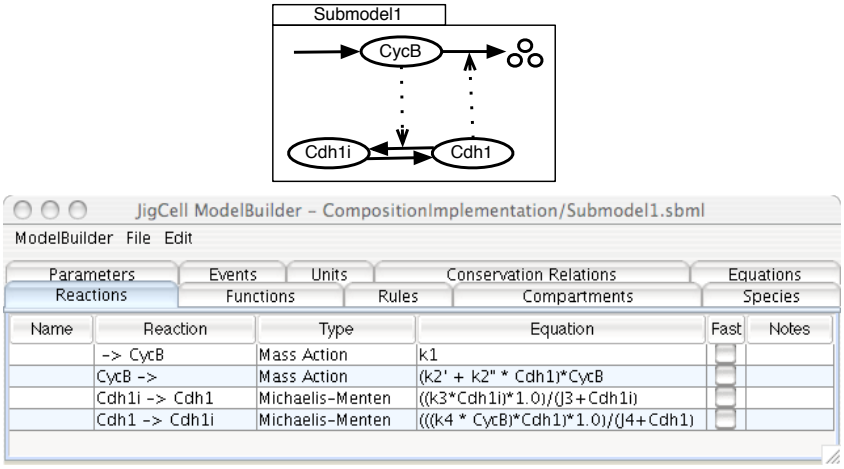
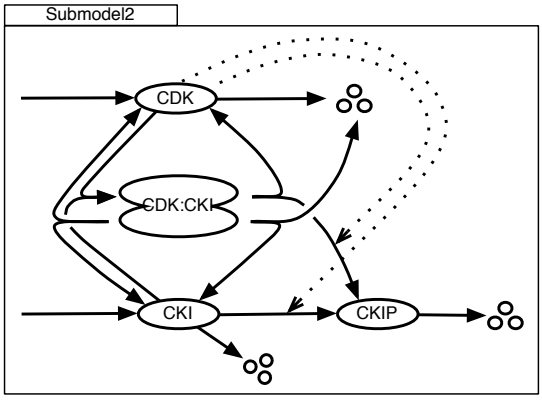


Figure 3.21: Submodel I wiring diagram and reaction definitions in the JigCell ModelBuilder.

$Initial\ Fused\ Model$ (Figure 3.24). The mapping tables in Figures 3.6, 3.7, 3.8, 3.15, 3.16, 3.17, and 3.18 are generated as the initial best guesses by the Fusion Wizard. Both the species and reaction mapping tables need to be updated to take into account that $CycB$ in $SubmodelI$, CDK in $SubmodelII$, and MPF in $SubmodelIII$ represent the same biological species. Thus, the species and reaction mapping tables are updated to their final states (Figures 3.25 and 3.26).

Once the fusion process is completed, the $Initial\ Fused\ Model$ is generated and opened in the JigCell



JigCell ModelBuilder – CompositionImplementation/Submodel2.sbml

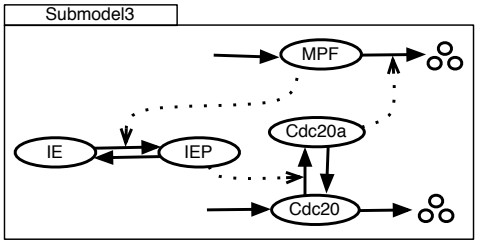
ModelBuilder File Edit

Parameters | Events | Units | Conservation Relations | Equations

Reactions | Functions | Rules | Compartments | Species

Name	Reaction	Type	Equation	Fast	Notes
	-> CDK	Mass Action	k1		
	CDK ->	Mass Action	k2*CDK		
	-> CKI	Mass Action	k3		
	CKI ->	Mass Action	k4*CKI		
	CKI -> CKIP	Mass Action	(k5 * CDK)*CKI		
	CKIP ->	Mass Action	k6*CKIP		
	CDK + CKI -> CDK:CKI	Mass Action	(k7*CDK)*CKI		
	CDK:CKI -> CDK + CKI	Mass Action	k8*CDK:CKI		
	CDK:CKI -> CKI	Mass Action	k2'*CDK:CKI		
	CDK:CKI -> CDK + CKIP	Mass Action	(k4' * CDK)*CDK:CKI		

Figure 3.22: Submodel II



JigCell ModelBuilder – CompositionImplementation/Submodel3.sbml

ModelBuilder File Edit

Parameters | Events | Units | Conservation Relations | Equations

Reactions | Functions | Rules | Compartments | Species

Name	Reaction	Type	Equation	Fast	N...
	-> MPF	Mass Action	k1		
	MPF ->	Mass Action	k2*MPF		
	-> Cdc20	Mass Action	k3		
	Cdc20 ->	Mass Action	k4*Cdc20		
	IE -> IEP	Mass Action	(k5 * MPF)*IE		
	IEP -> IE	Mass Action	k6*IEP		
	Cdc20 -> Cdc20a	Michaelis-Menten	((k7*Cdc20)*IEP)/(J7+Cdc20)		
	Cdc20a -> Cdc20	Michaelis-Menten	((k8*Cdc20a)*1.0)/(J8+Cdc20a)		

Figure 3.23: Submodel III

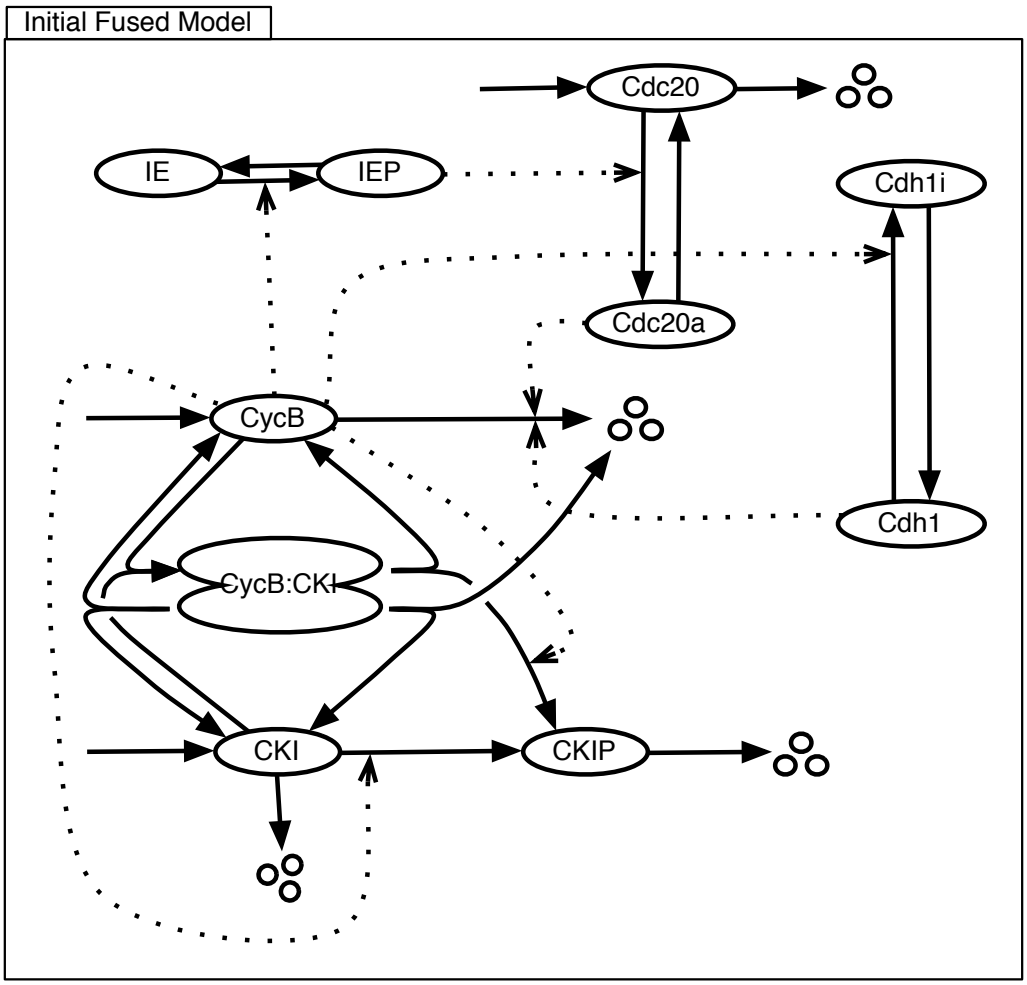


Figure 3.24: Initial Fusion of Submodels I-III.

ModelBuilder for editing. Three additional reactions for starter kinase activation (synthesis and degradation of SK and synthesis of M respectively), are added (in red) to complete the $Final\ Fused\ Model$ (Figure 3.27). SK refers to a "starter kinase" and M represents cell "mass". M increases according to a logistic rate equation, and M is decreased by a factor of 2 each time the cell divides. Cell division is triggered when $CycB$ drops below a certain threshold, as $CycB$ is degraded by $Cdc20$ and $Cdh1$. SK activation is invoked when cells are big enough to make the transition from G1 to S/G2/M. SK accumulates to high enough levels to inactivate CKI, allowing $CycB$ to come up and turn off $Cdh1$. See [74] for details.

These steps produce the $Final\ Fused\ Model$ in Figure 3.27 which must then be simulated to verify that its dynamic properties represents the observed behavior of growing-dividing yeast cells in expected ways. Simulating the resulting model on XPP produces the simulation output shown in Figure 3.28, which closely matches the simulation output from Tyson and Novak's model (Figure 8 in [74]).

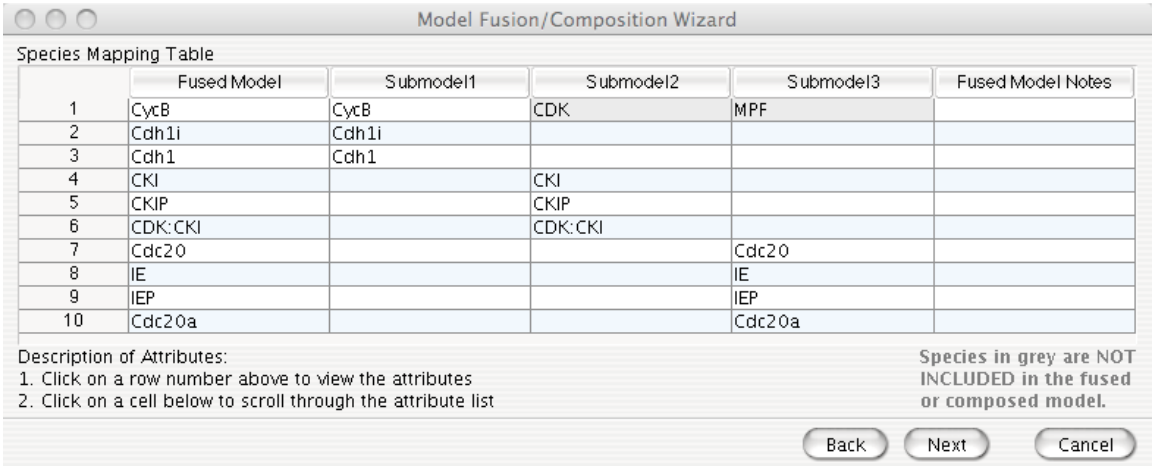


Figure 3.25: Fusion Wizard final species mapping table.

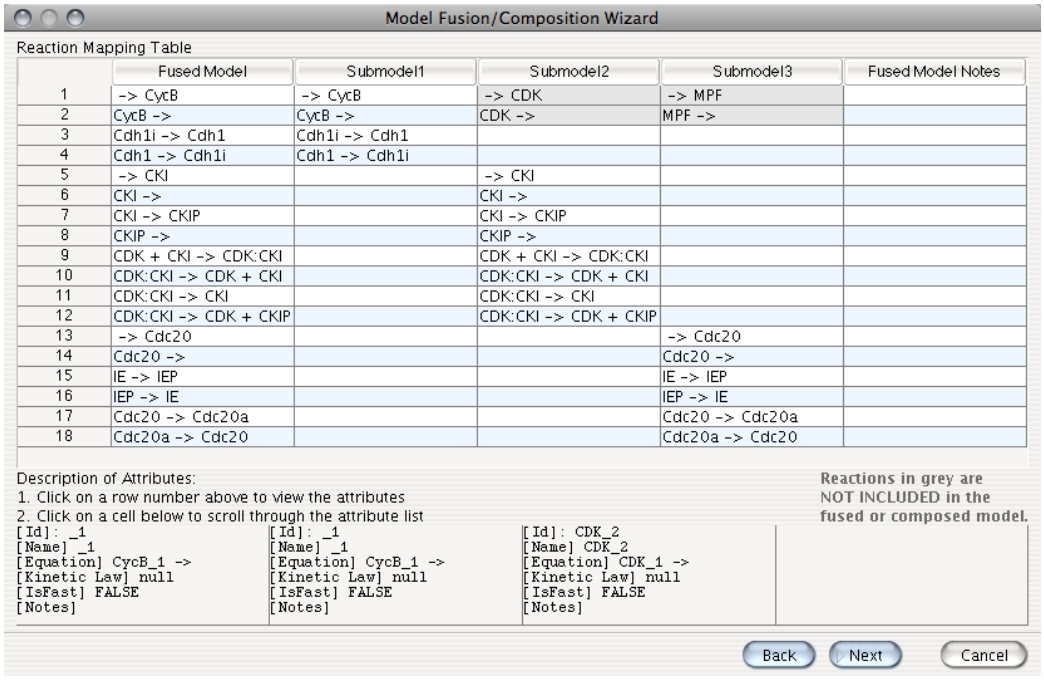


Figure 3.26: Fusion Wizard final reaction mapping table.

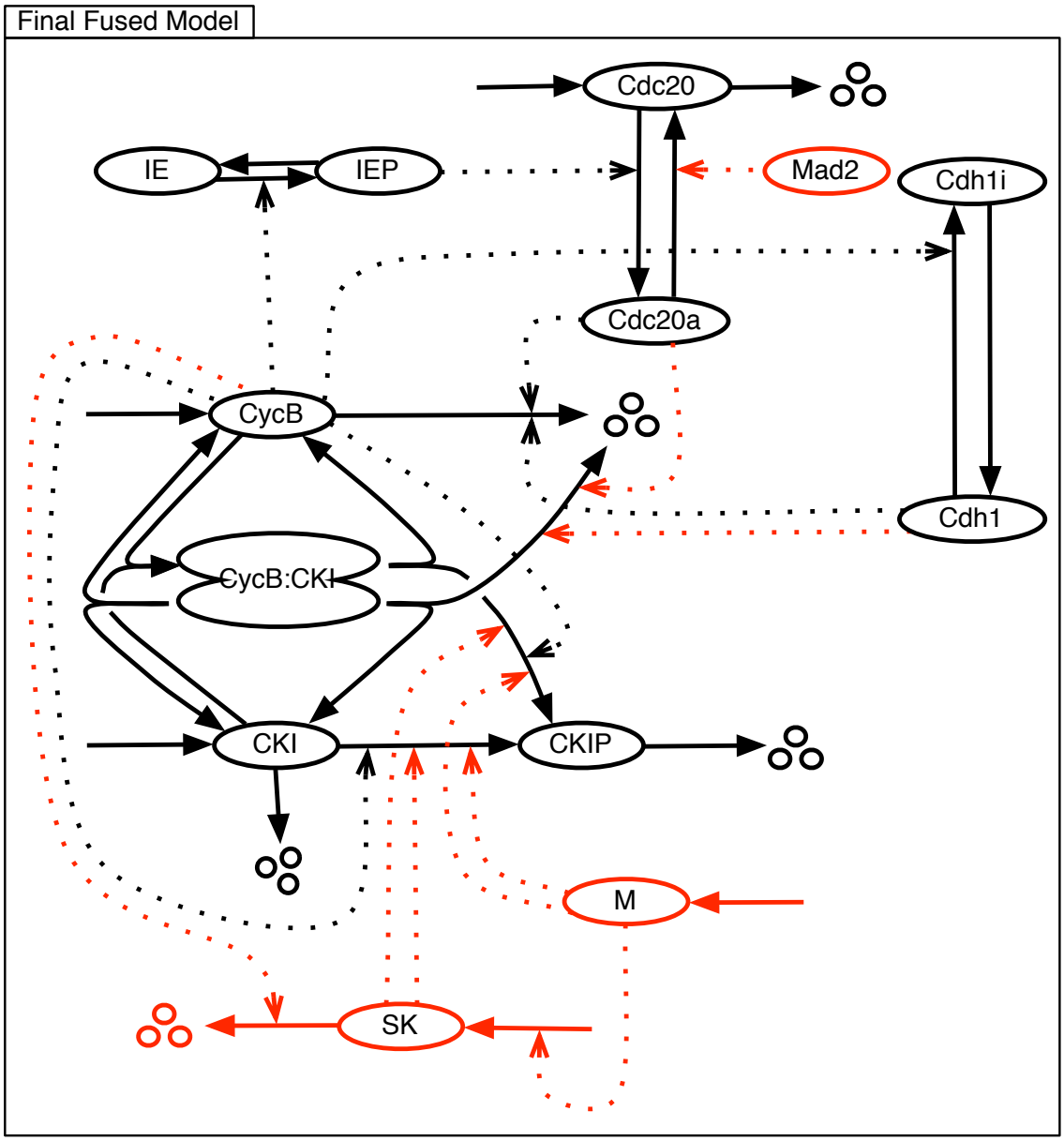


Figure 3.27: Final Fusion of Submodels I-III. Components in red were added after the fusion process was completed.

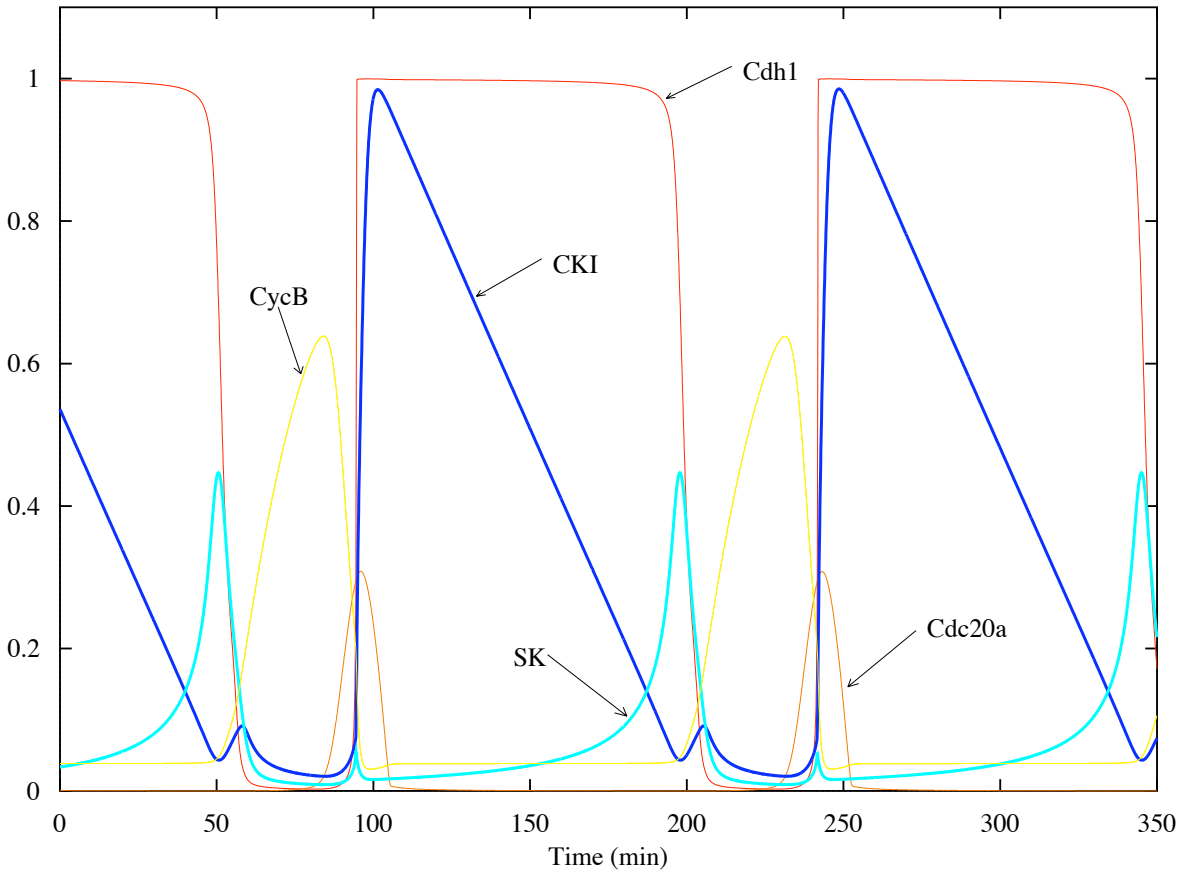


Figure 3.28: Simulation for the $Final\ Fused\ Model$ using XPP.

Chapter 4

Model Composition

Model Fusion was not intended as our ultimate solution for solving the open-ended model composition problem in systems biology, however it was a rather good place to start. The problem with fusion is that beyond some size, fused models will become too complex to grasp and manage as single entities. In this case, it may be more useful to represent large models as compositions of distinct components. Thus, while model fusion is a useful tool for manipulating small to mid-sized models, it does not seem to be a viable solution in the long run. Model Composition provides a potential solution to our goal to build models of large reaction networks, as it is a step-by-step process to generate a composed model (also called a hierarchy of models). A composed model is built from two or more submodels by describing their redundancies and interactions. One can think of models not as monolithic entities, but rather as a collage of smaller submodels. Within the context of SBML, new language features provide a vocabulary to describe the hierarchy of models, and all the relationships among submodels. Composition is a reversible process, in that removing the inter-model interaction description that holds the composed model together recovers the individual submodels. To illustrate composition, consider a large model (called $Global$), composed of two submodels (A and B). Model A contains the chemical species x and model B contains the species y. It is now possible to make a new reaction in $Global$ that represents $x \rightarrow y$, by referring to x and y in A and B respectively. The $Global$ model will only contain a single reaction. The names of reactants and products for that reaction refer to the corresponding species in the two submodels.

This chapter introduces the process by which models are composed together. Section 4.1 describes the SBML language features needed to describe model composition. Section 4.2 describes how to resolve naming conflicts and identify equivalences between components within models. Section 4.3 describes the prototype software developed to implement the ideas in this chapter. Section 4.4 describes the composition process with an example that models the protein interaction network controlling cell division. Finally, Section 4.5 describes the flattening process, the algorithm, and the prototype software developed.

Contents

4.1 SBML Syntax

The section describes the three SBML extensions (<submodels>, <instances>, and <links>) needed to describe composition. A composed model can contain one or more submodels within its structure. A submodel contains a valid SBML model (an SBML <model> structure, with its own namespace, and can itself be a composed model. A simple example (Figure 4.1) shows a composed model (Big) and its corresponding flattened model ($Flat$). Model Big contains a submodel called $Little$, and each (sub)model contains a compartment (names *comp1* and *comp2* respectively) and reaction ($A \rightarrow B$ and $C \rightarrow D$ respectively). The compartment for each submodel is linked, meaning that it is a shared compartment that appears in each representation. In contrast, the reactions are distinct.

After the list of submodels has been declared in the global model, the modeler needs to instantiate the submodels to use/access them through an <instance> structure. Finally, different components (species, reactions, etc) within either the submodels or the global model are connected/accessed using <link> structures.

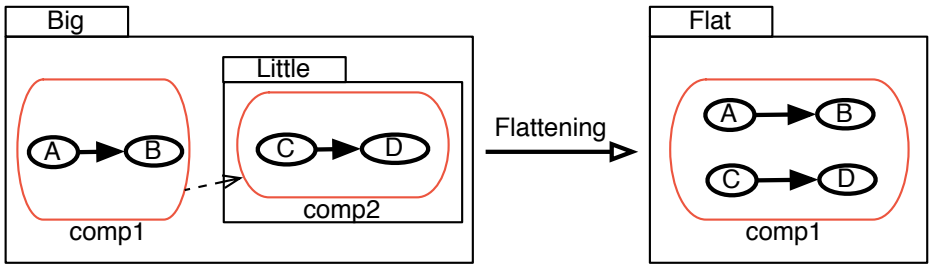


Figure 4.1: Composed model showing a link between two compartments in different submodels and its corresponding flattened model. The dashed line connecting the compartments in each model indicates that they are actually the same compartment.

4.1.1 Naming Convention

A naming convention is used to enable modelers to uniquely identify an SBML component (e.g. species, parameters, etc) within a model (or submodel). The proposed format of the naming convention is as follows:

```
<component object="ObjectIdentifier">
   <subobject object="SubobjectIdentifier"/>
</component>
```

We will also represent this same information using the syntax *ObjectIdentifier.SubobjectIdentifier*. This convention makes it possible to refer to SBML components with the same name in different models without having to change their names. Having a predefined naming convention ensures clarity during the composition process, and enables the modeler to uniquely identify all the components in the proposed composition. The format described by this naming convention replaces the component's name if that component is referenced outside its namespace in an <instance> or <link> structure. A species called *mySpecies1* in model $MyModel$ can be be referenced using its modified name, *MyModel.mySpecies1* and is expressed in SBML as follows:

```
<object="MyModel">
   <subobject object="mySpecies1"/>
</object>
```

The <object> structure is used to reference objects and refers to content implied by the <instance> structures. Reference to objects can only made from <link> (Section 4.1.4) and <port> (Section 5.1.1) structures.

4.1.2 Submodel

A composed model can contain one or more submodels within its structure. A <submodel> structure contains a valid SBML model (an SBML <model> structure), with its own namespace and can also be a composed model. Figure 4.2 shows how a composed model ($Huge$) can be created from a previously composed model ($Large$) and another model ($Small$) producing a composition hierarchy.

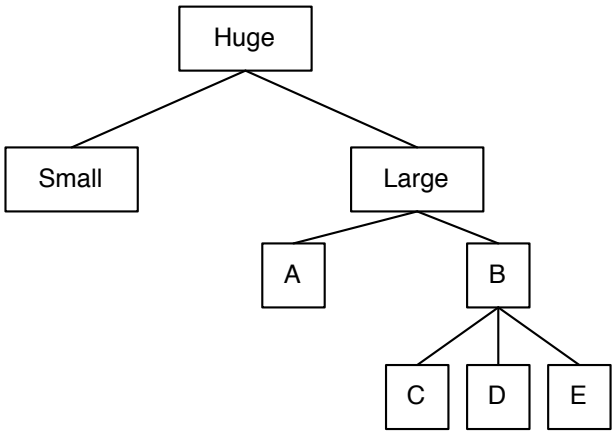


Figure 4.2: Composition Hierarchy

Since there is no restriction on the number of submodels a model can contain, a <submodel> structure is enclosed in a <listOfSubmodels> structure. A simple example (Figure 4.3) shows how model Big contains a submodel called $Little$, both models contains a single compartment (*comp1* and *comp2* respectively). To avoid unnecessary confusion the two compartments were named differently. The sample SBML code for Figure 4.3 is provided below:

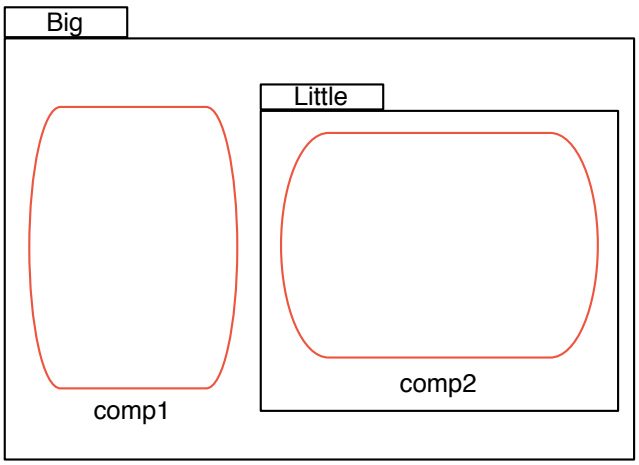


Figure 4.3: The empty composed model (from Figure 4.1) showing two compartments in different models.

```
<model id="Big">
   <listOfCompartments>
      <compartment id="comp1" volume="1"/>
   </listOfCompartments>
   <listOfSubmodels>
      <model id="Little">
```

```
        <listOfCompartments>
           <compartment id="comp2" volume="1"/>
        </listOfCompartments>
      </model>
   </listOfSubmodels>
</model>
```

4.1.3 Instances

Each <instance> structure (enclosed in a <listOfInstances> structure) refers to a particular <model> structure. An <instance> indicates that a copy of a submodel is being instantiated within the current model. Models can be composed of multiple instances of a particular submodel. The instance structure uses the XML Linking Language (XLink) [15] to refer to submodels, as it is a standard mechanism for linking XML elements between XML/SBML documents. An instance of submodel $Little$ (called *Submodel_Little*) can be made within model Big to access submodel $Little$ in model Big (Figure 4.3).

The <instance> structure contains three attributes: an id (a unique identifier for the <instance>), the XLink's $type$, and the XLink's $href$. The $href$ attribute is represented using an XPointer string [22] which is used for locating data within an XML document. The $type$ attribute takes the values $simple$ and $extended$. A $simple$ link is a link that associates exactly two resources, one local and one remote. The direction of the link is from the former to the latter and thus is always an outbound link. An $extended$ link associates an arbitrary number of resources. The participating resources may each be local or remote. For our example we only need to link together two objects (resources) and so the value of the *type* field will be $simple$. The sample SBML code for Figure 4.3 can be extended to include <instance> structures and is provided below:

```
<model id="Big">
  <listOfCompartments>
   <compartment id="comp1" volume="1"/>
  </listOfCompartments>
  <listOfSubmodels>
    <model id="Little">
      <listOfCompartments>
        <compartment id="comp2" volume="1"/>
      </listOfCompartments>
    </model>
  </listOfSubmodels>
  <listOfInstances>
    <instance
      id="Submodel_Little"
      xlink:type="simple"
      xlink:href="#xpointer
      (/sbml/model/listOfSubmodels/model[@id=%22Little%22])"/>
```

```
  </listOfInstances>
</model>
```

The above example shows an *href* attribute where the submodel $Little$ occurs within the same SBML document. If the submodel $Little$ occurred in another SBML document named $temp.sbml$ in the current directory, the *href* attribute of the <instance> structure would have $temp.sbml$ prepended to it.

4.1.4 Links

A <link> structure (enclosed in a <listOfLinks>) connects two entities (SBML components) in a composed model. A <link> should be able to connect two <species>, <parameters>, <reactions>, or <compartments> to each other. Linking components in composition can be achieved using mapping tables similar to those created during fusion, whereby components on the same row in the mapping table will be automatically linked together.

A <link> is composed of two fields, <from> and <to>. The <to> field references an object (the *to object*) whose attribute values will be overridden by the object referenced by the <from> field (the *from object*). The objects referenced by <from> and <to> fields must be of the same type. Only those attribute values that have been declared in the *from object* will be overridden in the *to object*. This is somewhat analogous in *C/C++* to treating the *to object* as a pointer, and the *from object* as its target. However, a *to object* can have attribute values that are retained if no overriding attribute value is declared in the *from object*. Note that if we have two components inside a (sub)model we are still able to link sub-objects of the components using our object/sub-object naming convention. The following example shows how the two compartments in Big and $Little$ (from Figure 4.3) can be linked together (Figure 4.4).

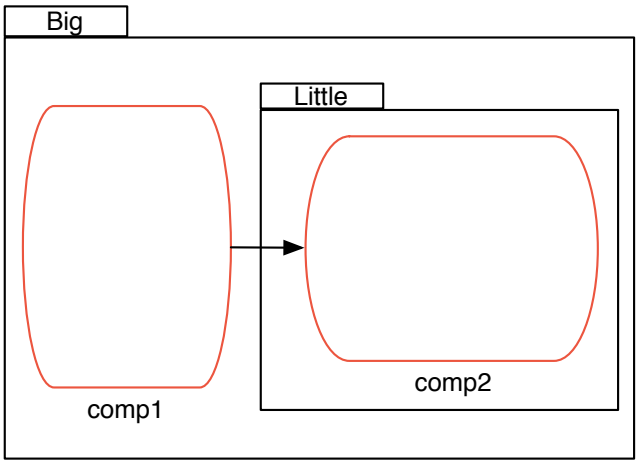


Figure 4.4: The empty composed model (from Figure 4.1) showing a link between two compartments in different submodels.

```
<model id="Big">
```

```
  <listOfCompartments>
   <compartment id="comp1" volume="1"/>
  </listOfCompartments>
  <listOfSubmodels>
    <model id="Little">
      <listOfCompartments>
        <compartment id="comp2" volume="1"/>
      </listOfCompartments>
    </model>
  </listOfSubmodels>
  <listOfInstances>
    <instance
      id="Submodel_Little"
      xlink:type="simple"
      xlink:href="#xpointer
      (/sbml/model/listOfSubmodels/model[@id=%22Little%22])"/>
  </listOfInstances>
  <listOfLinks>
    <link>
      <from object="comp1"/>
      <to object="Submodel_Little">
        <subobject object="comp2"/>
      </to>
    </link>
  </listOfLinks>
</model>
```

The <link> structure contains a $merge$ attribute, whose value can be either true (indicating a $merge$ link) or false (indicating a $replacement$ link). To see the difference, consider models R and T which each contain a chemical species called *S1* with different attributes. *S1* in Model R has attribute $InitialSubstance$ = 1.0. *S1* in Model T has attributes $InitialSubstance$ = 2.0 and $Constant$ = $true$. Linking *S1* in R to *S1* in T with a merge link uses *S1*'s attributes from T.*S1* that have not been declared in R.*S1*. The result is that *S1* has attributes $InitialSubstance$ = 1.0 and $Constant$ = $true$ since it keeps its old value for $InitialSubstance$ and gains the definition for $Constant$. If *S1* in R is linked to *S1* in T using a replacement link (i.e., the $merge$ attribute is false), then only R.*S1*'s attributes are used. The result will be that *S1* will only have attribute $InitialSubstance$ = 1.0 (it has no value for $Constant$). Specifying the type of link for composition requires adding a new field to the mapping table to specify the merge/replacement attribute.

The <link> structure can link certain combinations of differing SBML component types to each other, such as species ↔ parameters and rules ↔ species/parameters. A link can take a <species> structure as the *from object* and a <parameter> structure as the *to object*, and vice versa. The <link> structure should also be able to link together groups of reactions, thus enabling N to N links. The syntax for the <link>

structure should allow zero or more <from> and one or more <to> object references. A link with zero <from> references deletes the <to> object(s). This is useful to remove or ignore a component or a group of components within a particular (sub)model that do not participate in the overall composition. A link cannot contain more than one species or compartment <from> references (otherwise it would be possible to split a species or compartment). Note that a <link> to nothing is not allowed.

4.2 Composition Mapping Tables

We now explain in detail how to generate a hierarchy of models that produce what we call a composed model. It turns out that there are significant similarities between model fusion (Chapter 3) and model composition, as we discovered during the process of developing the fusion tool. The fusion process described in Chapter 3 defines a series of steps taken to merge two or more models together. This series of steps can be viewed as an "audit trail" used in generating the necessary mapping tables. Precisely this same information can be used to describe the set of instructions needed to connect/link the submodels for composition. Both composition and fusion should produce the same simulation results, as the simulation output of both fused and composed forms of a model should be identical. While fusion combines submodels together in an irreversible way, composition simply references submodel components by defining the "glue" that holds the submodels together. A major difference is that in fusion, the explicit description of relationships between entities within submodels is lost, while composition keeps a record of how models were connected together.

Constructing composed models is a bottom-up process as smaller models (both flat and composed) are first composed together to create larger models which in turn can be used to create even more complex models, as previously demonstrated in [68]. The result of composition is to generate a composition tree that describes the hierarchical relations among the various submodels. Changing the connections between submodels in the composed model results in a different composition tree structure. Both model construction and development are iterative processes, as models are usually constructed in increments, with modelers switching back and forth between adding components to a model and fine tuning models through simulations. Since model composition is a combination of constructing submodels and generating composition trees, a tool for composition should take into consideration the iterative nature of the process.

4.2.1 Layout and Setup

Composition mapping tables are constructed in the same way as fusion mapping tables (Section 3.1.3). The order of resolution for SBML component types is also the same as in fusion (Section 3.1.2). The two main differences between the two types of mapping tables is how they are used and the fact that composition mapping tables must also deal with combining both composed and flat models.

The first column of a composition mapping table is reserved for the root of the composition tree (the composed model), followed by its children (submodels). The mapping tables are able to identify equivalent components across the submodels in order to determine which components to add (and link) in the composed

model. The various lists of SBML components (species, reactions, parameters, etc) in each submodel are treated as distinct sets, and the intersection of these sets represent components that occur across all the submodels. For example, only those species that occur within all the submodels will be present in the intersection set and therefore automatically added to the composed model. To signify equivalence, links are automatically created from components in the composed model to components in the submodels. The composed model will also contain modeler defined components (selected during name resolution) from its children, which will also be linked together. Adding and linking components at each level in a composition hierarchy ensures that a composed model will only directly reference components at most one level below itself, but can indirectly reference components across many levels in the composition tree. The four actions available to the modeler during composition are:

1. Automatically add components that occur within the intersection (initially components with the same name) of the immediate submodels to the composed model only, and create links between equivalent components from the composed model to its submodels.
2. Add new user defined/selected components from the immediate submodels to the composed model and create links to the equivalent components in the submodels.
3. Define two or more SBML components within the immediate submodels to be equivalent, add them to the composed model, and create links to the equivalent components in the submodels.
4. Remove the link between a component in the composed model and a component within the submodels (which have previously been incorrectly linked together).

4.3 Composition Wizard

The composition prototype follows the same wizard interface paradigm as in Section 3.2. The Composition Wizard was originally developed as a stand-alone application, but over time as more functionality was added it was combined with the Fusion Wizard (Section 3.2) and renamed the Fusion/Composition Wizard. This was possible as the two processes require the same information to generate either fused or composed forms of a model. The difference between the two forms are in how they are represented and stored in SBML. The Composition Wizard proceeds in three main stages: setup, resolution and composing. During setup the modeler is guided through various steps that initialize the Composition Wizard. Once the environment has been initialized, the modeler starts resolving naming conflicts in the various submodels. Finally, when the composition mapping tables have been generated, the application uses this information to automatically create the composed model.

4.3.1 Setup

On start up, the Fusion/Composition Wizard displays the welcome panel (Figure 3.1) that provides some basic information about the application and the composition process. The next panel enables the modeler to

assign a name for the composed model (Figure 4.5). The button labelled "Open Composed Model" allows the modeler to load an existing composed model into the Fusion/Composition Wizard. If this option is selected, the composition mapping tables are generated based on the information provided in the composed model's <link> structures. An added benefit of the Composition Wizard over the Fusion Wizard is that one is able to save and track progress during the model development process.

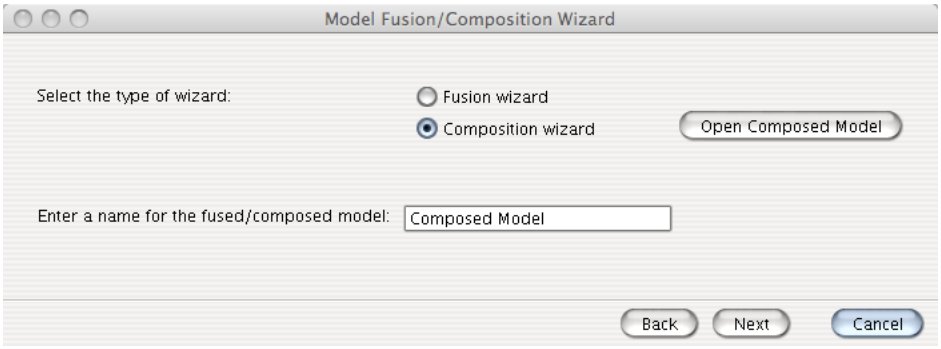


Figure 4.5: Fusion/Composition Wizard name and type panel enables modeler to name the composed model and select the type of Wizard to use (Fusion or Composition).

Next, submodels are selected from the file chooser panel (Figure 3.1). The modeler is now ready to initialize and set up the composition mapping tables. The resolution and auto-fill panels (Figures 3.4 and 3.5 respectively) are the same as in fusion and allow the modeler to select control options for the composition mapping tables. At this stage setup has been completed and the modeler can start resolving naming conflicts and assigning equivalencies within the submodels using the composition mapping tables.

4.3.2 Resolution

The first composition mapping table displayed is the compartment mapping table (Figure 4.6), followed by the species and the function mapping tables (Figures 4.8 and 4.9 respectively). While the layout and structure of the composition mapping tables is the same as the fusion mapping tables, how they are used is far different. Consider the initial compartment mapping table (Figure 4.6), all the models (the composed model and the three submodels) contain a compartment called $cell$. The use of color (gray and white) indicates the originating submodel for a component added to the composed model, therefore the $cell$ compartment in the $Composed\ Model$ was derived from $Submodel\ I$. In addition, links are automatically created between components on the same row, in this case from components in the composed model to their corresponding components within the submodels. After the modeler has set up the compartment mapping table to the state in Figure 4.6, the composed model will contain three submodels, one compartment and three links (Figure 4.7).

The other mapping tables are similar to the ones encountered in Section 3.2.2. The initial unit, reaction and parameter mapping tables for composition are provided for more detail (Figures 4.10, 4.11, and 4.12 respectively). It should be noted that the initial reaction mapping table contains no reactions in the composed

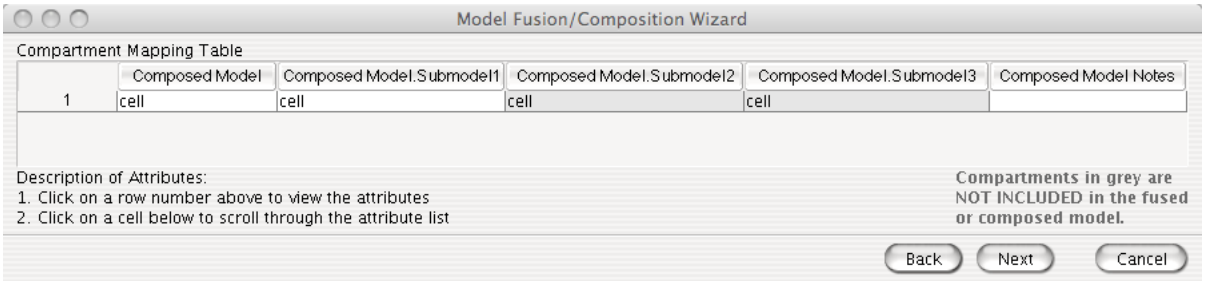


Figure 4.6: Composition Wizard compartment mapping table.

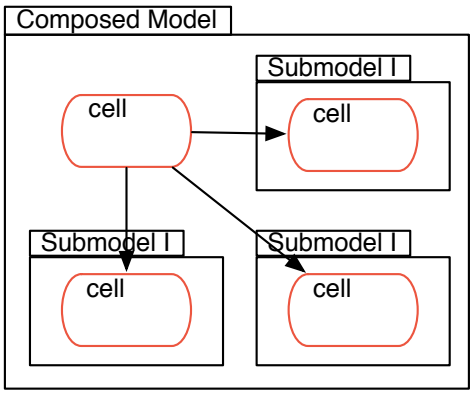


Figure 4.7: Conceptual representation of the Composed Model from Figure 4.6.

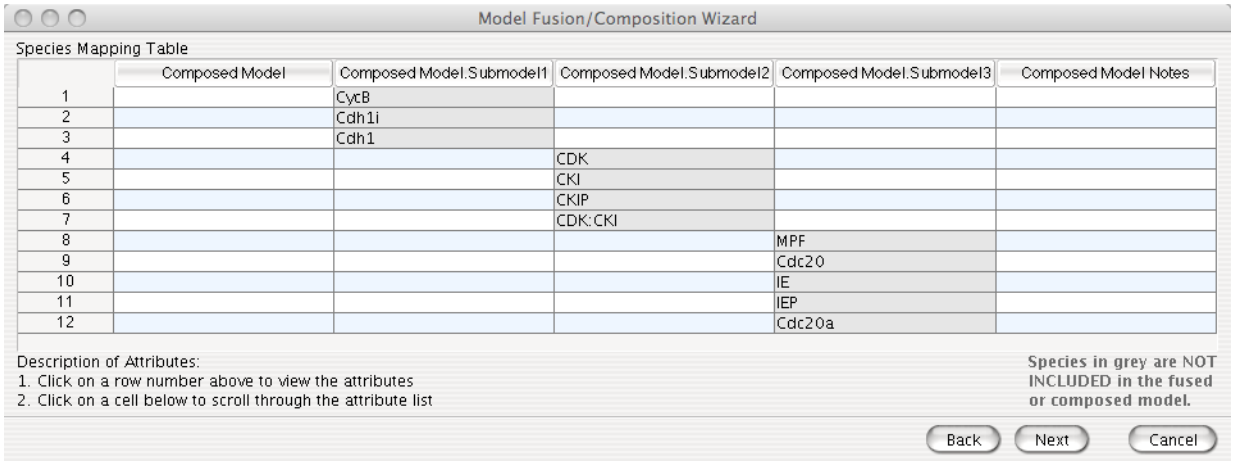


Figure 4.8: Composition Wizard species mapping table.

model. Section 4.4 will provide a more complete explanation of this and will include a walkthrough of the composition process with a relevant example.

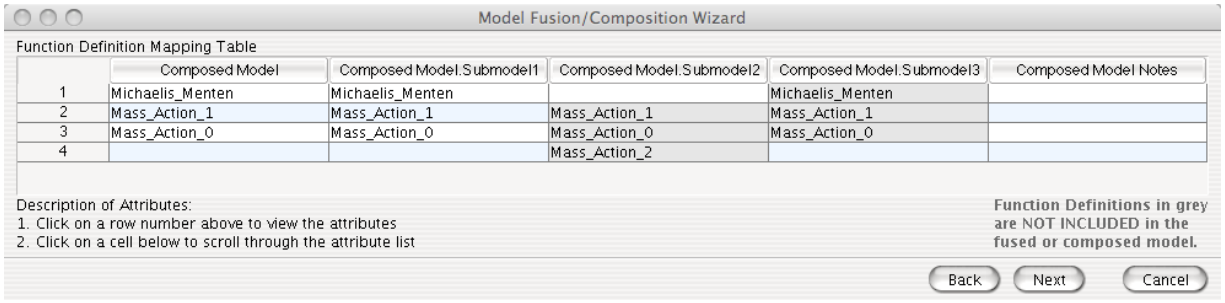


Figure 4.9: Composition Wizard function mapping table.

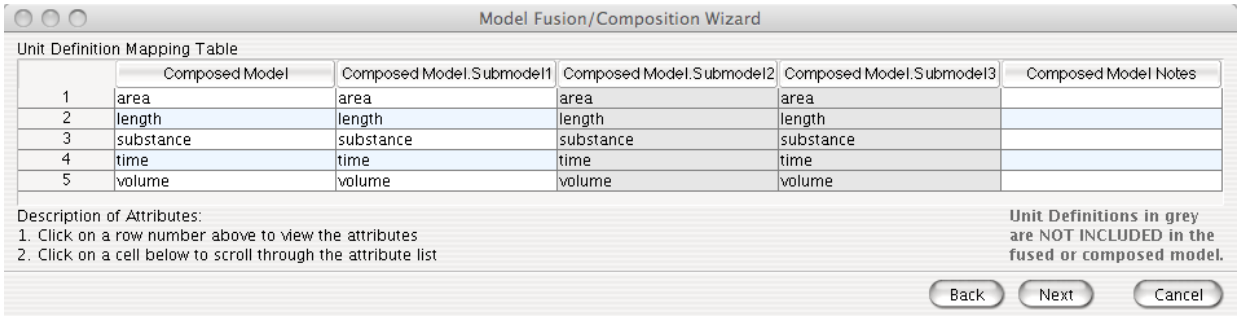


Figure 4.10: Composition Wizard unit mapping table.

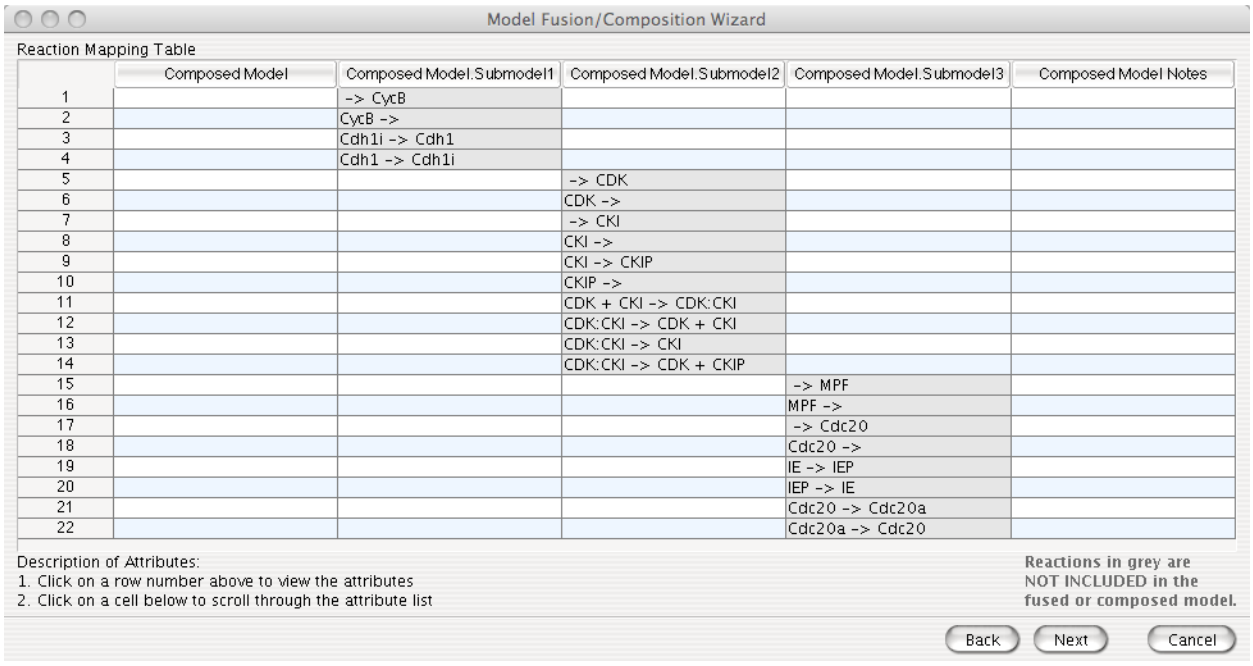


Figure 4.11: Composition Wizard reaction mapping table.

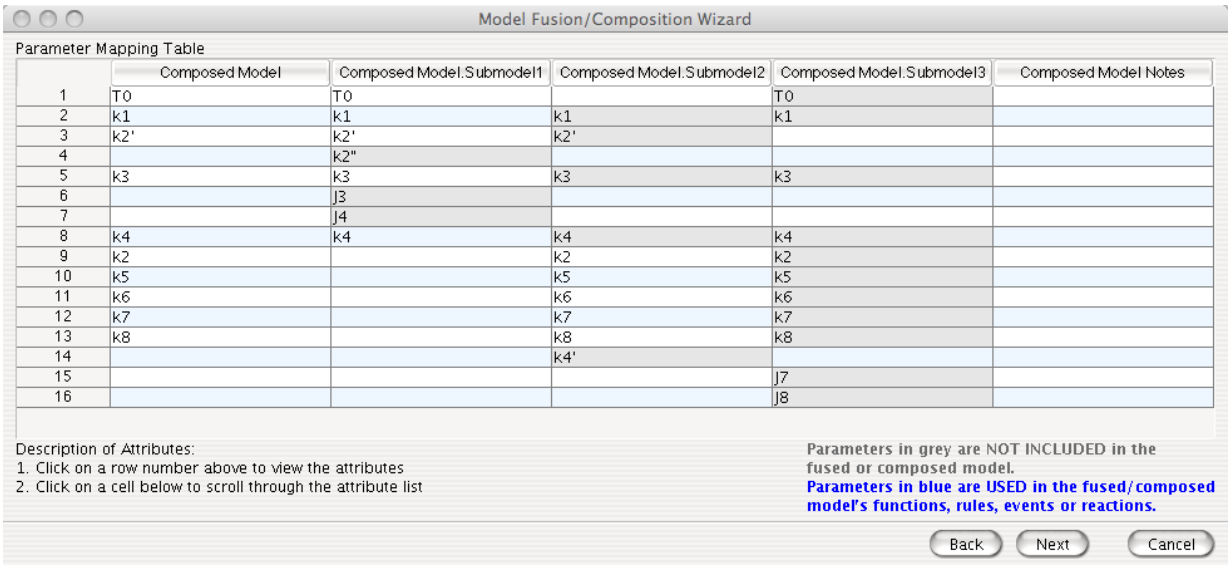


Figure 4.12: Composition Wizard parameter mapping table.

4.3.3 Linking Mechanism

The Composition Wizard resolves components based on their names and automatically adds and links the intersection of components within the submodels to the composed model (described in more detail in Section 4.4). The Composition Wizard restricts the linking mechanism to ensure components can only be linked together level-by-level and not across many levels. For example, suppose model X contains a submodel Y which itself contains another submodel Z. Both model X and Z contain the same species S. Then $X.S$ cannot be linked directly to $Z.S$ as there is more than a single level of distance between the two models in the composition hierarchy. In this case the Composition Wizard automatically adds a new species S to model Y and creates two links, one from $X.S$ to $Y.S$ and the other from $Y.S$ to $Z.S$. This mechanism ensures that components can be indirectly linked across any number of levels in a composition hierarchy and prevents the linking of components not defined in the namespace they are used in.

Since components in the submodels are also included in the resulting model when the composed model is flattened, deleting components must be done explicitly. Unlike the Fusion Wizard where components that are not needed in the fused model are simply not added to the fused column, the Composition Wizard behaves somewhat differently. Components that are not required in the composed model must be explicitly deleted using a delete flag in the composed column, called $DELETE\_FROM\_SUBMODEL$. This creates a link with an empty <from> reference, indicating the <to> reference will be marked for deletion (when the composed model is flattened). An example is provided in Figure 4.13, where two species will be explicitly deleted from the submodels they occur in when the composed model is flattened. After a modeler has selected the $DELETE\_FROM\_SUBMODEL$ option in Figure 4.13 two delete links are created (links with empty <from> references), one to SBF in $submodelA$ and the other to SBF in $submodelB$.

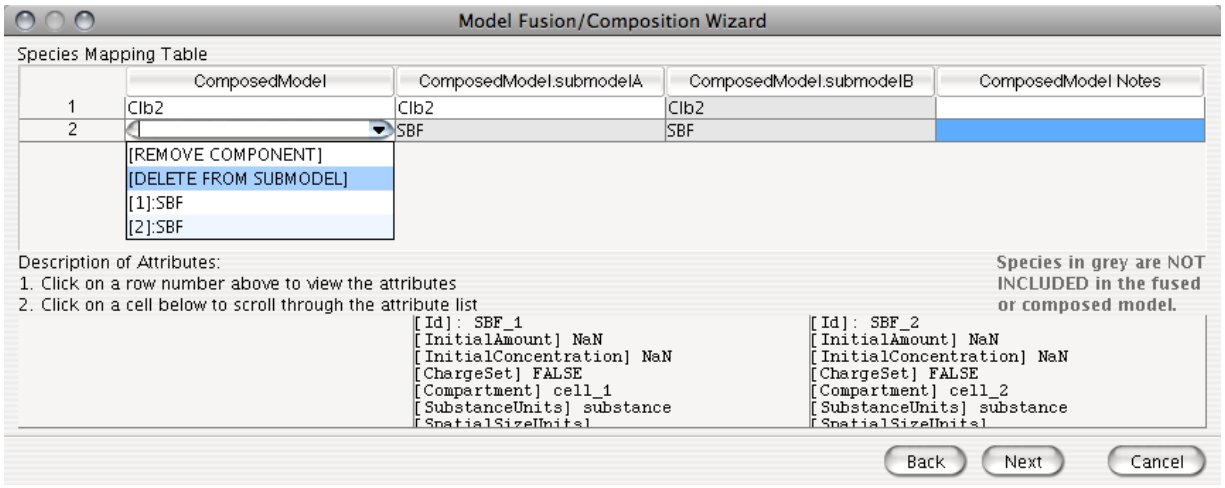


Figure 4.13: Composition Wizard delete mechanism to explicitly delete a component from a submodel.

4.3.4 Composing

Once resolution has been completed and and all naming conflicts are resolved, the modeler can select what to do with their results from a variety of options (Figure 3.19). In order to save the composed model (with its added SBML constructs) the option to "Save the composed model to file" must be selected.

4.3.5 Model Verification

The fusion process described in Chapter 3 defines a series of steps taken to merge two or more models together. This series of steps can be viewed as an "audit trail" used in generating the necessary mapping tables. Precisely this same information can be used to describe the set of instructions needed to connect and link the submodels for composition. Fusion on the one hand, and composition-then-flattening on the other, each produce the same series of transformations on the original collection of models. Thus, both composition and fusion produce the same results. This can be ensured in two ways, first, comparing the simulation results of the fused and flattened version of the composed model and second, fusion and composition are equivalent processes that take the same information to create larger models, the process therefore guarantees the same set of transforms are used to create the fused and composed models. By restricting the type of information required to generate the fused and composed forms we can say with some certainty that both fusion and composition will indeed produce the same results.

4.4 Biological Example

We now use the Composition Wizard to compose together the three sample models from Section 3.3 to create a $Composed\ Model$ (Figure 4.14). Species, reactions and modifier effects in red in the figure are new components that have been added after completing the composition process.

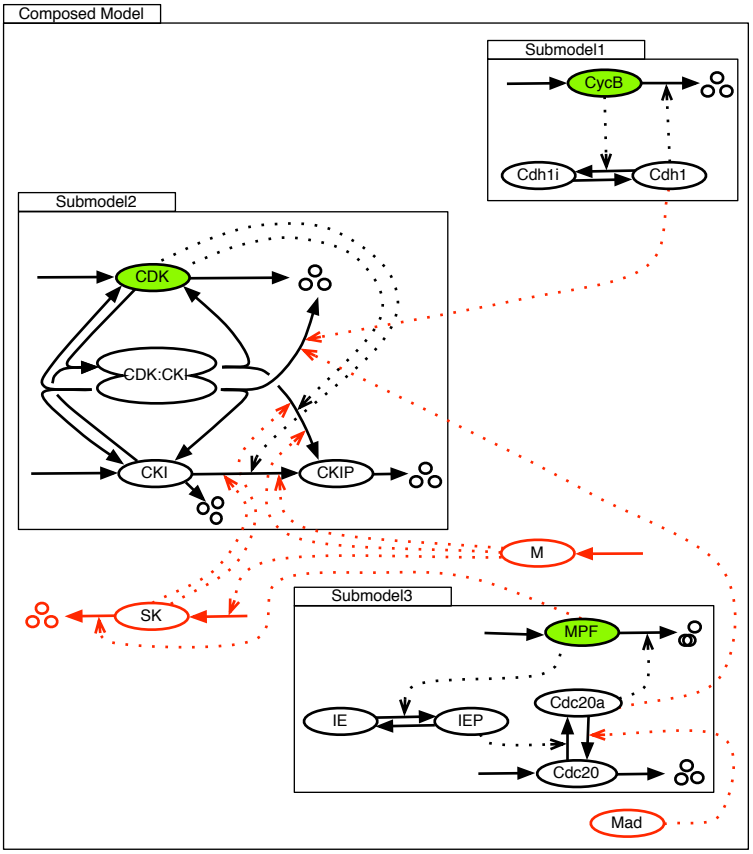


Figure 4.14: Conceptual representation of the composition of Submodels I-III. Colored species indicate equivalences across submodels. Components in red were added to the model after the composition process was completed.

The *Composed Model* shown in Figure 4.14 is only a conceptual version, showing how the user envisions the composed model but not how the computer represents it. The *Composed Model* produced by the Composition Wizard is better represented by Figure 4.15, which contains duplicated species, reactions and parameters (not shown in the Figure 4.14). In both figures, color is used to indicate equivalent species that have been linked together across submodels. Construction of the *Composed Model* requires multiple rounds of both model construction (using the JigCell ModelBuilder [75]) and composition (using the Composition Wizard). The three steps needed to create the *Composed Model* are as follows:

1. *Submodels I-III* are loaded into the Composition Wizard and initial mapping tables for species and reactions are produced. From these mapping tables, the user generates an intermediate composed model.
2. The intermediate composed model is opened in the ModelBuilder and additional species and reactions are added (shown in red in Figure 4.15).
3. The intermediate composed model is opened in the Composition Wizard and a second round of name resolution occurs, producing the final mapping tables.

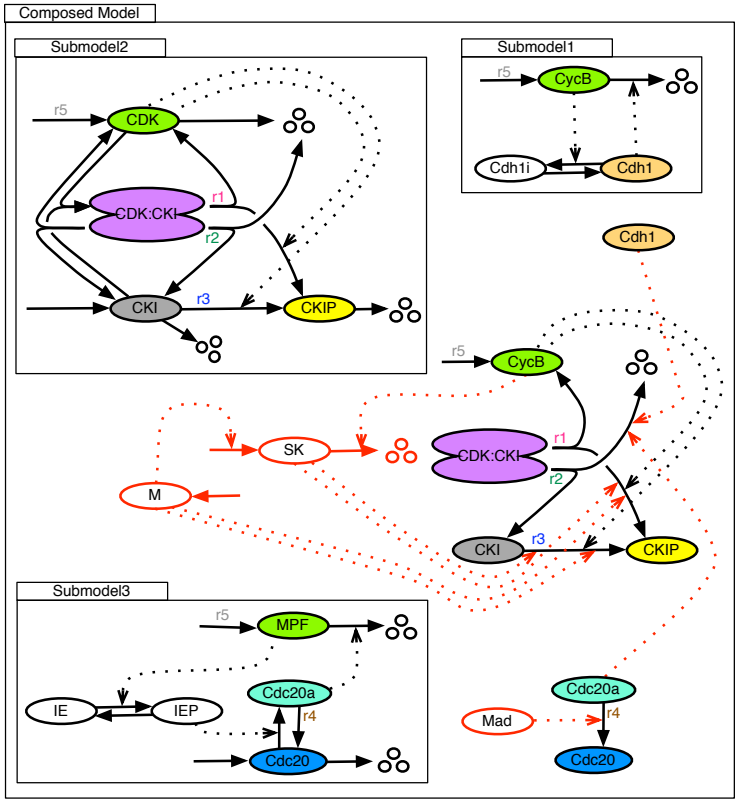


Figure 4.15: Actual representation of the composition of Submodels I-III by the Composition Wizard and the JigCell ModelBuilder. Colored species and numbered reactions ($r1$-$r5$) indicate equivalences across submodels. For example while there are four reactions labelled $r5$, only one (reaction $r5$ in the top-level composed model) is "used", the others are linked to the top-level reaction and remain "unused". Similarly with species, while the model contains four green species representing $CycB$ only the top-level $CycB$ is used, and the others become placeholders. Flattening this model will produce the model in Figure 3.27.

The Composition Wizard resolves components based on their names and automatically adds and links the intersection of components within the submodels to the composed model. Each (sub)model in this example contains only one compartment, and they are all identified as a single compartment in the composed model in the compartment mapping table. The compartment mapping table can also handle models with more than one compartment. As the cyclin-dependent kinase has been named differently in the three submodels, the first step in the initial species mapping table is to set these species to be equivalent to each other by placing them on the same row. Next the cyclin-dependent kinase is added to the composed model column and called $CycB$. This causes three links to be automatically created: one from $Composed\ Model.CycB$ to $Submodel\ I.CycB$, the second from $Composed\ Model.CycB$ to $Submodel\ II.CDK$, and the third from $Composed\ Model.CycB$ to $Submodel\ II.MPF$. In this way, all the instances of cyclin-dependent kinase are linked together in the initial species mapping table and thus represent the same species (Figure 4.16).

The next step is to create the initial reaction mapping table in Figure 4.17. The five reactions marked

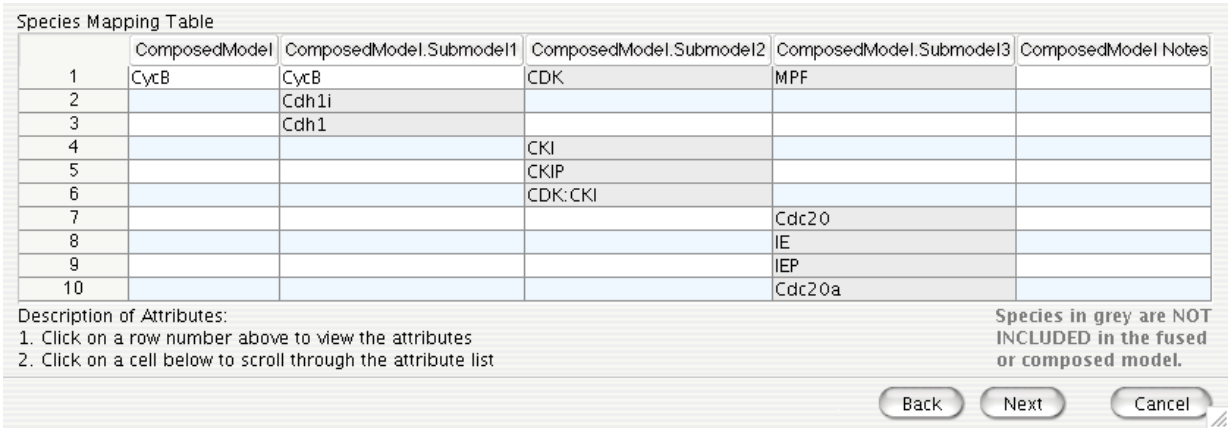


Figure 4.16: Initial species mapping table for the $Composed\ Model$ in the JigCell Composition Wizard.

$r1$-$r5$ in Figure 4.15 are added and linked in the composed model. Reactions ($r1$-$r4$) will be modified as a result of the composition. The $CycB$ synthesis reaction ($r5$) occurs in all submodels with the same rate law and is thus linked to ensure that only one copy of the reaction will be used during simulation or flattening. However the reverse reaction ($CycB$ degradation) is not added or linked in the composed model as it contains different rate laws (and modifier effects) in the three submodels.

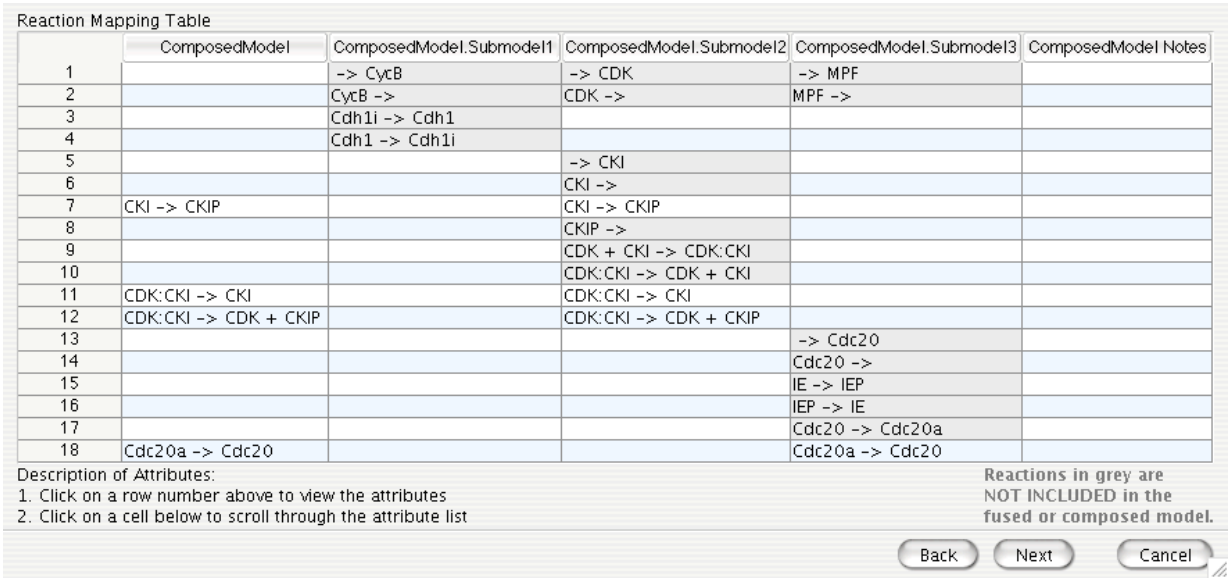
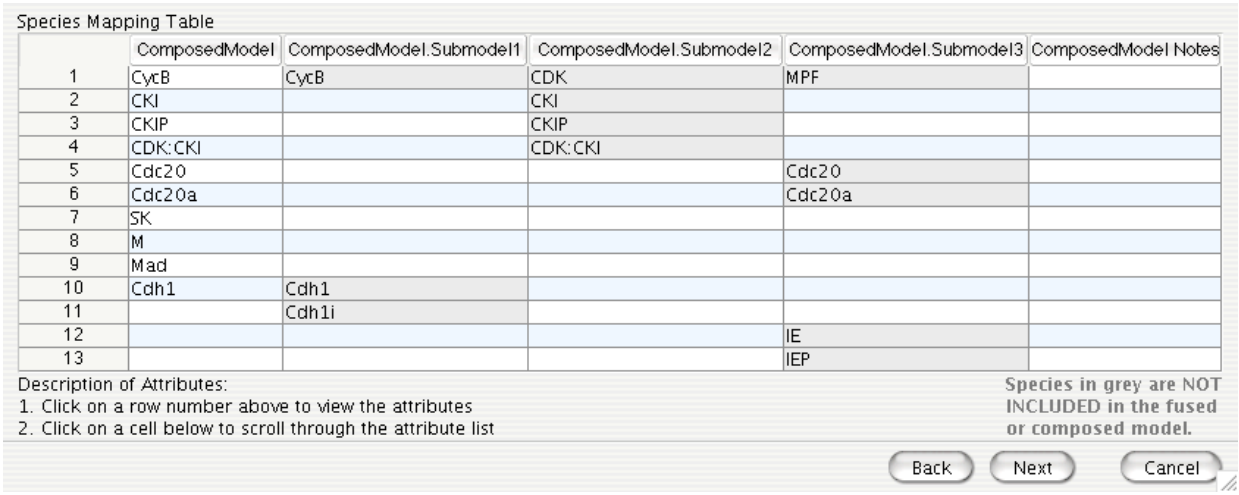


Figure 4.17: Initial reaction mapping table for the $Composed\ Model$.

Once the first round of composition is finished, an intermediate composed model is generated and opened in the JigCell ModelBuilder for editing. Three additional reactions ($\rightarrow SK$, $SK \rightarrow$, and $\rightarrow M$) are added, which correspond to the red reactions in Figure 4.15. SK refers to a "starter kinase" and M represents cell "mass". M increases according to a logistic rate equation, and M is decreased by a factor of 2 each time the

cell divides. Cell division is triggered when $CycB$ drops below a certain threshold, as cyclin B is degraded by $Cdc20$ and $Cdh1$ and is best represented as an event in the model (see [74] for details). Next, the four novel reactions in the composed model ($r1$-$r4$) are updated to reflect their new kinetic laws. This intermediate composed model is saved and loaded into the Composition Wizard. The components are once again resolved, and the final species and reaction mapping tables are produced (Figures 4.18 and 4.19 respectively).



Species Mapping Table

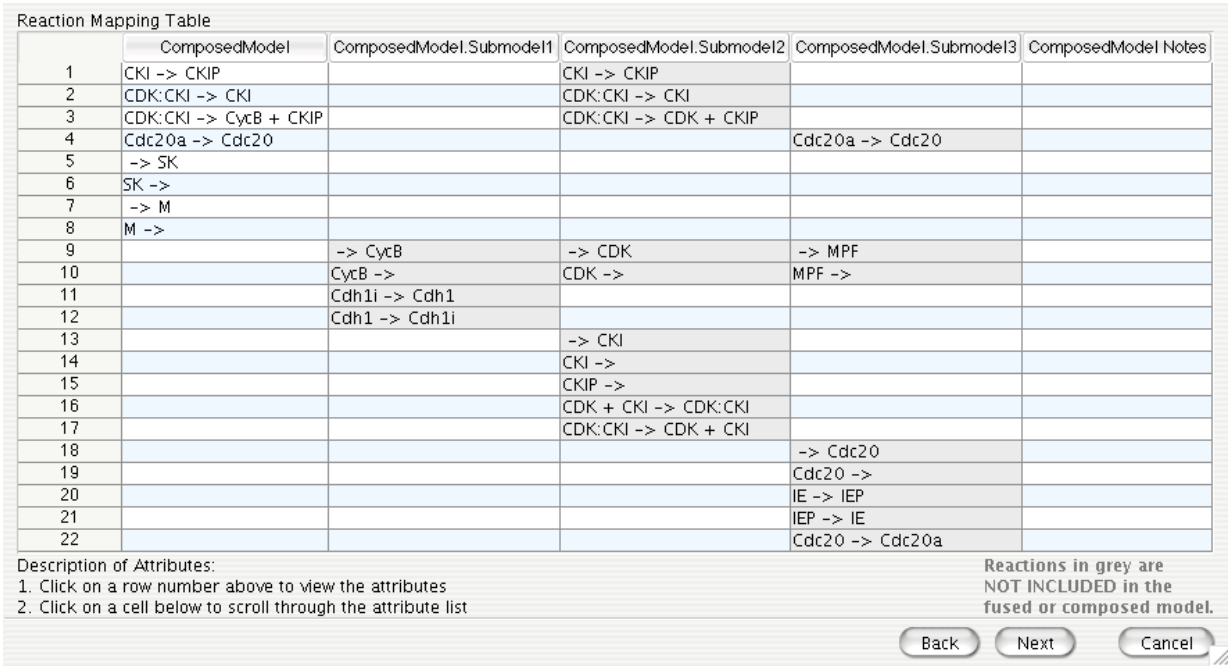
	ComposedModel	ComposedModel.Submodel1	ComposedModel.Submodel2	ComposedModel.Submodel3	ComposedModel Notes
1	CycB	CycB	CDK	MPF	
2	CKI		CKI		
3	CKIP		CKIP		
4	CDK:CKI		CDK:CKI		
5	Cdc20			Cdc20	
6	Cdc20a			Cdc20a	
7	SK				
8	M				
9	Mad				
10	Cdh1	Cdh1			
11		Cdh1i			
12				IE	
13				IEP	

Description of Attributes:
1. Click on a row number above to view the attributes
2. Click on a cell below to scroll through the attribute list

Species in grey are NOT INCLUDED in the fused or composed model.

Back Next Cancel

Figure 4.18: Final species mapping table for the $Composed\ Model$.



Reaction Mapping Table

	ComposedModel	ComposedModel.Submodel1	ComposedModel.Submodel2	ComposedModel.Submodel3	ComposedModel Notes
1	CKI -> CKIP		CKI -> CKIP		
2	CDK:CKI -> CKI		CDK:CKI -> CKI		
3	CDK:CKI -> CycB + CKIP		CDK:CKI -> CDK + CKIP		
4	Cdc20a -> Cdc20			Cdc20a -> Cdc20	
5	-> SK				
6	SK ->				
7	-> M				
8	M ->				
9		-> CycB	-> CDK	-> MPF	
10		CycB ->	CDK ->	MPF ->	
11		Cdh1i -> Cdh1			
12		Cdh1 -> Cdh1i			
13			-> CKI		
14			CKI ->		
15			CKIP ->		
16			CDK + CKI -> CDK:CKI		
17			CDK:CKI -> CDK + CKI		
18				-> Cdc20	
19				Cdc20 ->	
20				IE -> IEP	
21				IEP -> IE	
22				Cdc20 -> Cdc20a	

Description of Attributes:
1. Click on a row number above to view the attributes
2. Click on a cell below to scroll through the attribute list

Reactions in grey are NOT INCLUDED in the fused or composed model.

Back Next Cancel

Figure 4.19: Final reaction mapping table for the $Composed\ Model$.

These steps produce the final composed model (Figure 4.15) which must then be simulated to verify that its dynamic properties represents the observed behavior of growing-dividing yeast cells in expected ways. Since our current simulators require standard SBML (Level 2) input, composed models must be flattened before they can be simulated. This flattening is done by removing the additional constructs used to describe the composition. The JigCell flattening algorithm automates this process and produces a single flat model that can then be sent to a simulator. Simulating the flattened $Composed\ Model$ produces the simulation output shown in Figure 4.20, which closely matches the simulation output from Tyson and Novak's model (Figure 8 in [74]).

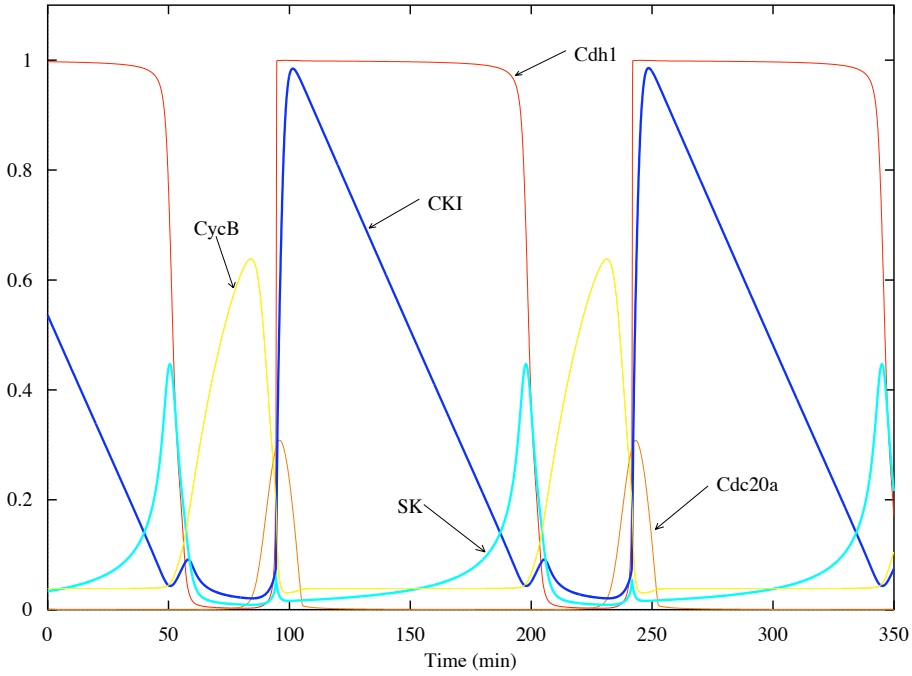


Figure 4.20: Simulation for the $Composed\ Model$ using XPP.

4.5 Model Flattening

Model Flattening automatically and unambiguously converts composed or aggregated models to their fused (or flattened) versions. Flattening takes a model which contains our additional SBML language constructs (i.e. <listOfSubmodels>, <listOfInstances>, <listOfPorts> and <listOfLinks>), and generates a valid SBML Level2 model (i.e., with our language constructs removed). The flattening process is done automatically, using the information provided by the composition or aggregation glue to perform the steps that otherwise are done by hand when a modeler fuses models. The purpose of flattening is to generate a standard SBML file (Level2) from our modified file, for the purpose of running simulations, performing analysis, etc.

4.6 Flattening Process

This section describes the process by which composed or aggregated models can be flattened. Flattening has three steps: separation, saving and resolution. During separation, submodels are separated based on the information within the <listOfSubmodels> and <listOfInstances> structures. Then components are separated one at a time in the same order as described for fusion. The information required to fuse models together is encoded by our added SBML language features used for composition and aggregation. This information is translated to create a single flattened model. Once this is done the connections/links between models is saved for reference during flattening. Finally, during the resolution step the components of the submodels are sorted, separated and assigned based on the model (or submodel) they originated from. The resolution step in flattening is similar to the resolution stage in fusion, except here it is done automatically rather than by the modeler.

4.6.1 Flattening Algorithm

The JigCell Flattening Algorithm (Table 4.1) is a modified breath first search algorithm and was implemented as a standalone application capable of being run from the command line or called from another JigCell application such as the ModelBuilder. For example, if a composed model is opened in the ModelBuilder the modeler is presented with the following choice (Figure 4.21):

1. If a "Yes" option is selected the Flattening Algorithm is called and once it automatically flattens the composed model the resulting flattened model is displayed in the ModelBuilder.
2. If a "No" option is selected, the ModelBuilder will open the composed model and treat it as a single model and only display its compartments, species, parameters, functions, events, units, and reactions ignoring it's submodels, instances and links. It should be noted that saving the model will also save the composition information (instances, submodels, and links), as this information is stored by the program even though its not displayed. This option is useful for adding components to the top level model while going through multiple rounds of model construction and composition.

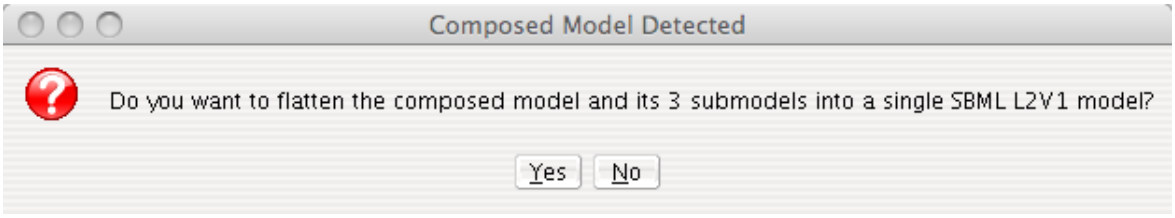


Figure 4.21: ModelBuilder dialog box that detects a composed model and queries modeler whether the composed model from Figure 4.15 should be flattened.

To better understand the flattening algorithm consider the example in Figure 4.22. The composed model $Model1\_Composed$ contains two species A and B, and two submodels ($Model2$ and $Model3$). Both the

Table 4.1: Algorithm for flattening a hierarchy of models starting from the model **Root**.

```
FlattenModels (ModelTree* Root) {
    Queue ModelTree* pending;
    Links linklist == null;
    pending.enqueue(Root);
    while pending ≠ ∅ {
        m ← pending.dequeue();
        for each Link in m.links
            ...addlink only adds Link if it is not already in linklist...
            linklist.addlink (Link);
        for each Species in m.species
            ... addspecies only adds Species if it is not alread in specieslist...
            specieslist.addspecies (Species);
        for each Children of m
            pending.enqueue (Child);
    }
}
```

submodels also contain two species (A and C in $Model2$ and A and D in $Model3$), in addition $Model3$ is itself a composed model. Species A in $Model1\_Composed$, $Model2$, and $Model3$ represents the same biological entity (and are thus colored the same). The three instances of species A are linked together with two links, from $Model1\_Composed.A$ to $Model2.A$ and $Model1\_Composed.A$ to $Model3.A$. Similarly species D in $Model3$ is linked to it's equivalent species in $Model4$ and $Model5$. When the composed model is flattened automatically using the flattening algorithm (Table 4.1), the resulting model ($Model1\_Flattened$) contains six species. Note only one instance of species A and D is present in $Model1\_Flattened$. Due to how the links are set up in $Model1\_Composed$, species A is actually $Model1\_Composed.A$ and species D is actually $Model3.D$. The flattened model ($Model1\_Flattened$) in Figure 4.22 in constructed in the following steps:

1. Species A and B from $Model1\_Composed$ are added to $Model1\_Flattened$
2. The two links in $Model1\_Composed$ ($Model1\_Composed.A \rightarrow Model2.A$ and $Model1\_Composed.A \rightarrow Model3.A$) are parsed. This results in deleting the two species in the <to> fields of the <link> structures ($Model2.A$ and $Model3.A$ respectively).
3. The remaining species of $Model2$ and $Model3$ are added to $Model1\_Flattened$.
4. While parsing $Model3$, it is detected as a composed model and the above steps are repeated.
5. The two links in $Model3$ ($Model3.D \rightarrow Model4.D$ and $Model3.D \rightarrow Model5.D$) are parsed. This results in deleting the two species in the <to> fields of the <link> structures ($Model4.D$ and $Model5.D$ respectively).
6. Finally, the remaining species of $Model4$ and $Model5$ are added to $Model1\_Flattened$.

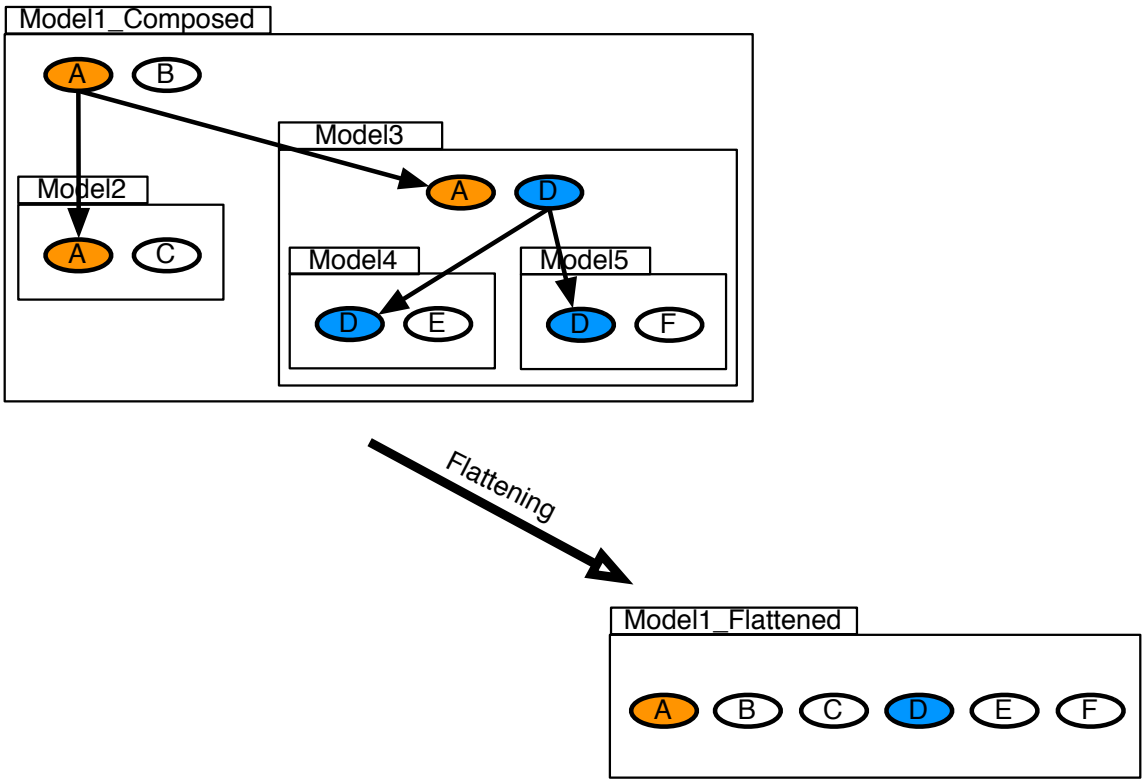


Figure 4.22: Visual representation of the flattening algorithm showing a composed model containing only species (A-F) and its corresponding flattened model. Color indicates biological equivalences and the solid arrows indicate <link> structures between species.

4.6.2 Uses

Flattening a composed or aggregated model has a number of advantages and disadvantages. The biggest advantage of flattening a model is that it allows us to use existing simulation tools, which have no support for composition or aggregation. We could think of model flattening as analogous to compiling a program written in a high-level programming language into the machine code ready to execute on a computer. In addition to simulating composed or aggregated models, flattening also allows modelers the use of a multitude of SBML-compliant tools for composed or aggregated models. Converting a composed or aggregated model into a flattened model removes redundancies as equivalent model entities (e.g. <species>, <parameters>, etc.) need only be declared once in the resulting flattened model. Removing redundancies decreases the size of the model while ensuring that no information is lost. Modelers have the option of flattening existing composed or aggregated models in situations where they would like to perform further composition or aggregation using non-hierarchical models, instead of using the composed or aggregated models as submodels. This option allows modelers to avoid having to deal with the additional complexity of having a separate and distinct hierarchy within their submodels during composition. It is left up to the modeler to decide if this increases or decreases clarity.

The drawbacks to flattening include losing the explicit description of relationships between entities within submodels. Information about individual submodels and their characteristics will be replaced by a fused model which contains no implicit history of how it was constructed. The flattened model might be too large and verbose to comprehend, and therefore to manage in practice. It is intended that models will be saved and maintained as composed or aggregated models and will only be flattening if the modeler needs to perform some form of analysis (such as simulation) on the model.

Chapter 5

Model Aggregation

Real molecular networks seem to be built up from simpler modules that carry out specific tasks and can be hooked together [73]. By allowing modelers to substitute an aggregate for groups of reactions, and enabling aggregated modules to be connected to one another, we envision that model construction will become faster and more intuitive while holding true to the apparent structure of the organism being modeled. Modularization is defined here as the process of grouping reactions together as a single entity (a module) with a defined set of inputs and outputs (called ports). A module is not a simplification of the group of reactions (or their behavior). It is purely representational and is used to aid better understanding of how parts of the model (the modules) interact with each other. Aggregation is the process of connecting modules together (by linking outputs of one module to inputs of another) in order to create a larger model (an aggregate of modules).

This chapter introduces the process by which models are aggregated together. Section 5.1 describes the SBML language features needed to describe model aggregation. Section 5.2 describes how to create an aggregate model from a collection of pre-existing modules. Section 5.3 describes the prototype software developed to implement the ideas in this chapter. Finally, Section 5.4 describes the aggregation process with an example that models the protein interaction network controlling cell division.

Contents

5.1 SBML Syntax

We implemented aggregation by means of new language features for SBML. All the language features previously identified for composition (Section 4.1) are also used for aggregation. In addition we added new language features to describe the connections between modules. The features both define the hierarchy of the modules and represent the interactions, relationships, and links between the modules. Aggregation, fusion and composition should produce the same results, in that the simulation output of all three forms of a model should be identical. While fusion combines submodels together in an irreversible way, aggregation (and compostion) simply reference module components by defining the "glue" that holds the modules together. A major difference is that in fusion, the explicit description of relationships between entities within submodels is lost, while aggregation (and compostion) keeps a record of how models were aggregated (or composed) together.

Like composition, aggregation is also an iterative process, as models are usually constructed in increments, with modelers switching back and forth between adding components (or modules) to a model and fine tuning models through simulations. Constructing aggregated models is a bottom-up process as smaller models are first aggregated together to create larger models which in turn can be used to create even more complex models, as previously demonstrated in [68]. An aggregated model can be created from a combination of flat and/or aggregated models. Model aggregation generates an aggregation graph that describes the relations among the various modules. Changing the connections between modules in an aggregated model results in a different aggregation graph structure. Since model aggregation is a combination of constructing modules and generating aggregation graphs, a tool for aggregation should support the iterative nature of the process.

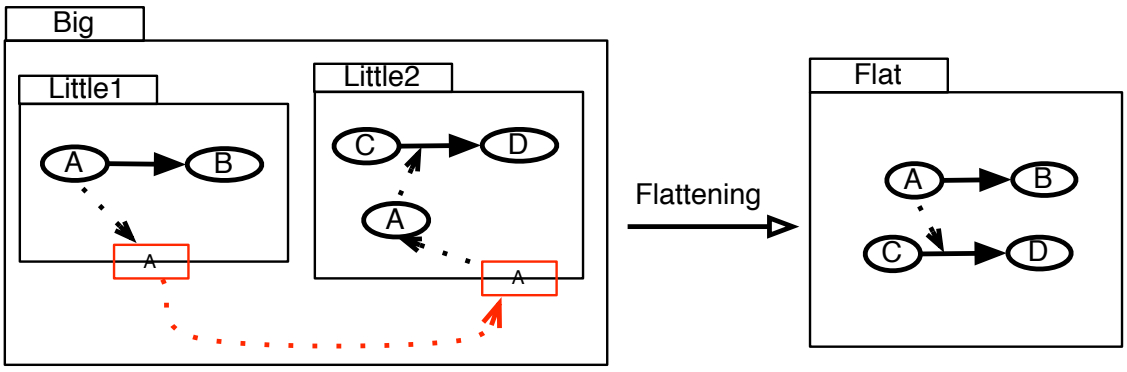


Figure 5.1: Aggregated model showing a link between two ports in different modules and the corresponding flattened model.

To illustrate the SBML language features needed to describe model aggregation, consider the example in Figure 5.1, which shows an aggregated model (Big) and its corresponding flattened model ($Flat$). Model Big contains two modules called $Little1$ and $Little2$. The SBML code to describe the two models $Little1$ and $Little2$ in Figure 5.1 is:

```
<model name="Little1">
```

```
  <listOfSpecies>
     <species id="A_1" name="A"/>
     <species id="B_1" name="B"/>
  </listOfSpecies>
  <listOfParameters>
    <parameter id="k1_1" name="k1"/>
  </listOfParameters>
  <listOfReactions>
    <reaction id="reaction1">
    <listOfReactants>
      <speciesReference species="A_1"/>
    </listOfReactants>
    <listOfProducts>
      <speciesReference species="B_1"/>
      </listOfProducts>
      <kineticLaw>
        <math:math>
          <math:ci>k1_1</math:ci>
        </math:math>
      </kineticLaw>
    </reaction>
  </listOfReactions>
</model>

<model name="Little2">
  <listOfSpecies>
     <species id="A_1" name="A"/>
     <species id="C_1" name="C"/>
     <species id="D_1" name="D"/>
  </listOfSpecies>
  <listOfParameters>
    <parameter id="k2_1" name="k2"/>
  </listOfParameters>
  <listOfReactions>
    <reaction id="reaction2">
      <listOfReactants>
        <speciesReference species="C_1"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="D_1"/>
      </listOfProducts>
      <listOfModifiers >
```

```
        <modifierSpeciesReference
           species="A_1"/>
      </listOfModifiers >
      <kineticLaw>
        <math:math>
          <math:apply>
            <math:times/>
              <math:ci>k2_1</math:ci>
              <math:ci>A_1</math:ci>
          </math:apply>
        </math:math>
      </kineticLaw>
    </reaction>
  </listOfReactions>
</model>
```

5.1.1 Ports

To begin aggregation we must first convert the (sub)models into modules by defining their ports. The <port> structure (enclosed in a <listOfPorts> structure) gives a modeler access to a particular species, parameter or another port within a module for aggregation. A <port> structure is composed of three attributes: id, $target$, and $mutable$. The id field gives a unique identifier to the port structure. The $target$ field specifies a single species, parameter, or another port by its SBML identifier. The boolean $mutable$ field specifies whether the port is an input or output port (false for output and true for input). The syntax for an output port structure is as follows:

```
<listOfPorts>
  <port id="Port_Id"
    target="species|parameter|port id"
    mutable="false"/>
</listOfPorts>
```

Input and output ports are distinguished from each other by their target type. <Port> structures are used in conjunction with the other language constructs described below.

5.1.2 Submodels and Instances

Once we have created the modules, we are ready to add them to the aggregated model which can contain one or more modules. A module (or submodel) is just an SBML model (enclosed within an SBML <model> structure), described in Section 4.1.2. After the list of modules has been declared and included in the aggregated model (Big), the modeler needs to instantiate the modules to use or access them through the

<instance> structure, described in Section 4.1.3. The naming convention adopted in Section 4.1.1 is also used for aggregation.

5.1.3 Links

Ports are connected together using the <link> structure described in Section 4.1.4. In aggregation, a <link> structure connects two ports in separate modules of an aggregated model. Linking components in aggregation can be achieved by connecting two port structures (and is described in more detail in Section 5.3). The SBML code to describe the connections between modules in Figure 5.1 is:

```
<model id="Big">
   <listOfSubmodels>
      <model id="Little1">
         ...
         <listOfPorts>
            <port id="A" target="A_1"
               mutable="true"/>
         </listOfPorts>
      </model>
      <model id="Little2">
         ...
         <listOfPorts>
            <port id="A" target="A_2"
               mutable="true"/>
         </listOfPorts>
      </model>
   </listOfSubmodels>

   <listOfInstances>
      <instance
         id="Little1_instance"
         xlink:type="simple"
         xlink:href="#xpointer
         (/sbml/model/listOfSubmodels/model[@id=%22Little1%22])"/>
      <instance
         id="Little2_instance"
         xlink:type="simple"
         xlink:href="#xpointer
         (/sbml/model/listOfSubmodels/model[@id=%22Little2%22])"/>
   </listOfInstances>

   <listOfLinks>
```

```
      <link>
         <from object="Little1_instance">
            <subobject object="A_1"/>
         <to object="Little2_instance">
            <subobject object="A_2"/>
         </to>
      </link>
   </listOfLinks>
</model>
```

In summary, the <listOfSubmodels> and <listOfInstances> structures are used to define the layout of the aggregated model. Connections between the modules within the layout are made using the <link> structures which can only connect <port> structures to each other. The four SBML language features discussed above are sufficient to describe all aspects of model aggregation.

5.2 Aggregation Process

The fundamental differences between aggregation and composition include:

1. the amount of access to model information
2. the initial source of these components and
3. new interactions cannot be made from the submodels used to create the aggregated model (new modules with the new interactions must be created and connected in the aggregated model instead).

The basic building blocks permitted for composition and aggregation are the same – in both cases, a building block is one or more reactions. However, in composition, a component's information is not hidden from other components. A modeler can link to any variable or component within a submodel. In aggregation, model information is deliberately hidden to control complexity, and a modeler can only link to variables or components that are explicitly made visible (through ports) in a given module. Components for composition are typically pre-existing models that thus might contain redundancies between components and were not necessarily created with the intent of combining with other components. In contrast, aggregation modules are designed to be connected.

A major benefit of aggregation is that the modeler does not need to know the details about what is within a module in order to use it, only its list of input and output ports. Constraints exist on what can be defined as an input or output port to a module. Inputs are parameters (with fixed values) or species participating as modifiers in reactions, while outputs are species (any reactant or product in any reaction within the module could be defined as an output). These constraints ensure that a consistent set of differential equations will always be produced from a network of modules.

5.3 Aggregation Connector

We now explain in detail how to generate an aggregated model by connecting together modules through their ports using the JigCell Aggregation Connector. The Aggregation Connector in Figure 5.2 has two functions:

1. it converts (sub)models to modules by specifying their ports, and
2. it connects modules together to create aggregated models.

The user interface is divided into two parts, an aggregation window (top) which allows modelers to connect modules together graphically, and an embedded JigCell ModelBuilder [75] (bottom) which displays a selected module using JigCell's standard spreadsheet interface.

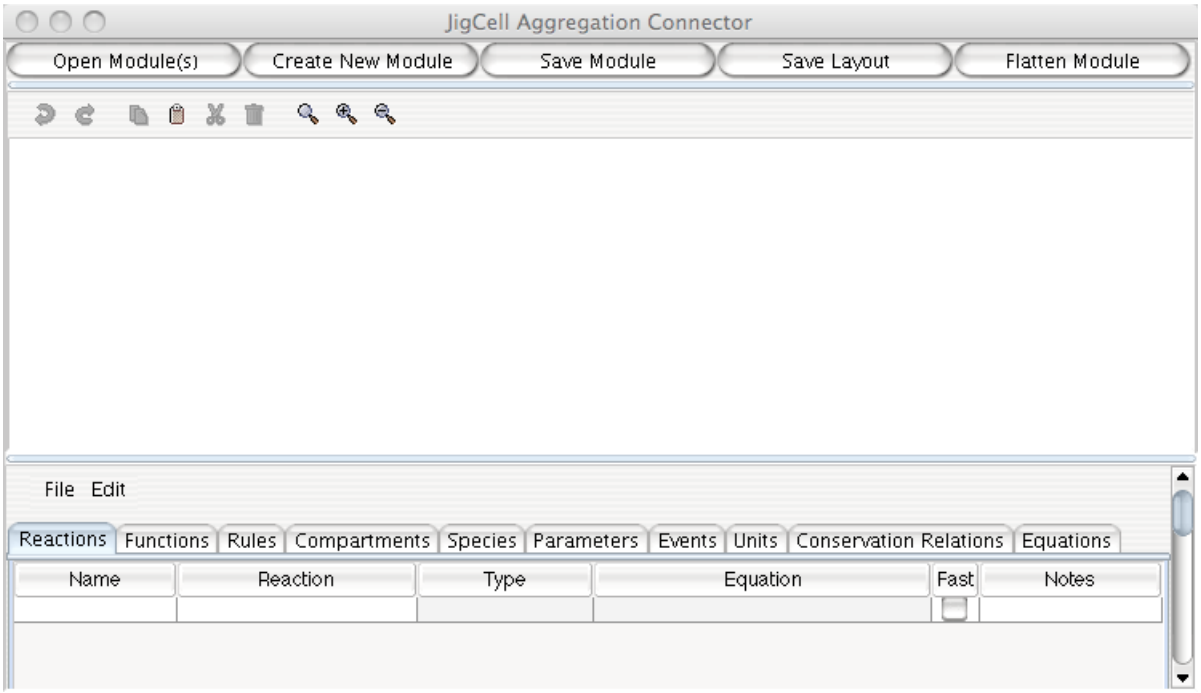


Figure 5.2: Screen shot of the JigCell Aggregation Connector.

The options shown in the top bar in Figure 5.2 are:

1. Open Module(s) - Displays a file chooser to enable the modeler to select what SBML model files to open from the filesystem.
2. Create New Module - Takes the existing connections between models displayed in the aggregation window and creates a new aggregated model.
3. Save Module - Saves the aggregated model displayed in the aggregation window as an SBML file.
4. Save Layout - Saves the layout information of the figure displayed in the aggregation window only (the SBML file is not saved in this case).
5. Flatten Module - Flattens the aggregated model and prompts a user with a save dialog box in order to save the flattened model in an SBML file.

To begin aggregation, the modeler presses the "Open Module(s)" button on the top left of the application window to select SBML models from a file chooser. These models can be both submodels (without defined ports) or modules (with defined ports). Modules are displayed 5.3 as boxes with inputs and outputs in the aggregation window, while submodels are only displayed as boxes without ports until such time as the modeler defines its inputs and outputs. Each module has the following components:

1. a name in the middle of the box
2. an optional set of input ports on the left hand side of the box
3. an optional set of output ports on the right hand side of the box
4. a box with a "+" sign in the top-left hand corner of each box, when clicked displays the module in the embedded ModelBuilder.

To see the complete model (reactions, species, etc) or convert a model to a module (by defining the ports), the modeler loads it into the embedded ModelBuilder. The ModelBuilder displays the module's components in a spreadsheet interface. The ModelBuilder has been extended to enable a modeler to select which species and parameters are defined as ports by adding an additional column to the species and parameters spreadsheets, respectively. After loading two or more modules onto the aggregation window, a modeler can link the output ports of one module to the input ports of another module to create an aggregated model. The links between modules are indicated as solid black lines in Figure 5.3.

Once the aggregated model is completed it can be saved in an SBML file, flattened into a standard SBML (Level 2) model, or be converted into a more complex module for use in future aggregation. When converting an existing aggregate model into a new module, the Aggregation Connector performs an initial best-guess for the ports. Since an output port can connect to more than one input port, all the output ports of the modules connected together in the aggregate model are made into output ports in the newly created module. Input ports on the other hand, can only be connected once, so only those input ports that are unconnected are exposed in the newly created module as input ports.

5.4 Biological Example

To illustrate how aggregation is implemented we again follow the approach used by Tyson and Novak when building their basic model of cell cycle control in yeast cells [74]. They built their model in stages starting from a simple model and then adding new pieces until they obtained a satisfactory representation of the cell cycle control system. Note that the sample models used here are slightly different from those used for fusion and composition. The models used for aggregation are customized (by removing redundancies) and information within them is restricted (using ports) in order to easily and quickly connect them together. The first model (which we will call $Module\ I$) deals with the interaction between the cyclin B-dependent kinase ($CycB$) and a cyclin-dependent kinase inhibitor (CKI), as shown in Figure 5.4. $Module\ II$ deals with the antagonistic interactions between $CycB$ and a cyclin B-degrading factor ($Cdh1$), as shown in Figure 5.5. $Module\ III$ deals the interaction between $CycB$ and a different form of the cyclin-degrading factor ($Cdc20$),

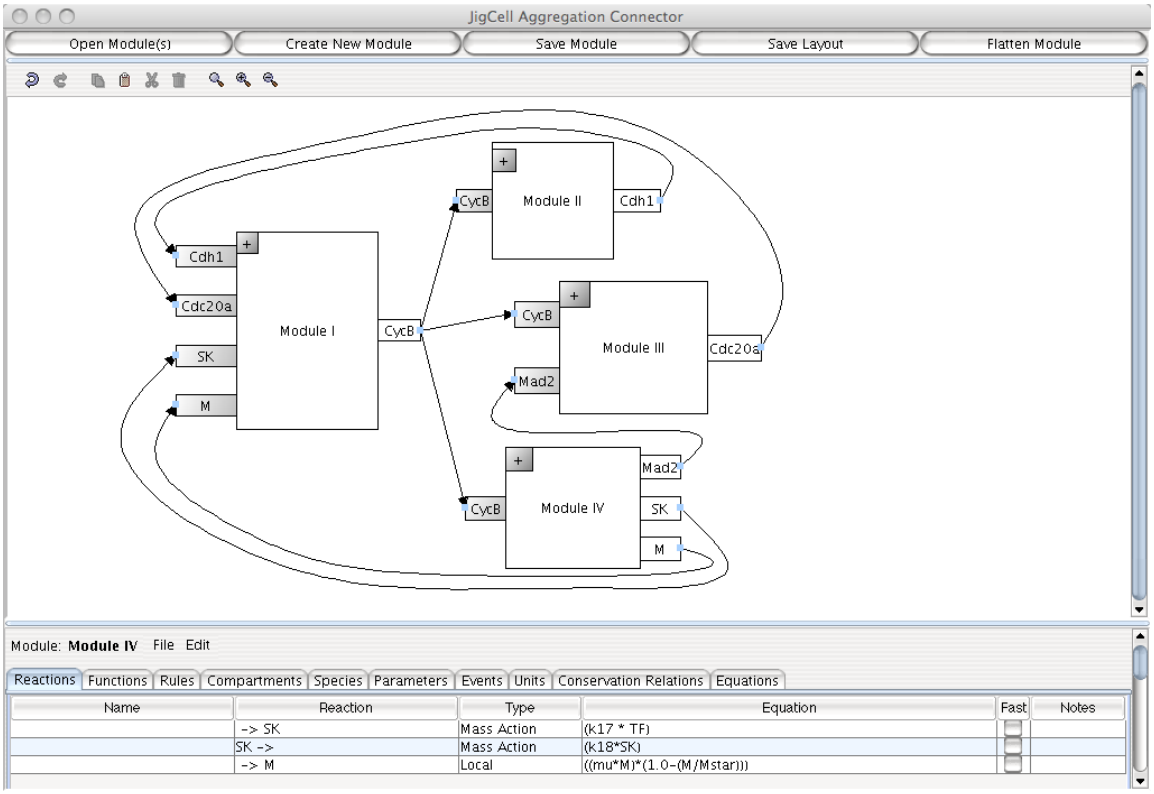
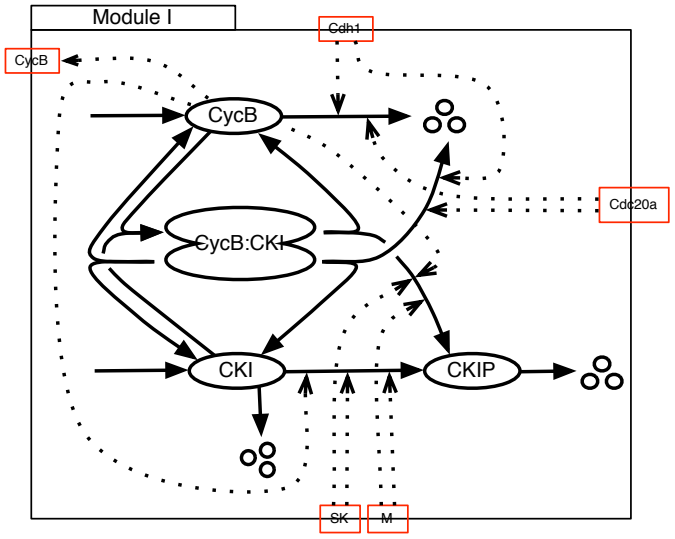


Figure 5.3: Screen shot for the $Aggregated\ Model$ in the JigCell Aggregation Connector.

as shown in Figure 5.6. Finally, $Module\ IV$ contains four additional reactions (synthesis and degradation of SK and M respectively), as shown in Figure 5.7. SK refers to a "starter kinase" and M represents cell "mass". M increases according to a logistic rate equation, and M is decreased by a factor of 2 each time the cell divides. Cell division is triggered when $CycB$ drops below a certain threshold, as $CycB$ is degraded by $Cdc20$ and $Cdh1$. See [74] for details. It should be noted that $Module\ IV$ is treated as a separate and distinct model for aggregation, but was added after fusion and composition were completed in the previous examples. This is because we are able to customize models to remove redundancies in aggregation.

The four models can be combined using the fusion process to create a $Fused\ Model$ (Figure 3.27) or customized and linked together in the Aggregation Connector to create an $Aggregated\ Model$ (Figure 5.3). The four steps needed to create the $Aggregate\ Model$ are as follows:

1. $Modules\ I$-IV are created with predefined ports in either the standard ModelBuilder or the version of the ModelBuilder embedded within the Aggregation Connector. For convenience, the ports are set for each module during the construction process but could just as easily be set or adjusted in the Aggregation Connector once the modules have been created.
2. $Modules\ I$-IV are loaded into the Aggregation Connector.
3. Links between ports are created by holding down the mouse button and dragging the mouse from an



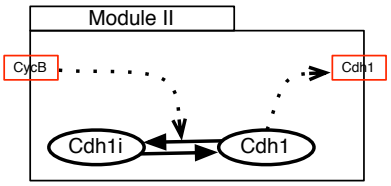
JigCell ModelBuilder – Module1.sbml

ModelBuilder File Edit

Reactions | Functions | Rules | Compartments | Species | Parameters | Events | Units | Conservation Relations | Equations

Name	Reaction	Type	Equation	Fast	Notes
	-> CycB	Mass Action	k1		
	CycB ->	Mass Action	(k2 + k2' * Cdc20a + k2" * Cdh1)*CycB		
	-> CKI	Mass Action	k3		
	CKI ->	Mass Action	k4*CKI		
	CKI -> CKIP	Mass Action	(k5 + k5' * SK + k5" * M * CycB)*CKI		
	CKIP ->	Mass Action	k6*CKIP		
	CycB + CKI -> CycB:CKI	Mass Action	(k7*CycB)*CKI		
	CycB:CKI -> CycB + CKI	Mass Action	k8*CycB:CKI		
	CycB:CKI -> CKI	Mass Action	(k2 + k2' * Cdc20a + k2"Cdh1)*CycB:CKI		
	CycB:CKI -> CycB + CKIP	Mass Action	(k5 + k5' * SK + k5" * M * CycB)*CycB:CKI		

Figure 5.4: Module I wiring diagram and reaction definitions in the JigCell ModelBuilder



JigCell ModelBuilder – Module2.sbml

ModelBuilder File Edit

Reactions | Functions | Rules | Compartments | Species | Parameters | Events | Units | Conservation Relations | Equations

Name	Reaction	Type	Equation	Fast	Notes
	Cdh1i -> Cdh1	Michaelis-Menten	((k10*Cdh1i)*1.0)/(J10+Cdh1i)		
	Cdh1 -> Cdh1i	Michaelis-Menten	(((k9 * CycB)*Cdh1)*1.0)/(J9+Cdh1)		

Figure 5.5: Module II

output port of one module to an input port of another module.

4. Once all the links have been created the model may be saved as an SBML file with additional language

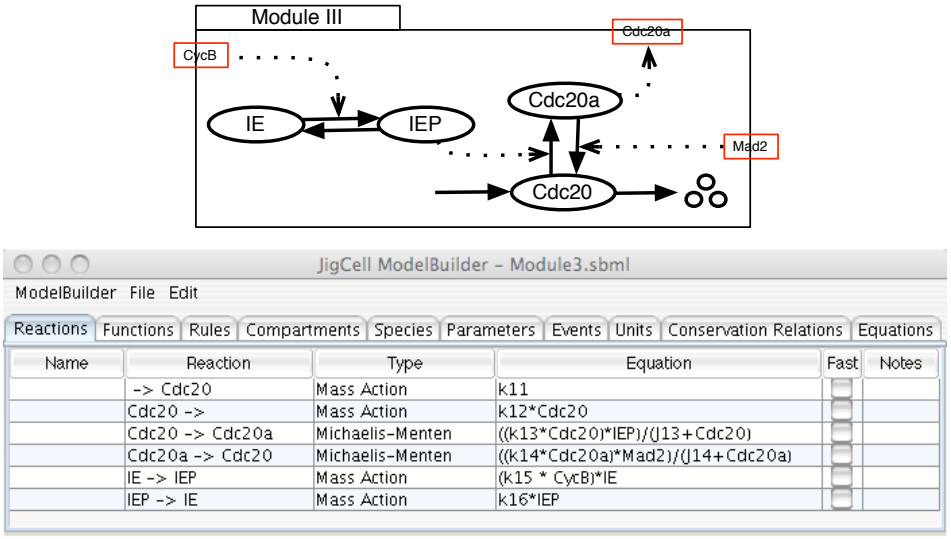


Figure 5.6: Module III

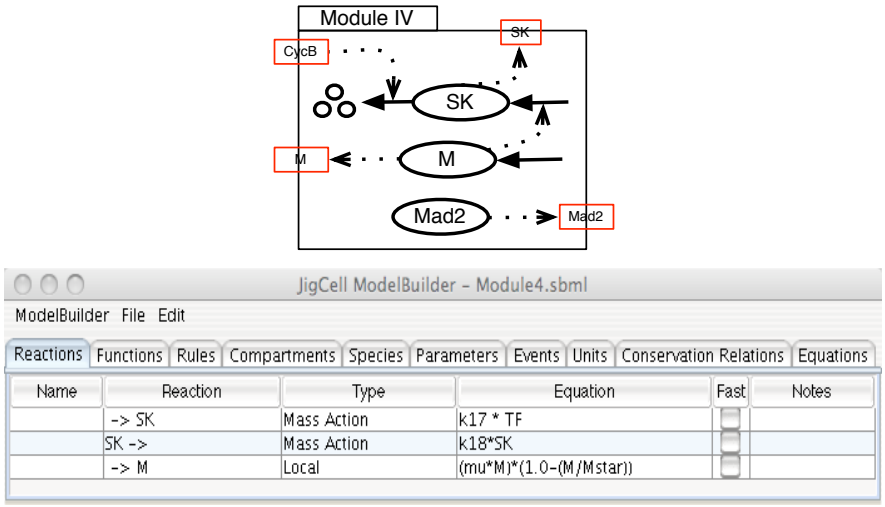


Figure 5.7: Module IV

features.

These steps produce the *Aggregate Model* in Figure 5.3 which must then be simulated to verify that its dynamic properties represents the observed behavior of growing-dividing yeast cells in expected ways. Since our current simulators require standard SBML (Level 2) input, aggregated models must be flattened before they can be simulated. This flattening is done by removing the additional constructs used to describe the aggregation. The JigCell flattening algorithm automates this process and produces a single flat model that can then be sent to a simulator. Simulating the flattened *Aggregated Model* on XPP produces the simulation output shown in Figure 5.8, which closely matches the simulation output from Tyson and Novak's model

(Figure 8 in [74]).

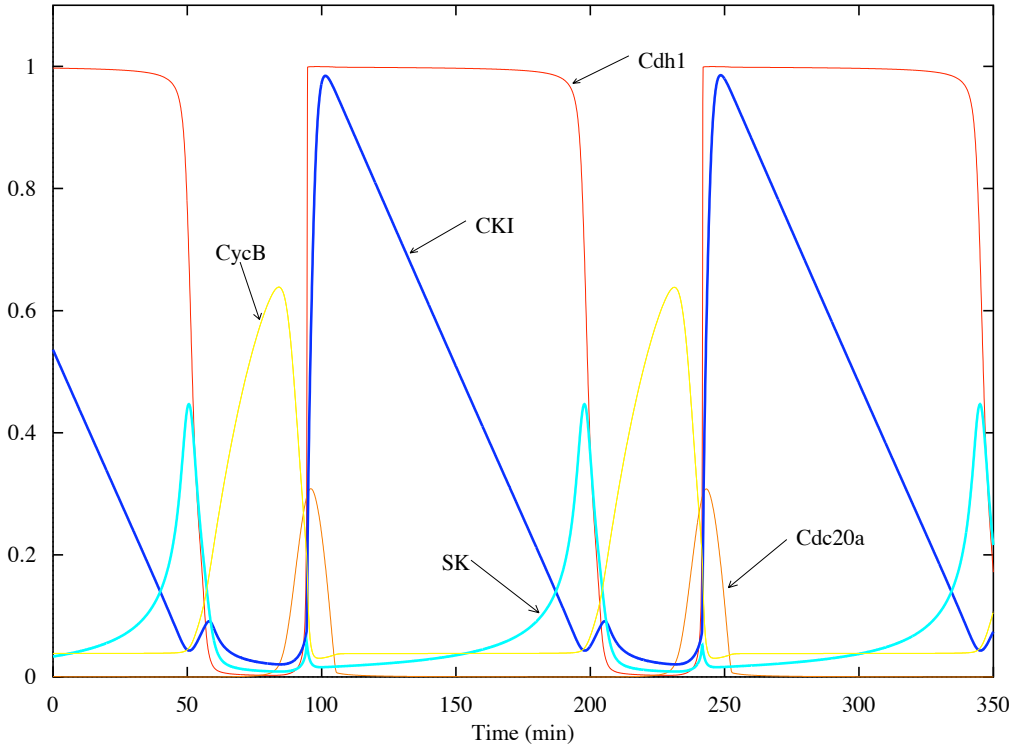


Figure 5.8: Simulation for the $Agggregated\ Model$ using XPP.

Chapter 6

Case Study: Combining Two Components For A Budding Yeast Model

Current modeling efforts by expert modelers (Kathy Chen and Suman Banik) in John Tyson's Group at Virginia Tech involve challenging issues in large-scale modeling. One such effort is focused on the morphogenesis checkpoint in budding yeast. Ciliberto et al. [12] developed a model of the morphogenesis checkpoint that was "hooked up" to a very primitive cell cycle engine in budding yeast. The aim of this case study is to successfully combine the morphogenesis checkpoint model with the full cell cycle engine model proposed by Chen et al. [11] using fusion, composition and aggregation. When simulated, both Chen's cell cycle engine model and Ciliberto's morphogenesis checkpoint model are able to describe certain distinct aspects of the cell cycle in budding yeast. When combined, it is envisaged that the resulting model will be able to not only describe the behavior of the individual models but also new behavior that couldn't be described previously.

This chapter applies the processes, tools, and frameworks presented in this dissertation to a large-scale biological modeling problem. Section 6.1 introduces the biology behind the cell cycle model. Section 6.2 introduces the biology behind the morphogenesis checkpoint model. Sections 6.3, 6.4, and 6.5 describe the steps an expert modeler takes to combine two these models using model fusion, model composition, and model aggregation, respectively. Finally, Section 6.6 summarizes our experiences and results.

Contents

6.1 Cell Cycle Model

The budding yeast cell cycle consists of four distinct phases: G1, S, G2, and M. The G1, S, and G2 phases are collectively known as interphase, while the M phase is composed of karyokinesis (where chromosomes divide) and cytokinesis (where cytoplasm divides forming two daughter cells). A newly born cell starts in the G1 phase (the first phase in interphase), which lasts from the end of the previous M phase till the beginning of DNA synthesis. During this phase the cell acquires resources and grows until there are suitable conditions for replication. After reaching a viable size for replication, the cell transitions to the S phase. During the S phase, the cell synthesizes a new copy of its DNA. After the completion of DNA synthesis, the cell continues to grow until it reaches a mass of approximately twice its birth size. This is known as the G2 phase in the standard cell cycle. However, the cell soon reaches the end of G2 and begins mitosis, which is the most physically complex portion of the cell cycle.

Mitosis, also called M phase, is the process of nuclear and cell division. By the end of mitosis, the cell will produce a complete new copy of itself. At the start of mitosis, the nuclear membrane usually breaks down (it should be noted that this does not occur in budding yeast) and a mitotic spindle forms. Then the cell enters metaphase, in which the duplicated chromosomes align themselves along the mitotic spindle. The cell next passes through anaphase and enters telophase. During this period, the pairs of chromosomes separate and move to the ends of the mitotic spindle, where new nucleuses are forming. Finally, the cell pinches until the mass of the cell divides. After division, the two cells are again in the G1 phase.

Chen et al. [11] previously developed a model of the budding yeast cell cycle. Figure 6.1 shows their wiring diagram for the biochemical reaction network that regulates DNA synthesis, bud emergence, and mitosis in budding yeast. Chen simplified the wiring diagram by omitting portions of the model and using abstractions. For example, a text description of a function (e.g. “Budding”) stands in the place of that part of the model and several species were combined together representing more that one biological entity. $Cln2$ in the model represents both $Cln1$ and $Cln2$ in the cell. $Clb5$ represents both $Clb5$ and $Clb6$, and $Clb2$ represents both $Clb1$ and $Clb2$. Although Chen does not formalize or describe these abstractions in detail, understanding the abstractions is essential to understanding the model.

Figures 6.2, 6.3, and 6.4 show this same budding yeast model in the JigCell ModelBuilder. The version of the budding yeast model in the ModelBuilder consists of 97 chemical reaction equations. The ModelBuilder translates the biochemical reaction network into a system of 39 differential equations.

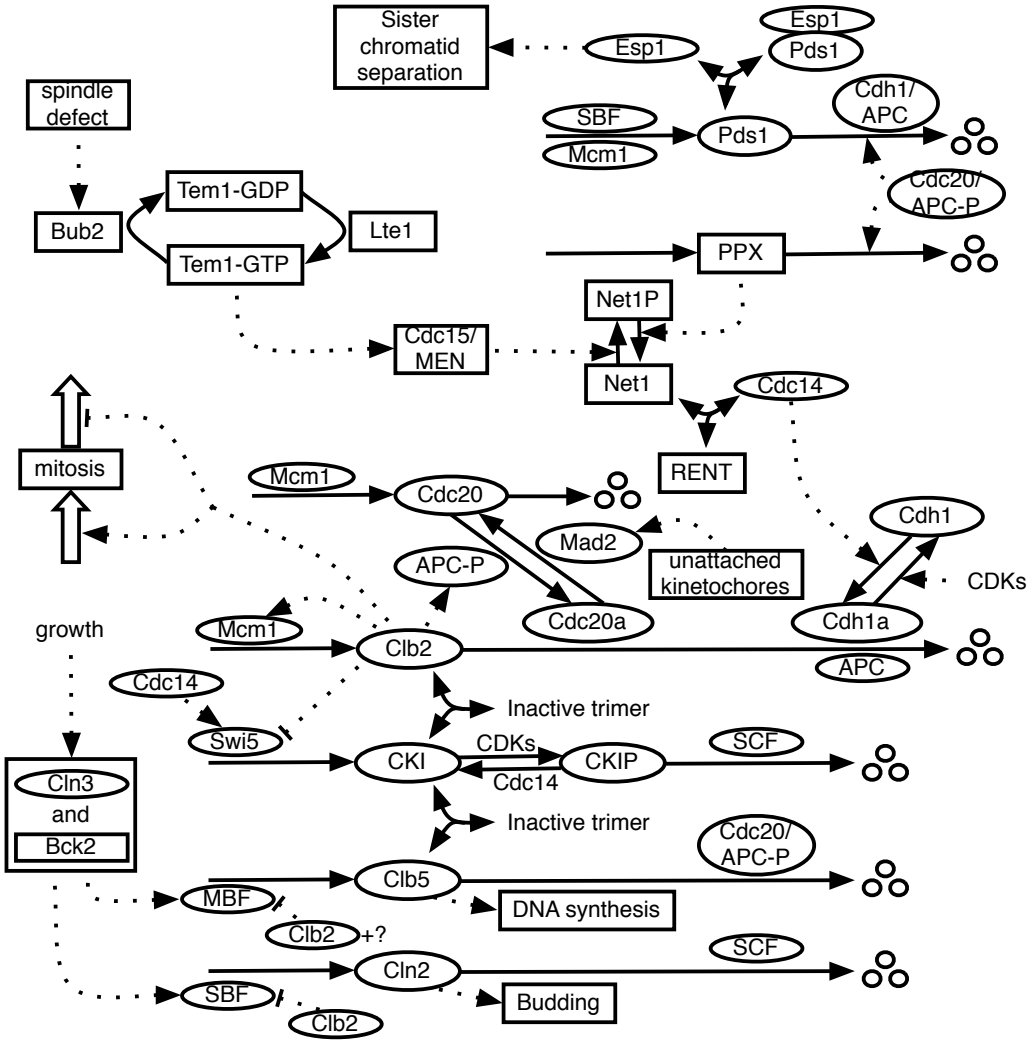
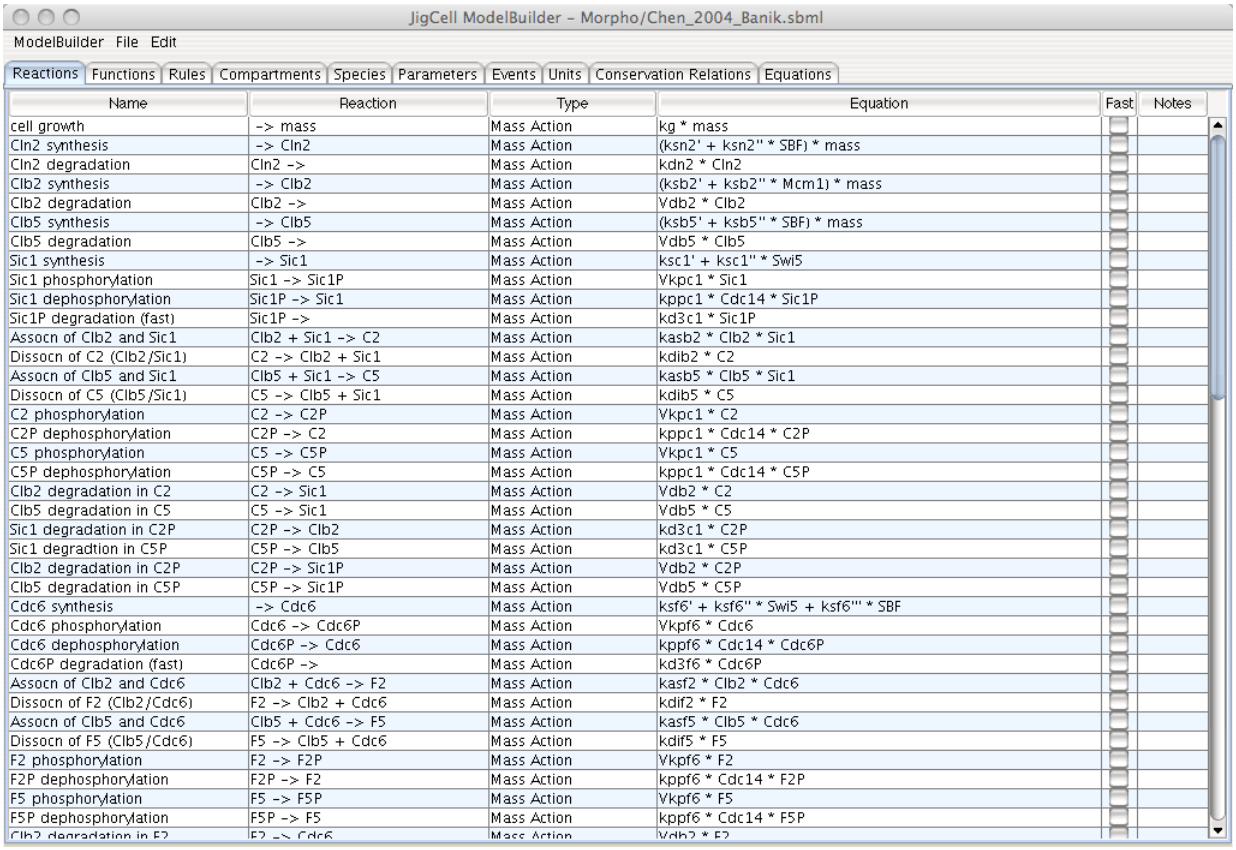
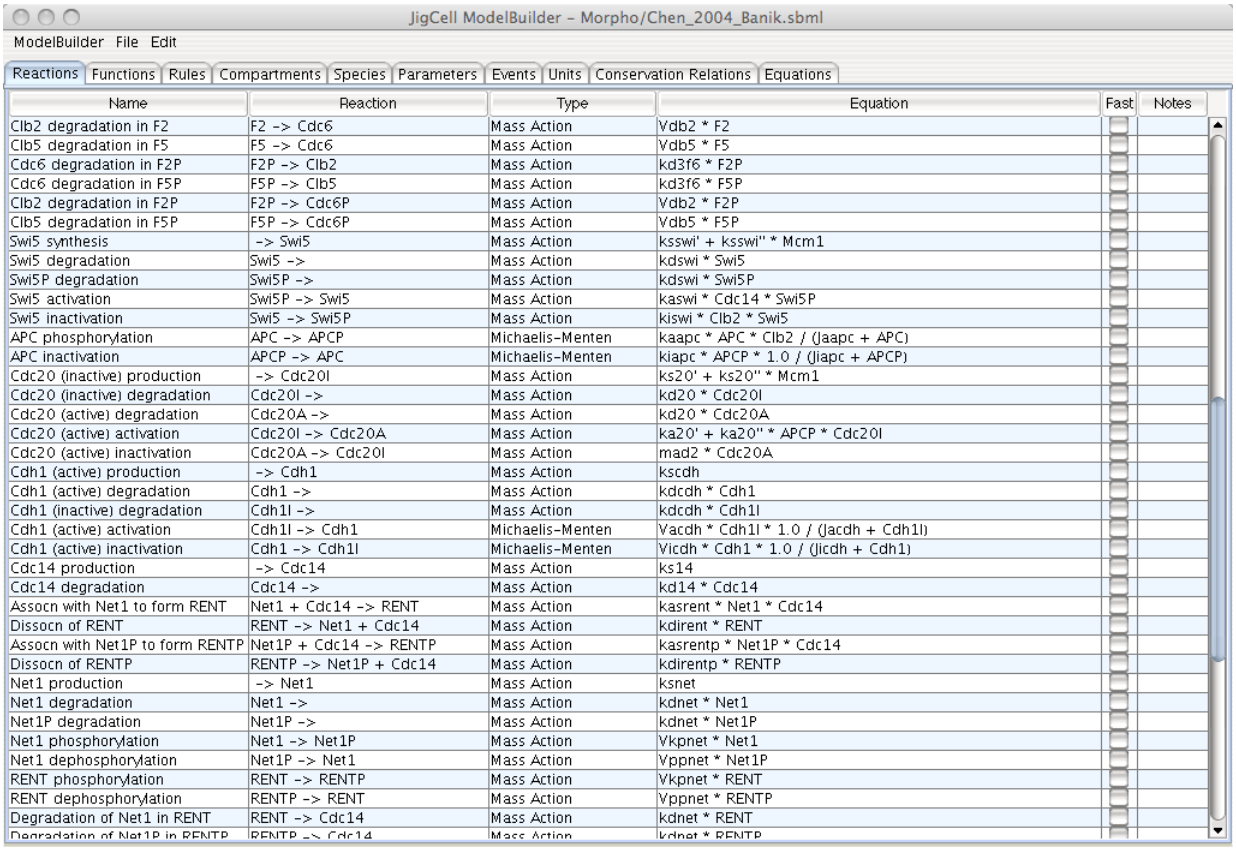


Figure 6.1: Wiring diagram for the budding yeast model by Chen.



Name	Reaction	Type	Equation	Fast	Notes
cell growth	-> mass	Mass Action	kg * mass		
Cln2 synthesis	-> Cln2	Mass Action	(ksn2' + ksn2'' * SBF) * mass		
Cln2 degradation	Cln2 ->	Mass Action	kdn2 * Cln2		
Clb2 synthesis	-> Clb2	Mass Action	(ksb2' + ksb2'' * Mcm1) * mass		
Clb2 degradation	Clb2 ->	Mass Action	Vdb2 * Clb2		
Clb5 synthesis	-> Clb5	Mass Action	(ksb5' + ksb5'' * SBF) * mass		
Clb5 degradation	Clb5 ->	Mass Action	Vdb5 * Clb5		
Sic1 synthesis	-> Sic1	Mass Action	ksc1' + ksc1'' * Swi5		
Sic1 phosphorylation	Sic1 -> Sic1P	Mass Action	Vkpc1 * Sic1		
Sic1 dephosphorylation	Sic1P -> Sic1	Mass Action	kppc1 * Cdc14 * Sic1P		
Sic1P degradation (fast)	Sic1P ->	Mass Action	kd3c1 * Sic1P		
Assocn of Clb2 and Sic1	Clb2 + Sic1 -> C2	Mass Action	kasb2 * Clb2 * Sic1		
Dissocn of C2 (Clb2/Sic1)	C2 -> Clb2 + Sic1	Mass Action	kdib2 * C2		
Assocn of Clb5 and Sic1	Clb5 + Sic1 -> C5	Mass Action	kasb5 * Clb5 * Sic1		
Dissocn of C5 (Clb5/Sic1)	C5 -> Clb5 + Sic1	Mass Action	kdib5 * C5		
C2 phosphorylation	C2 -> C2P	Mass Action	Vkpc1 * C2		
C2P dephosphorylation	C2P -> C2	Mass Action	kppc1 * Cdc14 * C2P		
C5 phosphorylation	C5 -> C5P	Mass Action	Vkpc1 * C5		
C5P dephosphorylation	C5P -> C5	Mass Action	kppc1 * Cdc14 * C5P		
Clb2 degradation in C2	C2 -> Sic1	Mass Action	Vdb2 * C2		
Clb5 degradation in C5	C5 -> Sic1	Mass Action	Vdb5 * C5		
Sic1 degradation in C2P	C2P -> Clb2	Mass Action	kd3c1 * C2P		
Sic1 degradtion in C5P	C5P -> Clb5	Mass Action	kd3c1 * C5P		
Clb2 degradation in C2P	C2P -> Sic1P	Mass Action	Vdb2 * C2P		
Clb5 degradation in C5P	C5P -> Sic1P	Mass Action	Vdb5 * C5P		
Cdc6 synthesis	-> Cdc6	Mass Action	ksf6' + ksf6'' * Swi5 + ksf6''' * SBF		
Cdc6 phosphorylation	Cdc6 -> Cdc6P	Mass Action	Vkpf6 * Cdc6		
Cdc6 dephosphorylation	Cdc6P -> Cdc6	Mass Action	kppf6 * Cdc14 * Cdc6P		
Cdc6P degradation (fast)	Cdc6P ->	Mass Action	kd3f6 * Cdc6P		
Assocn of Clb2 and Cdc6	Clb2 + Cdc6 -> F2	Mass Action	kasf2 * Clb2 * Cdc6		
Dissocn of F2 (Clb2/Cdc6)	F2 -> Clb2 + Cdc6	Mass Action	kdif2 * F2		
Assocn of Clb5 and Cdc6	Clb5 + Cdc6 -> F5	Mass Action	kasf5 * Clb5 * Cdc6		
Dissocn of F5 (Clb5/Cdc6)	F5 -> Clb5 + Cdc6	Mass Action	kdif5 * F5		
F2 phosphorylation	F2 -> F2P	Mass Action	Vkpf6 * F2		
F2P dephosphorylation	F2P -> F2	Mass Action	kppf6 * Cdc14 * F2P		
F5 phosphorylation	F5 -> F5P	Mass Action	Vkpf6 * F5		
F5P dephosphorylation	F5P -> F5	Mass Action	kppf6 * Cdc14 * F5P		
Clb2 degradation in F2	F2 -> Cdc6	Mass Action	Vdb2 * F2		

Figure 6.2: 1^{st} of 3 screenshots of the reaction network for the budding yeast model of Figure 6.1 in the JigCell ModelBuilder.



Name	Reaction	Type	Equation	Fast	Notes
Clb2 degradation in F2	F2 -> Cdc6	Mass Action	Vdb2 * F2		
Clb5 degradation in F5	F5 -> Cdc6	Mass Action	Vdb5 * F5		
Cdc6 degradation in F2P	F2P -> Clb2	Mass Action	kd3f6 * F2P		
Cdc6 degradation in F5P	F5P -> Clb5	Mass Action	kd3f6 * F5P		
Clb2 degradation in F2P	F2P -> Cdc6P	Mass Action	Vdb2 * F2P		
Clb5 degradation in F5P	F5P -> Cdc6P	Mass Action	Vdb5 * F5P		
Swi5 synthesis	-> Swi5	Mass Action	ksswi' + ksswi'' * Mcm1		
Swi5 degradation	Swi5 ->	Mass Action	kdswi * Swi5		
Swi5P degradation	Swi5P ->	Mass Action	kdswi * Swi5P		
Swi5 activation	Swi5P -> Swi5	Mass Action	kaswi * Cdc14 * Swi5P		
Swi5 inactivation	Swi5 -> Swi5P	Mass Action	kiswi * Clb2 * Swi5		
APC phosphorylation	APC -> APCP	Michaelis-Menten	kaapc * APC * Clb2 / (Jaapc + APC)		
APC inactivation	APCP -> APC	Michaelis-Menten	kiapc * APCP * 1.0 / (Jiapc + APCP)		
Cdc20 (inactive) production	-> Cdc20I	Mass Action	ks20' + ks20'' * Mcm1		
Cdc20 (inactive) degradation	Cdc20I ->	Mass Action	kd20 * Cdc20I		
Cdc20 (active) degradation	Cdc20A ->	Mass Action	kd20 * Cdc20A		
Cdc20 (active) activation	Cdc20I -> Cdc20A	Mass Action	ka20' + ka20'' * APCP * Cdc20I		
Cdc20 (active) inactivation	Cdc20A -> Cdc20I	Mass Action	mad2 * Cdc20A		
Cdh1 (active) production	-> Cdh1	Mass Action	kscdh		
Cdh1 (active) degradation	Cdh1 ->	Mass Action	kdcdh * Cdh1		
Cdh1 (inactive) degradation	Cdh1I ->	Mass Action	kdcdh * Cdh1I		
Cdh1 (active) activation	Cdh1I -> Cdh1	Michaelis-Menten	Vacdh * Cdh1I * 1.0 / (Jacdh + Cdh1I)		
Cdh1 (active) inactivation	Cdh1 -> Cdh1I	Michaelis-Menten	Vicdh * Cdh1 * 1.0 / (Jicdh + Cdh1)		
Cdc14 production	-> Cdc14	Mass Action	ks14		
Cdc14 degradation	Cdc14 ->	Mass Action	kd14 * Cdc14		
Assocn with Net1 to form RENT	Net1 + Cdc14 -> RENT	Mass Action	kasrent * Net1 * Cdc14		
Dissocn of RENT	RENT -> Net1 + Cdc14	Mass Action	kdirent * RENT		
Assocn with Net1P to form RENTP	Net1P + Cdc14 -> RENTP	Mass Action	kasrentp * Net1P * Cdc14		
Dissocn of RENTP	RENTP -> Net1P + Cdc14	Mass Action	kdirentp * RENTP		
Net1 production	-> Net1	Mass Action	ksnet		
Net1 degradation	Net1 ->	Mass Action	kdnet * Net1		
Net1P degradation	Net1P ->	Mass Action	kdnet * Net1P		
Net1 phosphorylation	Net1 -> Net1P	Mass Action	Vkpnet * Net1		
Net1 dephosphorylation	Net1P -> Net1	Mass Action	Vppnet * Net1P		
RENT phosphorylation	RENT -> RENTP	Mass Action	Vkpnet * RENT		
RENT dephosphorylation	RENTP -> RENT	Mass Action	Vppnet * RENTP		
Degradation of Net1 in RENT	RENT -> Cdc14	Mass Action	kdnet * RENT		
Degradation of Net1P in RENTP	RENTP -> Cdc14	Mass Action	kdnet * RENTP		

Figure 6.3: 2^{nd} of 3 screenshots of the reaction network for the budding yeast model of Figure 6.1 in the JigCell ModelBuilder.

Degradation of Net1P in RENTP	RENTP -> Cdc14	Mass Action	kdnet * RENTP		
Degradation of Cdc14 in RENT	RENT -> Net1	Mass Action	kd14 * RENT		
Degradation of Cdc14 in RENTP	RENTP -> Net1P	Mass Action	kd14 * RENTP		
Tem1 kinetics (activation)	Tem1GDP -> Tem1GTP	Michaelis-Menten	kte1 * Tem1GDP * 1.0 / (Jatem + Tem1GDP)		
(inactivation)	Tem1GTP -> Tem1GDP	Michaelis-Menten	kbub2 * Tem1GTP * 1.0 / (Jitem + Tem1GTP)		
Cdc15 kinetics (activation)	Cdc15i -> Cdc15	Mass Action	ka15' * Tem1GDP + ka15'' * Tem1GTP + ka15''' * Cdc14 * ...		
(inactivation)2	Cdc15 -> Cdc15i	Mass Action	ki15 * Cdc15		
PPX synthesis	-> PPX	Mass Action	ksppx		
PPX degradation	PPX ->	Mass Action	Vdppx * PPX		
Pds1 production	-> Pds1	Mass Action	kspds' + ks1pds'' * SBF + ks2pds'' * Mcm1		
Pds1 degradation	Pds1 ->	Mass Action	Vdpds * Pds1		
Degradation of Pds1 in PE	PE -> Esp1	Mass Action	Vdpds * PE		
Assocn with Esp1 to form PE	Esp1 + Pds1 -> PE	Mass Action	kasesp * Esp1 * Pds1		
Dissocn of PE	PE -> Pds1 + Esp1	Mass Action	kdiesp * PE		
ORI kinetics	-> ORI	Mass Action	ksori * (eorib5 * Clb5 + eorib2 * Clb2)		
	ORI ->	Mass Action	kdori * ORI		
BUD kinetics	-> BUD	Mass Action	ksbud * (ebudn2 * Cln2 + ebudn3 * Cln3 + ebudb5 * Clb5)		
	BUD ->	Mass Action	kdbud * BUD		
SPN kinetics	-> SPN	Mass Action	kspn * Clb2 / (Jspn + Clb2)		
	SPN ->	Mass Action	kdspn * SPN		
	-> mad2	Mass Action	0.0		
	-> kte1	Mass Action	0.0		
	-> bub2	Mass Action	0.0		

Figure 6.4: 3^{rd} of 3 screenshots of the reaction network for the budding yeast model of Figure 6.1 in the JigCell ModelBuilder.

6.2 Morphogenesis Checkpoint Model

The main transitions of the cell cycle, namely the onset of DNA replication (Start transition), entry into mitosis (G2-M transition), and exit from mitosis (Finish transition), are controlled by checkpoint mechanisms. When bud formation is impaired (perhaps due to environmental stimuli such as heat or osmotic shock), the morphogenesis checkpoint in budding yeast delays the cell cycle at the G2-M transition. This delay effectively prevents the formation of dinucleated cells, which are less viable than mononucleated cells. However, the arrest in the cycle is not complete. After several hours, the cell cycle will continue and unbudded cells will undergo mitosis and dinucleate cells will be formed. Ciliberto et al. [12] previously developed a model of the morphogenesis checkpoint in budding yeast, using a much simpler version of Chen's cell cycle engine. Figure 6.5 shows their wiring diagram for the $Swe1$ box part of the model. The complete model is presented in Figures 6.6 and 6.7. The version of the morphogenesis model in the ModelBuilder consists of 52 chemical reaction equations. The ModelBuilder translates the biochemical reaction network into a system of differential equations with 21 differential equations.

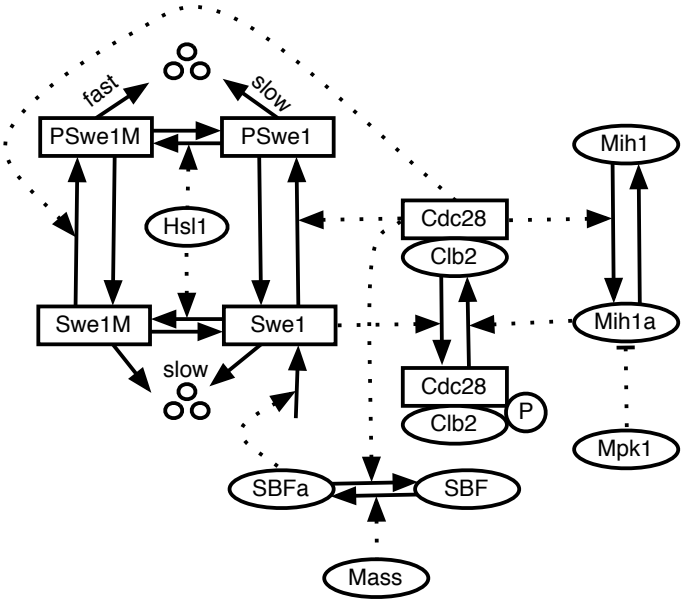
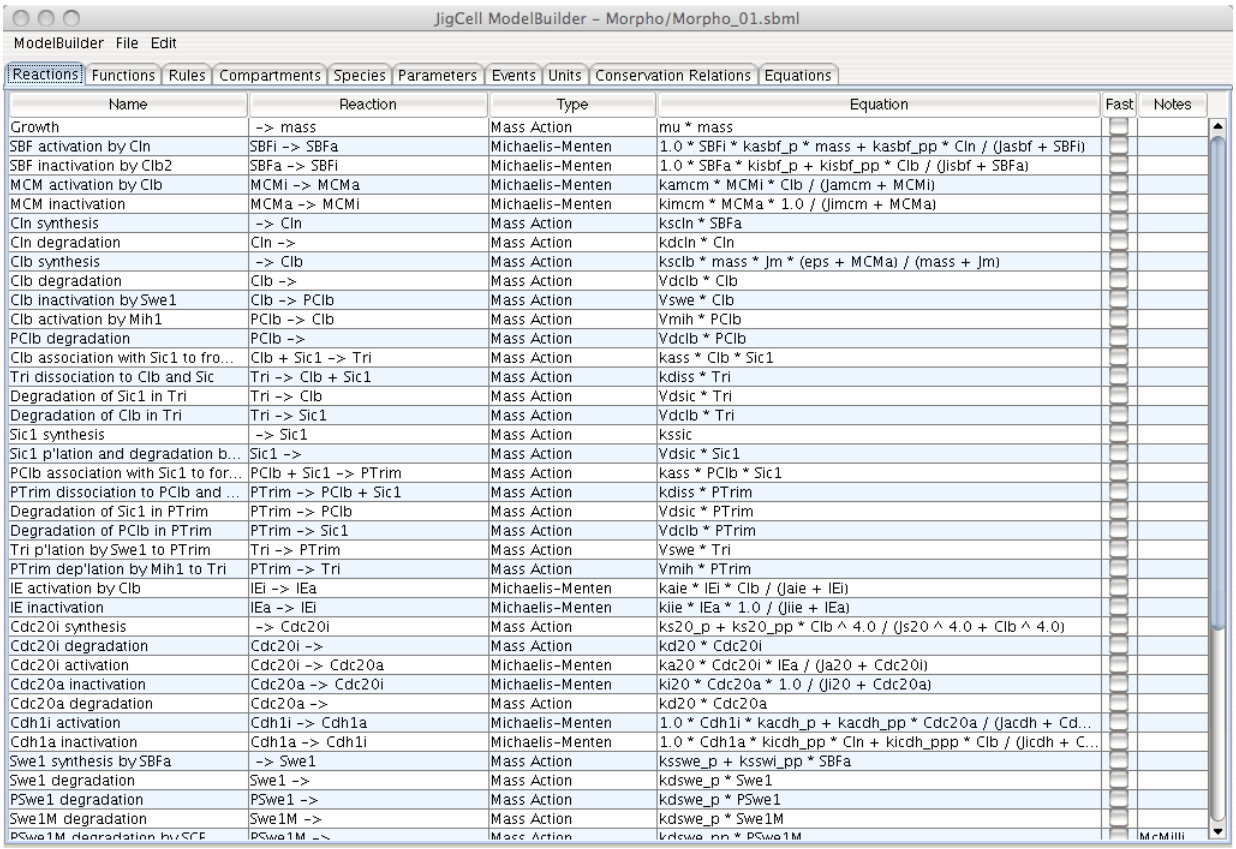


Figure 6.5: Wiring diagram for the morphogenesis checkpoint model by Ciliberto.



Name	Reaction	Type	Equation	Fast	Notes
Growth	-> mass	Mass Action	mu * mass		
SBF activation by Cln	SBFi -> SBFa	Michaelis-Menten	1.0 * SBFi * kasbf_p * mass + kasbf_pp * Cln / (Jasbf + SBFi)		
SBF inactivation by Clb2	SBFa -> SBFi	Michaelis-Menten	1.0 * SBFa * kisbf_p + kisbf_pp * Clb / (Jisbf + SBFa)		
MCM activation by Clb	MCMi -> MCMa	Michaelis-Menten	kamcm * MCMi * Clb / (Jamcm + MCMi)		
MCM inactivation	MCMa -> MCMi	Michaelis-Menten	kimcm * MCMa * 1.0 / (Jimcm + MCMa)		
Cln synthesis	-> Cln	Mass Action	kscln * SBFa		
Cln degradation	Cln ->	Mass Action	kdcln * Cln		
Clb synthesis	-> Clb	Mass Action	ksclb * mass * Jm * (eps + MCMa) / (mass + Jm)		
Clb degradation	Clb ->	Mass Action	Vdclb * Clb		
Clb inactivation by Swe1	Clb -> PClb	Mass Action	Vswe * Clb		
Clb activation by Mih1	PClb -> Clb	Mass Action	Vmih * PClb		
PClb degradation	PClb ->	Mass Action	Vdclb * PClb		
Clb association with Sic1 to fro...	Clb + Sic1 -> Tri	Mass Action	kass * Clb * Sic1		
Tri dissociation to Clb and Sic	Tri -> Clb + Sic1	Mass Action	kdiss * Tri		
Degradation of Sic1 in Tri	Tri -> Clb	Mass Action	Vdsic * Tri		
Degradation of Clb in Tri	Tri -> Sic1	Mass Action	Vdclb * Tri		
Sic1 synthesis	-> Sic1	Mass Action	kssic		
Sic1 p'lation and degradation b...	Sic1 ->	Mass Action	Vdsic * Sic1		
PClb association with Sic1 to for...	PClb + Sic1 -> PTrim	Mass Action	kass * PClb * Sic1		
PTrim dissociation to PClb and ...	PTrim -> PClb + Sic1	Mass Action	kdiss * PTrim		
Degradation of Sic1 in PTrim	PTrim -> PClb	Mass Action	Vdsic * PTrim		
Degradation of PClb in PTrim	PTrim -> Sic1	Mass Action	Vdclb * PTrim		
Tri p'lation by Swe1 to PTrim	Tri -> PTrim	Mass Action	Vswe * Tri		
PTrim dep'lation by Mih1 to Tri	PTrim -> Tri	Mass Action	Vmih * PTrim		
IE activation by Clb	IEi -> IEa	Michaelis-Menten	kaie * IEi * Clb / (Jaie + IEi)		
IE inactivation	IEa -> IEi	Michaelis-Menten	kiie * IEa * 1.0 / (Jiie + IEa)		
Cdc20i synthesis	-> Cdc20i	Mass Action	ks20_p + ks20_pp * Clb ^ 4.0 / (Js20 ^ 4.0 + Clb ^ 4.0)		
Cdc20i degradation	Cdc20i ->	Mass Action	kd20 * Cdc20i		
Cdc20i activation	Cdc20i -> Cdc20a	Michaelis-Menten	ka20 * Cdc20i * IEa / (Ja20 + Cdc20i)		
Cdc20a inactivation	Cdc20a -> Cdc20i	Michaelis-Menten	ki20 * Cdc20a * 1.0 / (Ji20 + Cdc20a)		
Cdc20a degradation	Cdc20a ->	Mass Action	kd20 * Cdc20a		
Cdh1i activation	Cdh1i -> Cdh1a	Michaelis-Menten	1.0 * Cdh1i * kacdh_p + kacdh_pp * Cdc20a / (Jacdh + Cd...		
Cdh1a inactivation	Cdh1a -> Cdh1i	Michaelis-Menten	1.0 * Cdh1a * kicdh_pp * Cln + kicdh_ppp * Clb / (Jicdh + C...		
Swe1 synthesis by SBFa	-> Swe1	Mass Action	ksswe_p + ksswi_pp * SBFa		
Swe1 degradation	Swe1 ->	Mass Action	kdswe_p * Swe1		
PSwe1 degradation	PSwe1 ->	Mass Action	kdswe_p * PSwe1		
Swe1M degradation	Swe1M ->	Mass Action	kdswe_p * Swe1M		
PSwe1M degradation by SCF	PSwe1M ->	Mass Action	kdswe_pp * PSwe1M		McMilli...

Figure 6.6: 1^{st} of 2 screenshots of the reaction network for the morphogenesis checkpoint model in budding yeast of Figure 6.5 in the JigCell ModelBuilder.

PSwe1M degradation by SCF	PSwe1M ->	Mass Action	kdswe_pp * PSwe1M		McMilli...
Swe1 inactivation by Clb to for...	Swe1 -> PSwe1	Michaelis-Menten	kiwee * Swe1 * Clb / (Jiwee + Swe1)		McMilli...
PSwe1 activation to Swe1	PSwe1 -> Swe1	Michaelis-Menten	kawee * PSwe1 * 1.0 / (Jawee + PSwe1)		
Swe1M inactivation by Clb to for...	Swe1M -> PSwe1M	Michaelis-Menten	kiwee * Swe1M * Clb / (Jiwee + Swe1M)		
PSwe1M activation to Swe1M	PSwe1M -> Swe1M	Michaelis-Menten	kawee * PSwe1M * 1.0 / (Jawee + PSwe1M)		
Swe1 inactivation by BUD to for...	Swe1 -> Swe1M	Mass Action	khsl1 * BUD * Swe1		
Swe1M activation to Swe1	Swe1M -> Swe1	Mass Action	khsl1r * Swe1M		
PSwe1 inactivation by Bud to for...	PSwe1 -> PSwe1M	Mass Action	khsl1 * BUD * PSwe1		
PSwe1M activation to PSwe1	PSwe1M -> PSwe1	Mass Action	khsl1r * PSwe1M		
Mih1i activation by Clb	Mih1i -> Mih1a	Michaelis-Menten	kamih * Mih1i * Clb / (Jamih + Mih1i)		
Mih1a inactivation	Mih1a -> Mih1i	Michaelis-Menten	kimih * Mih1a * 1.0 / (Jimih + Mih1a)		
BE synthesis	-> BE	Mass Action	ksbud * Cln		
BE degradation	BE ->	Mass Action	kdbud * BE		
Budding flag	-> BUD	Mass Action	0.0		
Anaphase flag	-> ANA	Mass Action	0.0		

Figure 6.7: 2^{nd} of 2 screenshots of the reaction network for the morphogenesis checkpoint model in budding yeast of Figure 6.5 in the JigCell ModelBuilder.

6.3 Fusion of the Cell Cycle and Morphogenesis Models

An expert modeler used the Fusion Wizard (Section 3.2) to combine the cell cycle model (Section 6.1) and the morphogenesis model (Section 6.2) to produce the combined model (Figure 6.8). The primary aim was to replace the simple cell cycle engine in the morphogenesis model with the more complex cell cycle model. Creating the combined model involved a single round of fusion followed by editing a number of reactions, rules, and events in the ModelBuilder.

The expert modeler started the Fusion Wizard and went through the initialization process, where the fused model was given a name ($Fused\_Cell\_Morpho$), the submodels where loaded into the application, and the environment was specified. The Fusion Wizard detected that the Events did not have any names or ids associated with them and informed the modeler that automatic names and ids would be assigned. This occurred as SBML <event> structures contain optional id and $name$ fields. Once the submodels were selected and the Fusion Wizard was initialized, name resolution for the various SBML components began, starting with the compartment mapping table (Figures 6.9).

The species mapping table generated for the $Fused\_Cell\_Morpho$ model was quite large and for the sake of convenience and readability is broken up into three screenshots (Figure 6.10, A.1, and A.2). Next, the function, rule, and event mapping tables were generated (Figures 6.11, 6.12, and 6.13). Like the species mapping table, both the reaction and parameter mapping tables produced were too large to fit on a single screen shot and had to be broken up into five (Figures 6.14, A.3, A.4, A.5, and A.6) and seven (Figures 6.15, A.7, A.8, A.9, A.10, A.11, and A.12) screenshots respectively.

After the mapping tables were generated, the newly created fused model was saved and opened in the JigCell ModelBuilder. New reactions were added (components in red in Figure 6.8) in the ModelBuilder (Figure 6.16, A.13, A.14, and A.15) and the rules (Figure 6.17) and events (Figure 6.18) were updated accordingly. These steps produced the $Fused\_Cell\_Morpho$ model which was then sent to a simulator (Section 6.6).

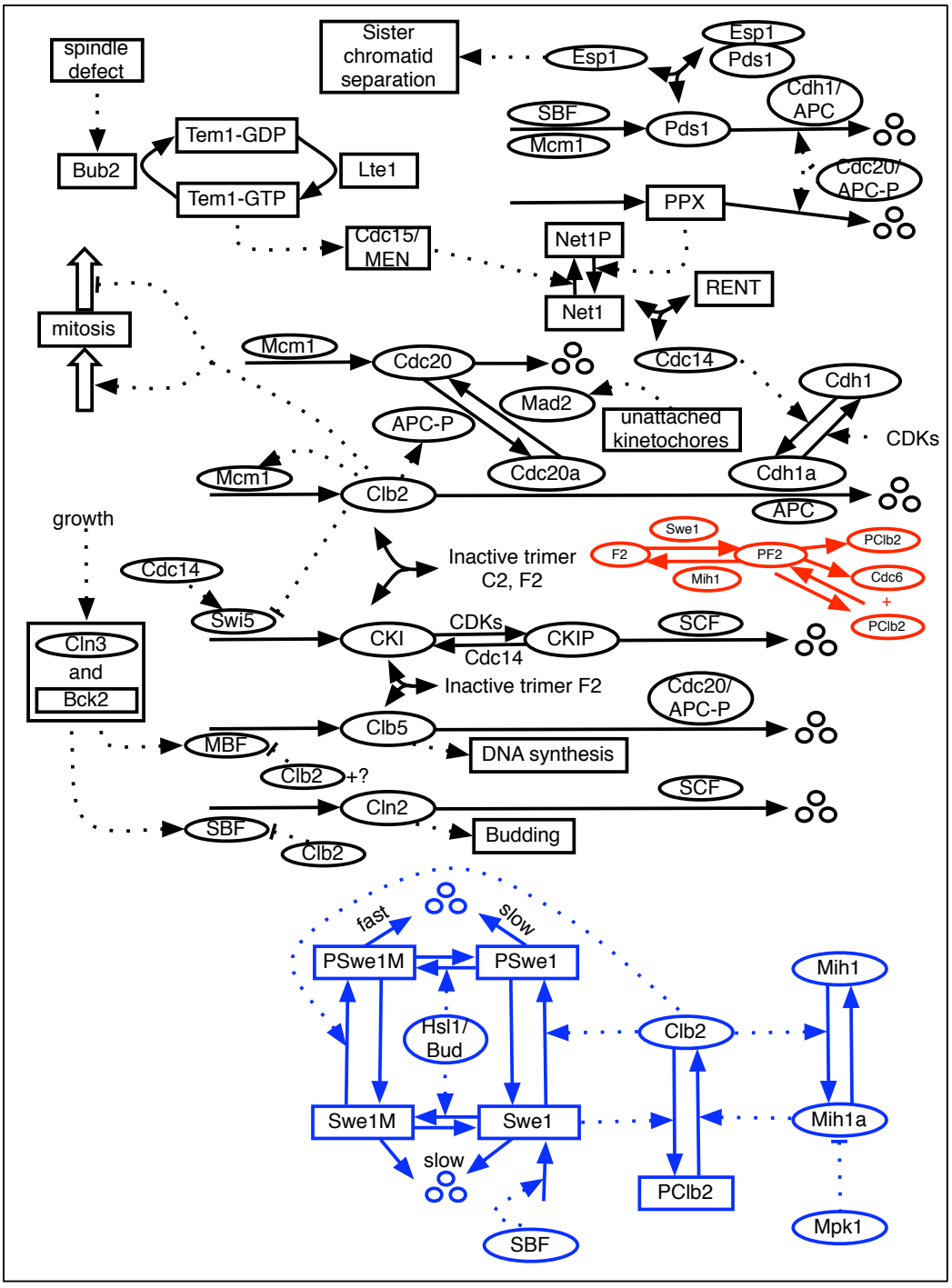


Figure 6.8: Fusion of the morphogenesis checkpoint model (in blue) and cell cycle model (in black) in budding yeast. Components in red were added after the fusion process was completed. For easier readability, not all reactions of the fused model are included in this schematic representation. For a full list of reactions refer to Figure 6.16.

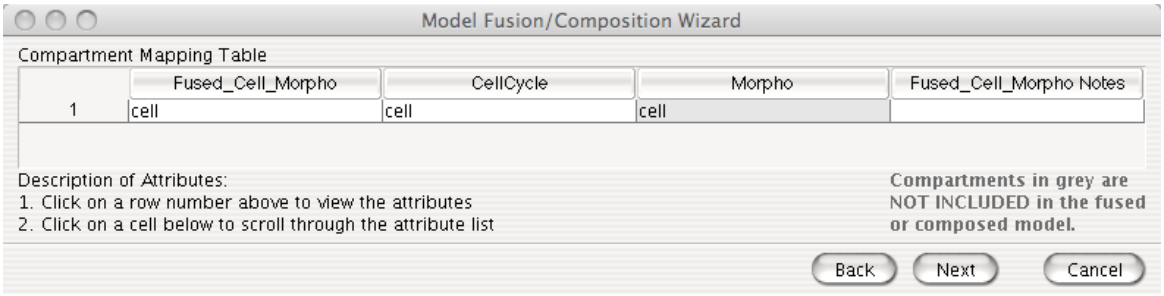


Figure 6.9: Compartment mapping table of the morphogenesis and cell cycle combined model.

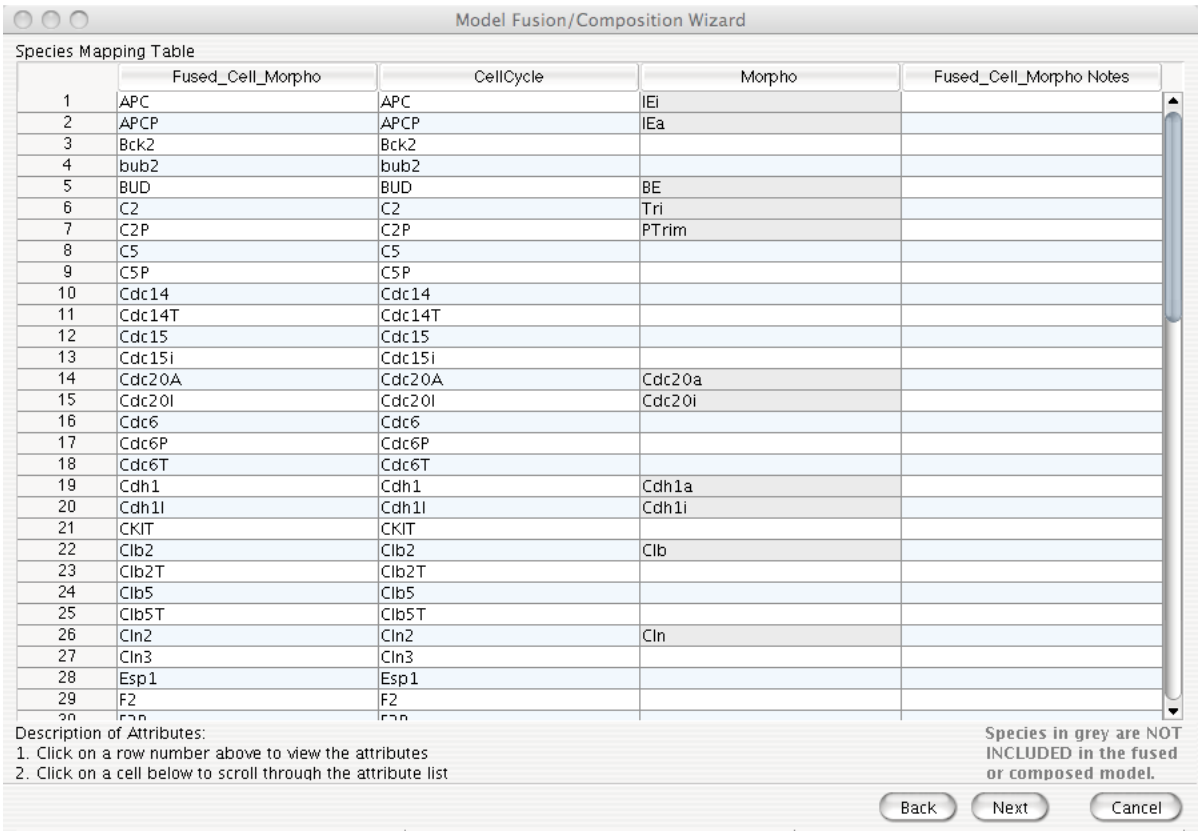


Figure 6.10: 1^{st} of 3 screenshots of the species mapping table of the morphogenesis and cell cycle combined model. The rest can be found in Appendix A.

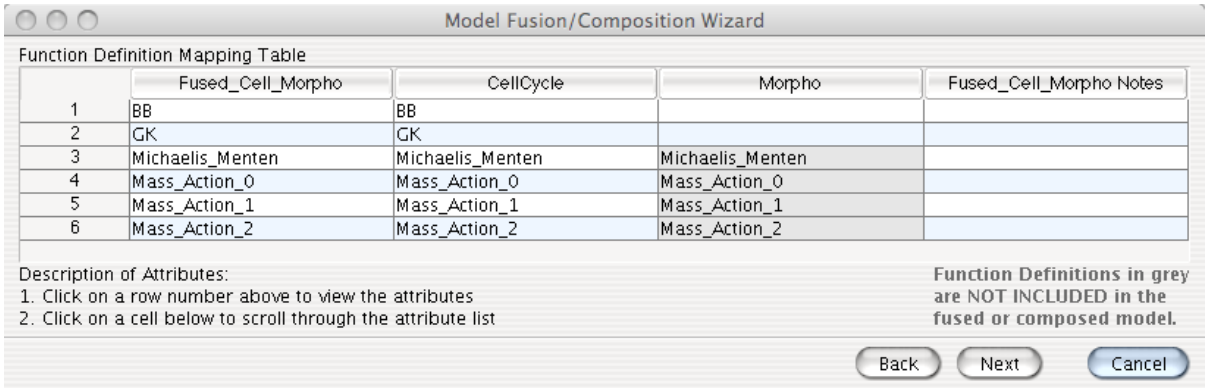


Figure 6.11: Function mapping table of the morphogenesis and cell cycle combined model.

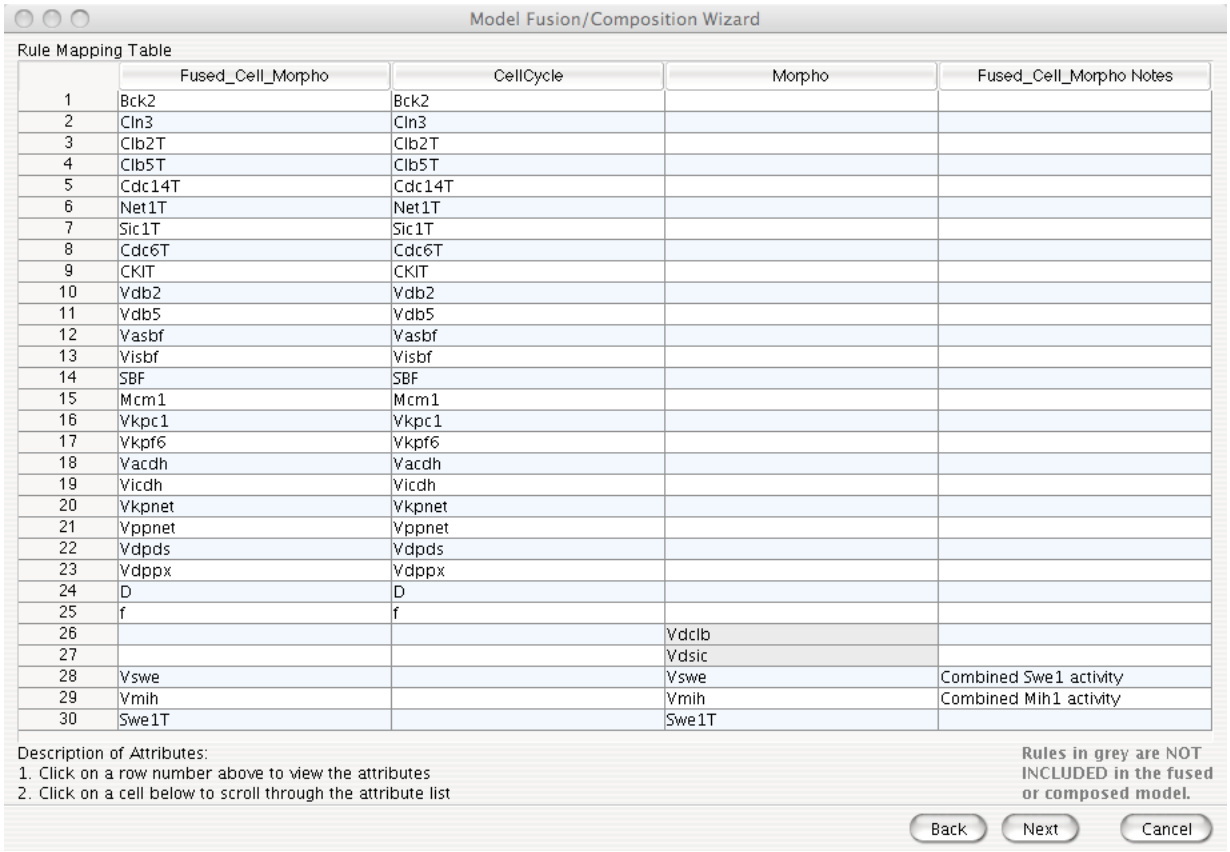


Figure 6.12: Rule mapping table of the morphogenesis and cell cycle combined model.

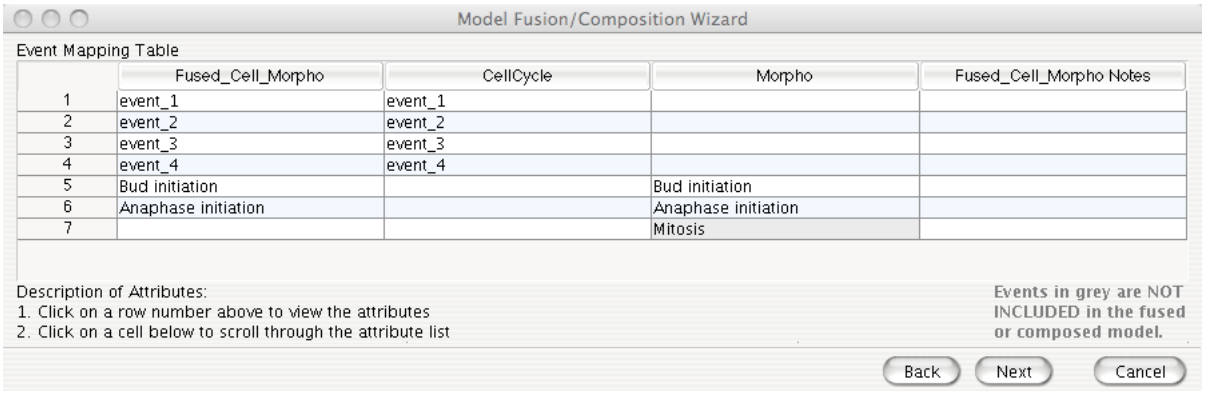


Figure 6.13: Event mapping table of the morphogenesis and cell cycle combined model.

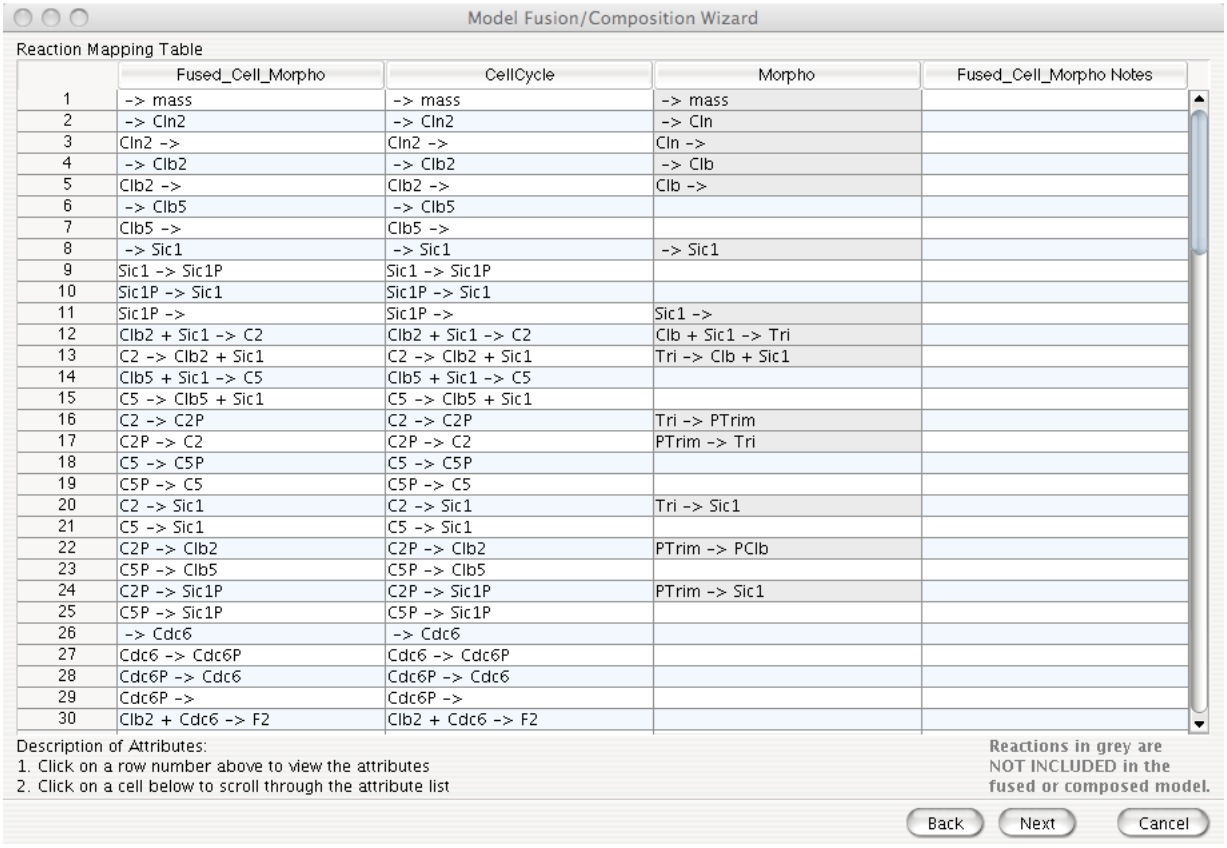


Figure 6.14: 1^{st} of 5 screenshots of the reaction mapping table of the morphogenesis and cell cycle combined model. The rest can be found in Appendix A.

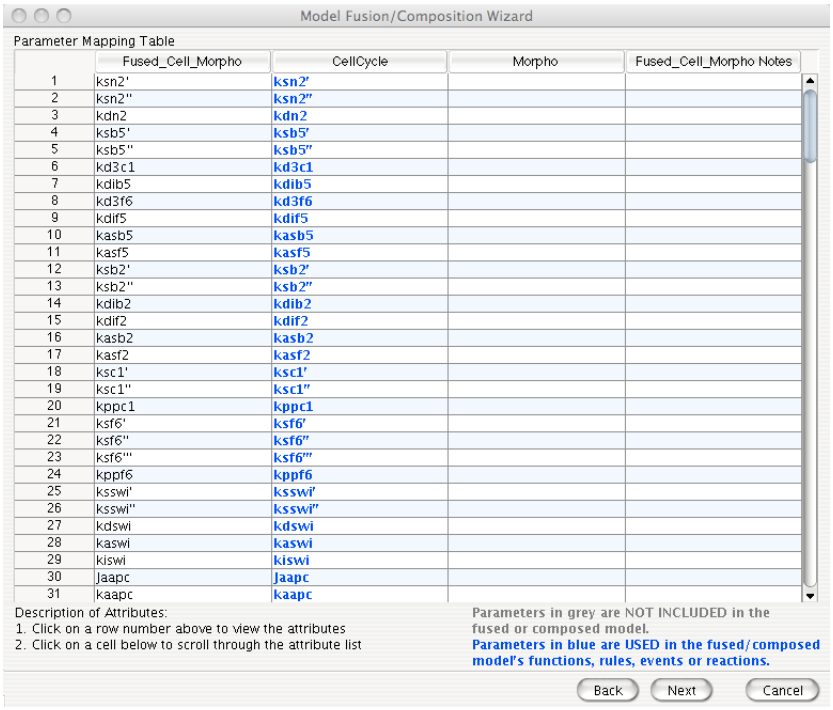


Figure 6.15: 1^{st} of 7 screenshots of the parameter mapping table of the morphogenesis and cell cycle combined model. The rest can be found in Appendix A.

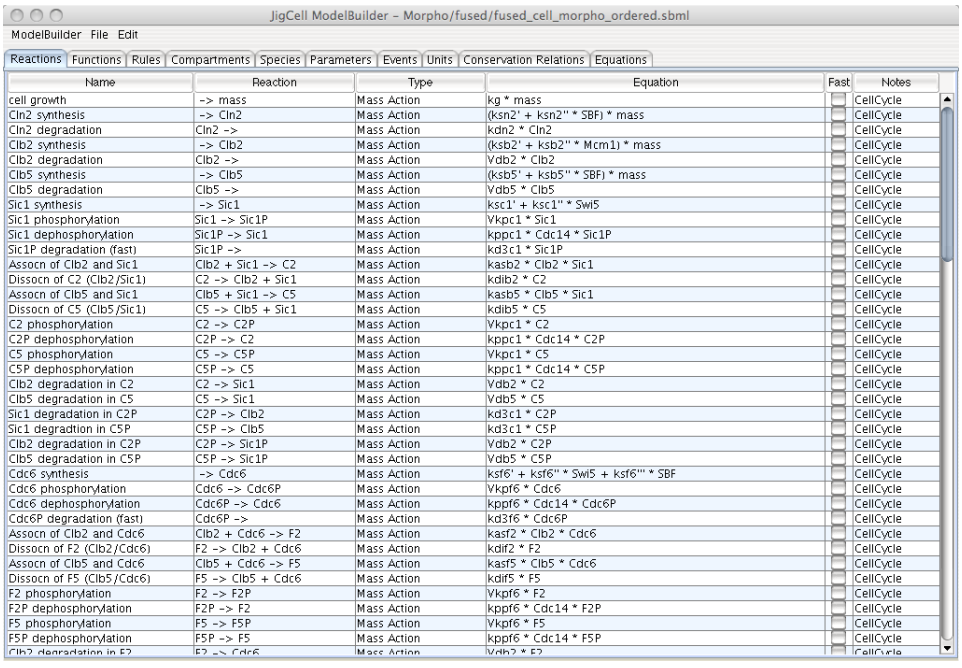
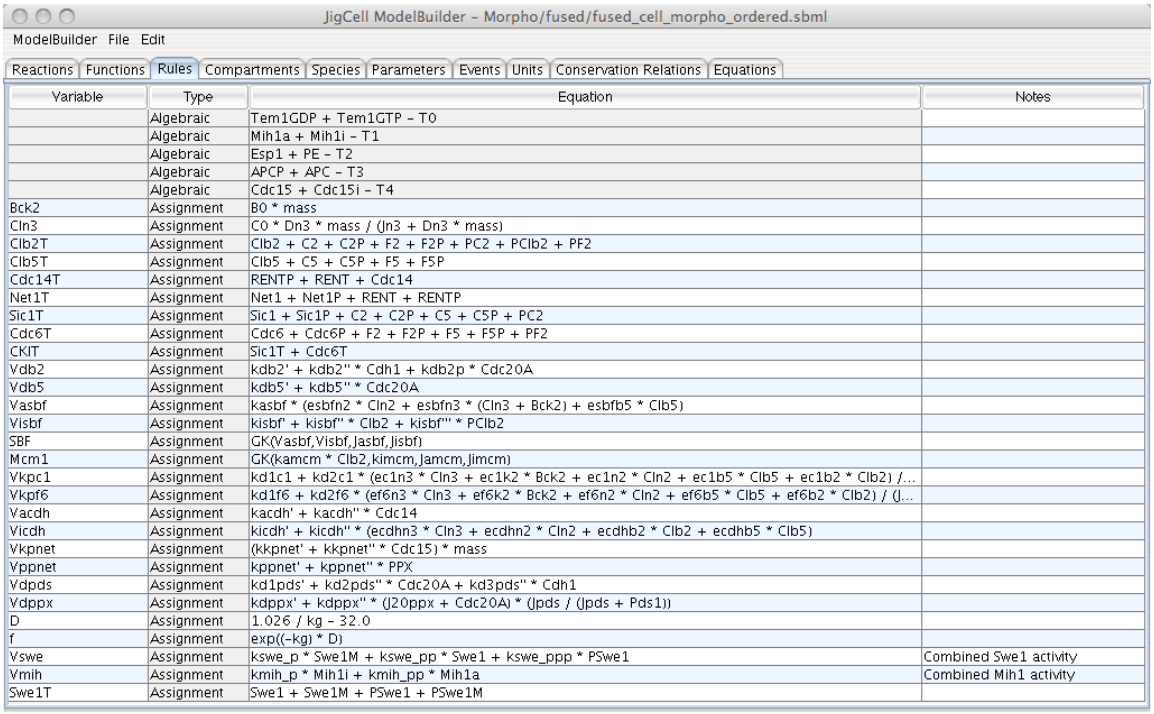
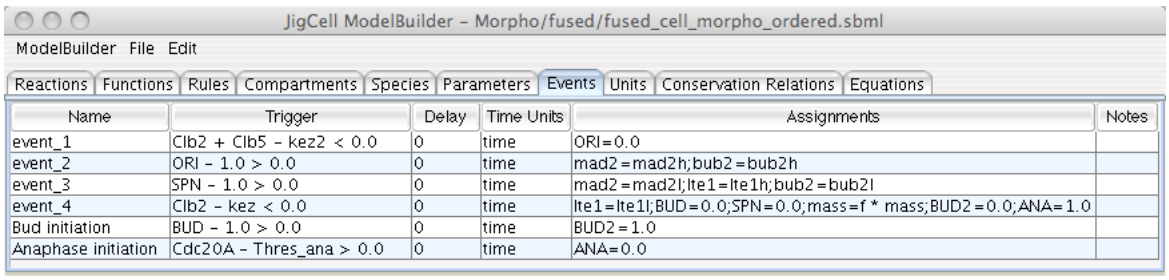


Figure 6.16: 1^{st} of 4 screenshots of the reactions spreadsheet for the $Fused\_Cell\_Morpho$ model in the JigCell ModelBuilder. The rest can be found in Appendix A.



Variable	Type	Equation	Notes
	Algebraic	Tem1GDP + Tem1GTP – T0	
	Algebraic	Mih1a + Mih1i – T1	
	Algebraic	Esp1 + PE – T2	
	Algebraic	APCP + APC – T3	
	Algebraic	Cdc15 + Cdc15i – T4	
Bck2	Assignment	B0 * mass	
Cln3	Assignment	C0 * Dn3 * mass / (Jn3 + Dn3 * mass)	
Clb2T	Assignment	Clb2 + C2 + C2P + F2 + F2P + PC2 + PClb2 + PF2	
Clb5T	Assignment	Clb5 + C5 + C5P + F5 + F5P	
Cdc14T	Assignment	RENTP + RENT + Cdc14	
Net1T	Assignment	Net1 + Net1P + RENT + RENTP	
Sic1T	Assignment	Sic1 + Sic1P + C2 + C2P + C5 + C5P + PC2	
Cdc6T	Assignment	Cdc6 + Cdc6P + F2 + F2P + F5 + F5P + PF2	
CKIT	Assignment	Sic1T + Cdc6T	
Vdb2	Assignment	kdb2' + kdb2'' * Cdh1 + kdb2p * Cdc20A	
Vdb5	Assignment	kdb5' + kdb5'' * Cdc20A	
Vasbf	Assignment	kasbf * (esbfn2 * Cln2 + esbfn3 * (Cln3 + Bck2) + esbfb5 * Clb5)	
Visbf	Assignment	kisbf' + kisbf'' * Clb2 + kisbf''' * PClb2	
SBF	Assignment	GK(Vasbf, Visbf, Jasbf, Jisbf)	
Mcm1	Assignment	GK(kamcm * Clb2, kimcm, Jamcm, Jimcm)	
Vkpc1	Assignment	kd1c1 + kd2c1 * (ec1n3 * Cln3 + ec1k2 * Bck2 + ec1n2 * Cln2 + ec1b5 * Clb5 + ec1b2 * Clb2) /...	
Vkpf6	Assignment	kd1f6 + kd2f6 * (ef6n3 * Cln3 + ef6k2 * Bck2 + ef6n2 * Cln2 + ef6b5 * Clb5 + ef6b2 * Clb2) / (J...	
Vacdh	Assignment	kacdh' + kacdh'' * Cdc14	
Vicdh	Assignment	kicdh' + kicdh'' * (ecdhn3 * Cln3 + ecdhn2 * Cln2 + ecdhb2 * Clb2 + ecdhb5 * Clb5)	
Vkpnet	Assignment	(kkpnet' + kkpnet'' * Cdc15) * mass	
Vppnet	Assignment	kppnet' + kppnet'' * PPX	
Vdpds	Assignment	kd1pds' + kd2pds'' * Cdc20A + kd3pds'' * Cdh1	
Vdppx	Assignment	kdppx' + kdppx'' * (J20ppx + Cdc20A) * (Jpds / (Jpds + Pds1))	
D	Assignment	1.026 / kg – 32.0	
f	Assignment	exp((-kg) * D)	
Vswe	Assignment	kswe_p * Swe1M + kswe_pp * Swe1 + kswe_ppp * PSwe1	Combined Swe1 activity
Vmih	Assignment	kmih_p * Mih1i + kmih_pp * Mih1a	Combined Mih1 activity
Swe1T	Assignment	Swe1 + Swe1M + PSwe1 + PSwe1M	

Figure 6.17: Rules spreadsheet for the $Fused\_Cell\_Morpho$ model in the JigCell ModelBuilder.



Name	Trigger	Delay	Time Units	Assignments	Notes
event_1	Clb2 + Clb5 – kez2 < 0.0	0	time	ORI=0.0	
event_2	ORI – 1.0 > 0.0	0	time	mad2=mad2h;bub2=bub2h	
event_3	SPN – 1.0 > 0.0	0	time	mad2=mad2l;lte1=lte1h;bub2=bub2l	
event_4	Clb2 – kez < 0.0	0	time	lte1=lte1l;BUD=0.0;SPN=0.0;mass=f * mass;BUD2=0.0;ANA=1.0	
Bud initiation	BUD – 1.0 > 0.0	0	time	BUD2=1.0	
Anaphase initiation	Cdc20A – Thres_ana > 0.0	0	time	ANA=0.0	

Figure 6.18: Events spreadsheet for the $Fused\_Cell\_Morpho$ model in the JigCell ModelBuilder.

6.4 Composition of the Cell Cycle and Morphogenesis Models

An expert modeler used the Composition Wizard (Section 4.3) to combine the cell cycle model (Section 6.1) and the morphogenesis model (Section 6.2) to produce a composed model called $Composed\_Cell\_Morpho$. Construction of the $Composed\_Cell\_Morpho$ model involved multiple rounds of composition and construction and are detailed below:

1. The cell cycle model and the morphogenesis model were composed together using the Composition Wizard to produce an intermediate $Composed\_Cell\_Morpho$ model.
2. The intermediate $Composed\_Cell\_Morpho$ model was then edited in the ModelBuilder to add additional reactions, and update certain rules and events.
3. The intermediate $Composed\_Cell\_Morpho$ was loaded into the Composition Wizard to produce the final $Composed\_Cell\_Morpho$ model.

This multi-step process was necessary as a number of new reactions were added to the intermediate $Composed\_Cell\_Morpho$ model (components in red in Figure 6.8). This resulted in having to run through the composition wizard again to create links between the newly added components and the existing ones.

As in fusion, the expert modeler started the Fusion/Composition Wizard, named the composed model $Composed\_Cell\_Morpho$, and selected to use the Composition Wizard. The submodels were then loaded into the application and the environment was specified. As in Section 6.3, the Composition Wizard detected events without ids or names within the submodels and resolved this automatically.

The initial compartment, function, unit, and parameter composition mapping tables generated by the Composition Wizard were not modified at all by the expert modeler during the two rounds of composition and therefore screenshots for them are not included (as they are essentially the same as their equivalent fusion mapping tables in the previous section). The initial species mapping table created during the first round of composition was large and was broken up into three screenshots (Figures 6.19, A.18, and A.19). The event and rule mapping tables were updated by the modeler during the first round of composition and remained unchanged thereafter (Figures 6.20 and 6.21 respectively). The initial reaction mapping table generated after the first round of composition was too large to fit on a single screenshot and had to be broken up into five screenshots (Figures 6.22, A.20, A.21, A.22, and A.23).

After the first round of composition was completed, the newly created composed model was saved and opened in the JigCell ModelBuilder. New reactions were added (components in red in Figure 6.8), and the rules and events were updated accordingly. The assignments of some rules and the event assignments of some events were modified to include the effects of both the cell cycle and morphogenesis models. The expert modeler then went through a second round of composition by opening the $Composed\_Cell\_Morpho$ model in the Composition Wizard using the "Open Composed Model" button (Section 4.3.1). This second round of composition produced the final species and reaction composition mapping tables (Figures 6.23, A.24, and A.25 and Figures 6.24, A.26, A.27, A.28, and A.29 respectively). Collectively these steps produced the final $Composed\_Cell\_Morpho$ model which was flattened for simulation (Section 6.6).

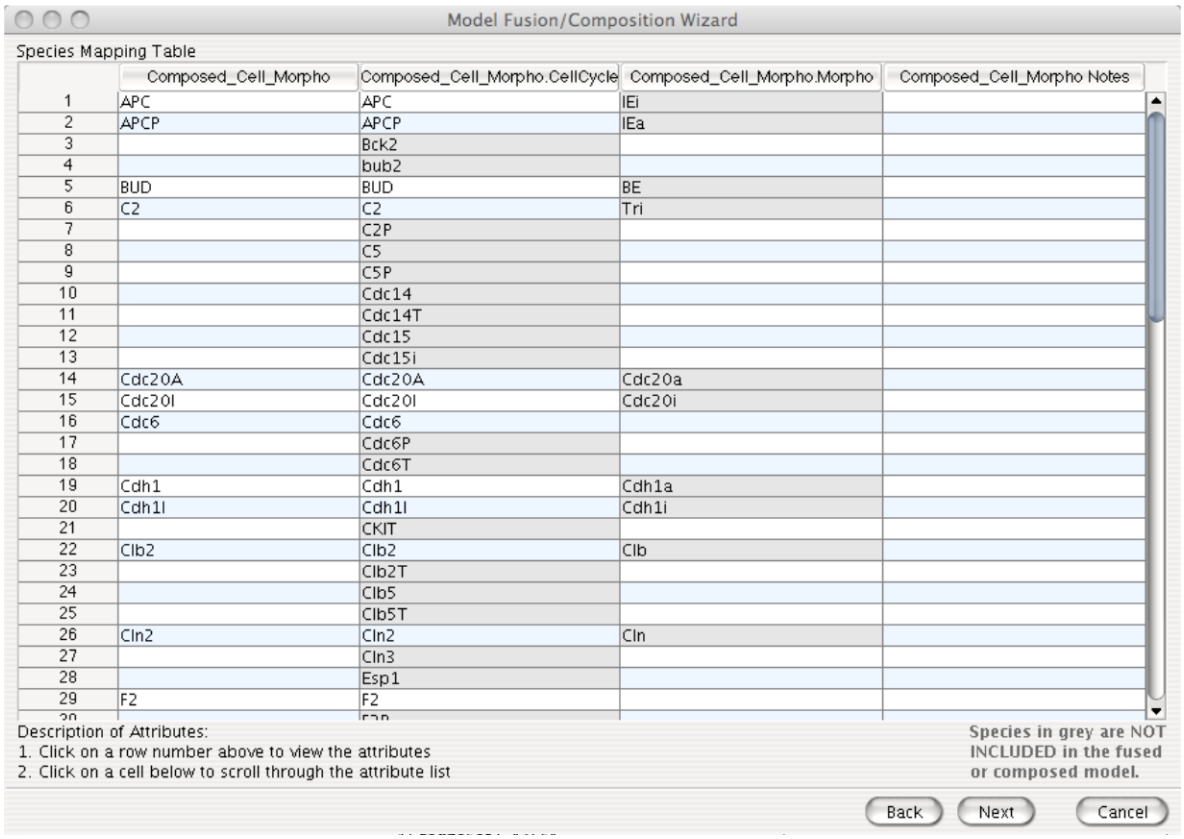


Figure 6.19: 1^{st} of 3 screenshots of the initial species composition mapping table of the morphogenesis and cell cycle combined model. The rest can be found in Appendix A.

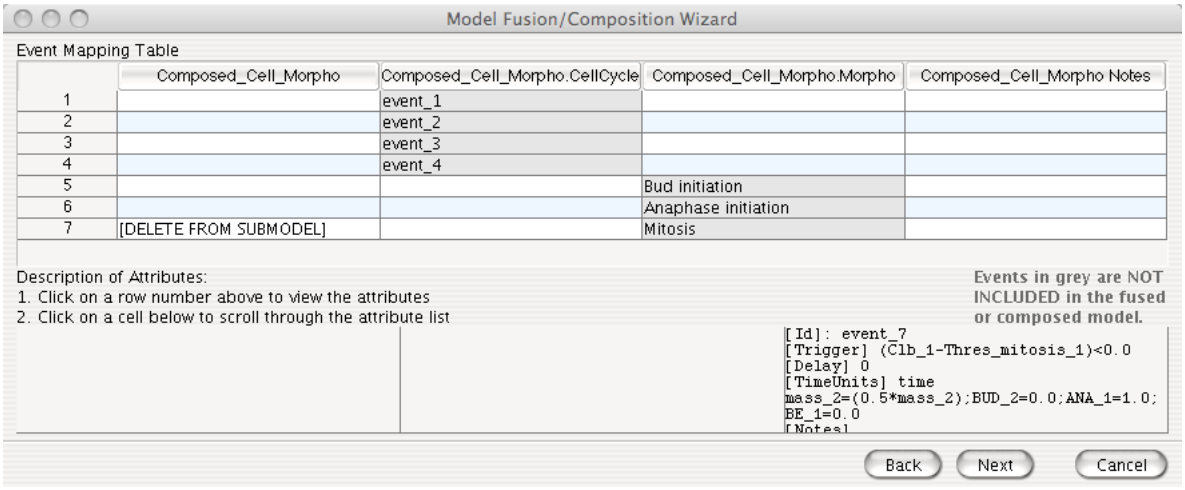


Figure 6.20: Screenshot of the event composition mapping table of the morphogenesis and cell cycle combined model.

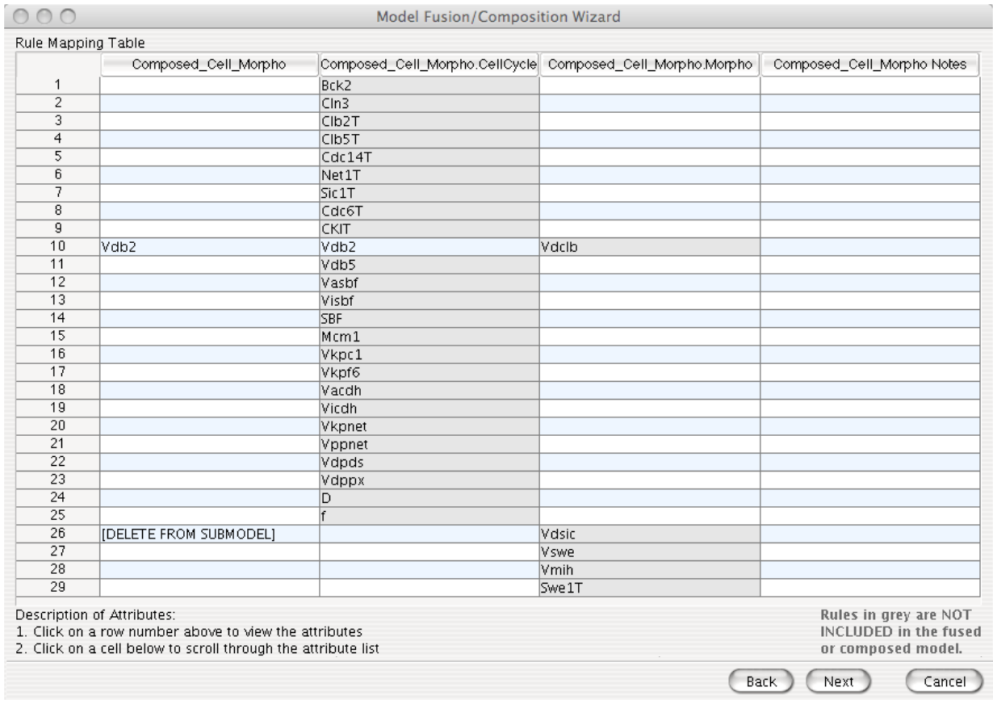


Figure 6.21: Screenshot of the rule composition mapping table of the morphogenesis and cell cycle combined model.

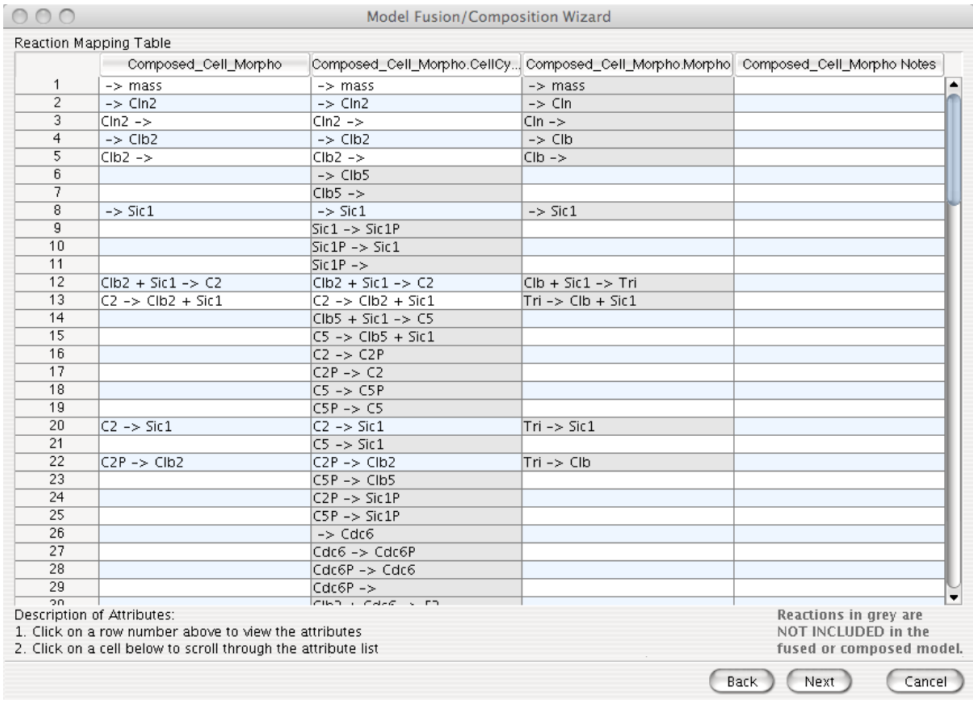


Figure 6.22: 1^{st} of 5 screenshots of the initial reaction composition mapping table of the morphogenesis and cell cycle combined model. The rest can be found in Appendix A.

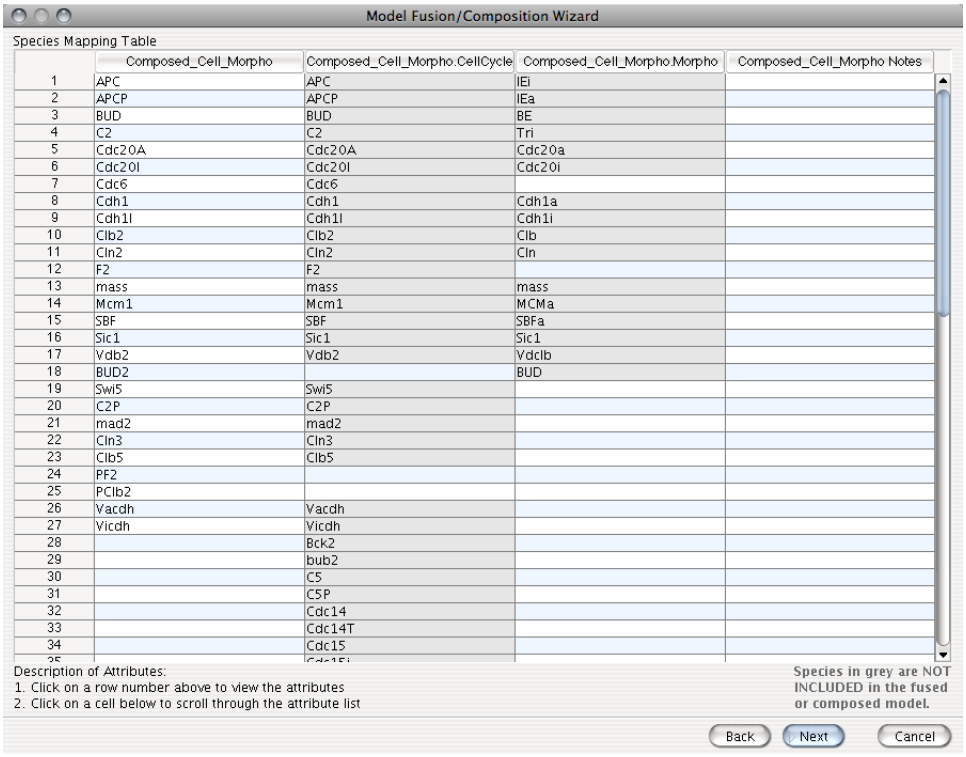


Figure 6.23: 1^{st} of 3 screenshots of the final species composition mapping table of the morphogenesis and cell cycle combined model. The rest can be found in Appendix A.

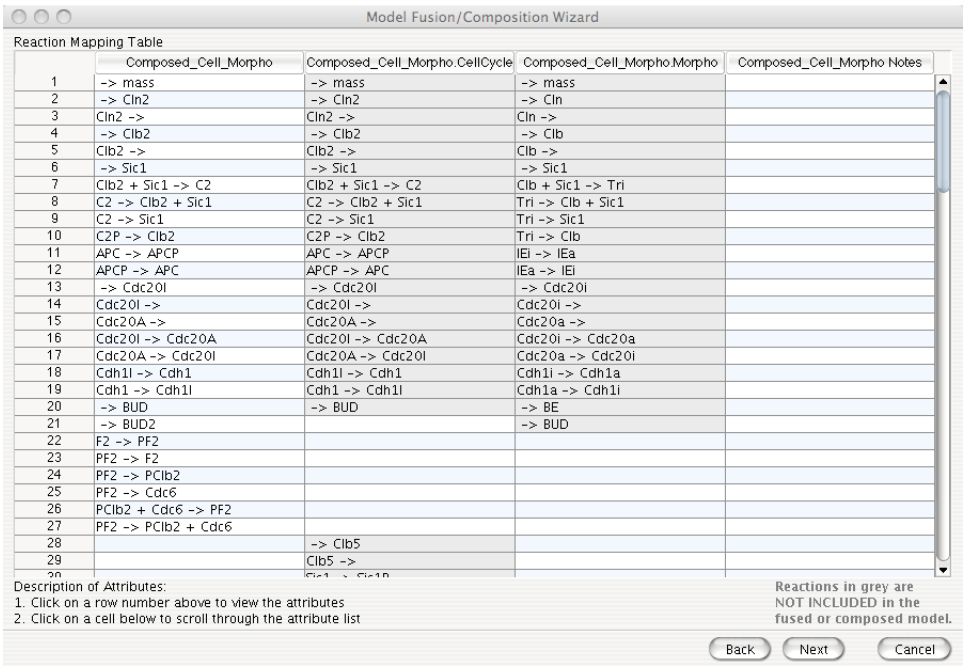


Figure 6.24: 1^{st} of 5 screenshots of the final reaction composition mapping table of the morphogenesis and cell cycle combined model. The rest can be found in Appendix A.

6.5 Aggregation of the Cell Cycle and Morphogenesis Models

The expert modeler first customized the morphogenesis and cell cycle models before beginning aggregation. First, redundancies were removed from the two models, this step essentially involved removing the reactions for the simpler cell cycle component of the morphogenesis model, and updating rules and events. Then the new reactions (in red in Figure 6.8) were added to the morphogenesis model. The screen shots for the ModelBuilder reactions, rules, and events spreadsheets for the cell cycle module (Figures 6.25, A.16, A.17, 6.26, and 6.27) and morphogenesis module (Figures 6.28, 6.29, and 6.30) are provided for reference. Finally, the two models were converted into modules ready for aggregation by specifying their ports. After these steps were performed, the modeler loaded the modules into the Aggregation Connector to create the aggregated model (called $Aggregated\_Cell\_Morpho$) by connecting the corresponding ports in each module to each other (Figure 6.31). It should be noted that usually species linked together via linked ports play roles in reactions across submodels (either as reactants, products or modifiers), however four particular species are linked in the $Aggregated\_Cell\_Morpho$ model that play roles in events ($BUD2$) and rules ($PF2$, $PC2$, and $PClb2$) in the $Aggregated\_Cell\_Morpho$ model. These steps produced the $Aggregated\_Cell\_Morpho$ model which was then flattened for simulation (Section 6.6).

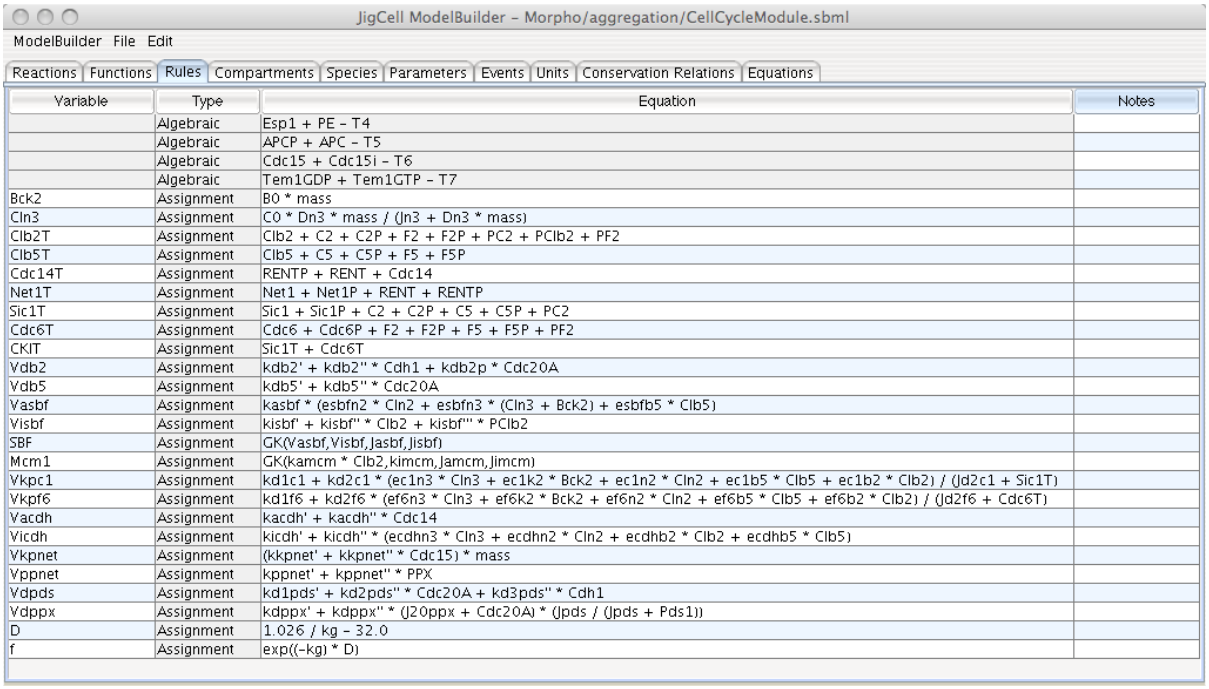
JigCell ModelBuilder - Morpho/aggregation/CellCycleModule.sbml

ModelBuilder File Edit

Reactions | Functions | Rules | Compartments | Species | Parameters | Events | Units | Conservation Relations | Equations

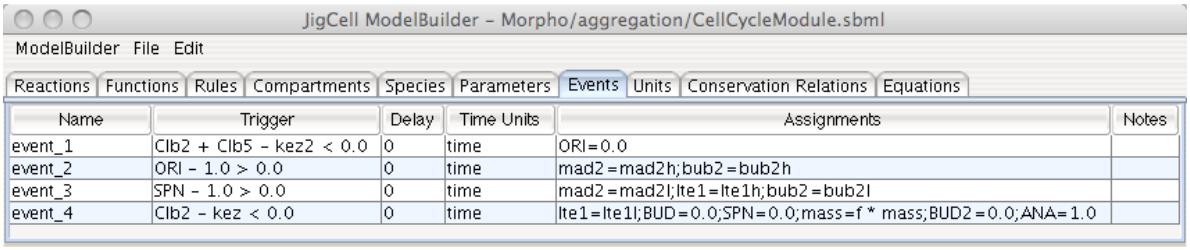
Name	Reaction	Type	Equation	Fast	Notes
cell growth	-> mass	Mass Action	kg * mass		
Cln2 synthesis	-> Cln2	Mass Action	(ksn2' + ksn2'' * SBF) * mass		
Cln2 degradation	Cln2 ->	Mass Action	kdn2 * Cln2		
Clb2 synthesis	-> Clb2	Mass Action	(ksb2' + ksb2'' * Mcm1) * mass		
Clb2 degradation	Clb2 ->	Mass Action	Vdb2 * Clb2		
Clb5 synthesis	-> Clb5	Mass Action	(ksb5' + ksb5'' * SBF) * mass		
Clb5 degradation	Clb5 ->	Mass Action	Vdb5 * Clb5		
Sic1 synthesis	-> Sic1	Mass Action	ksc1' + ksc1'' * Swi5		
Sic1 phosphorylation	Sic1 -> Sic1P	Mass Action	Vkpc1 * Sic1		
Sic1 dephosphorylation	Sic1P -> Sic1	Mass Action	kppc1 * Cdc14 * Sic1P		
Sic1P degradation (fast)	Sic1P ->	Mass Action	kd3c1 * Sic1P		
Assocn of Clb2 and Sic1	Clb2 + Sic1 -> C2	Mass Action	kasb2 * Clb2 * Sic1		
Dissocn of C2 (Clb2/Sic1)	C2 -> Clb2 + Sic1	Mass Action	kdib2 * C2		
Assocn of Clb5 and Sic1	Clb5 + Sic1 -> C5	Mass Action	kasb5 * Clb5 * Sic1		
Dissocn of C5 (Clb5/Sic1)	C5 -> Clb5 + Sic1	Mass Action	kdib5 * C5		
C2 phosphorylation	C2 -> C2P	Mass Action	Vkpc1 * C2		
C2P dephosphorylation	C2P -> C2	Mass Action	kppc1 * Cdc14 * C2P		
C5 phosphorylation	C5 -> C5P	Mass Action	Vkpc1 * C5		
C5P dephosphorylation	C5P -> C5	Mass Action	kppc1 * Cdc14 * C5P		
Clb2 degradation in C2	C2 -> Sic1	Mass Action	Vdb2 * C2		
Clb5 degradation in C5	C5 -> Sic1	Mass Action	Vdb5 * C5		
Sic1 degradation in C2P	C2P -> Clb2	Mass Action	kd3c1 * C2P		
Sic1 degradtion in C5P	C5P -> Clb5	Mass Action	kd3c1 * C5P		
Clb2 degradation in C2P	C2P -> Sic1P	Mass Action	Vdb2 * C2P		
Clb5 degradation in C5P	C5P -> Sic1P	Mass Action	Vdb5 * C5P		
Cdc6 synthesis	-> Cdc6	Mass Action	ksf6' + ksf6'' * Swi5 + ksf6''' * SBF		
Cdc6 phosphorylation	Cdc6 -> Cdc6P	Mass Action	Vkpf6 * Cdc6		
Cdc6 dephosphorylation	Cdc6P -> Cdc6	Mass Action	kppf6 * Cdc14 * Cdc6P		
Cdc6P degradation (fast)	Cdc6P ->	Mass Action	kd3f6 * Cdc6P		
Assocn of Clb2 and Cdc6	Clb2 + Cdc6 -> F2	Mass Action	kasf2 * Clb2 * Cdc6		
Dissocn of F2 (Clb2/Cdc6)	F2 -> Clb2 + Cdc6	Mass Action	kdif2 * F2		
Assocn of Clb5 and Cdc6	Clb5 + Cdc6 -> F5	Mass Action	kasf5 * Clb5 * Cdc6		
Dissocn of F5 (Clb5/Cdc6)	F5 -> Clb5 + Cdc6	Mass Action	kdif5 * F5		
F2 phosphorylation	F2 -> F2P	Mass Action	Vkpf6 * F2		
F2P dephosphorylation	F2P -> F2	Mass Action	kppf6 * Cdc14 * F2P		
F5 phosphorylation	F5 -> F5P	Mass Action	Vkpf6 * F5		
F5P dephosphorylation	F5P -> F5	Mass Action	kppf6 * Cdc14 * F5P		
Clb2 degradation in F2	F2 -> Cdc6	Mass Action	Vdb2 * F2		

Figure 6.25: 1^{st} of 3 screenshots of the reactions spreadsheet for the cell cycle module in the JigCell ModelBuilder. The rest can be found in Appendix A



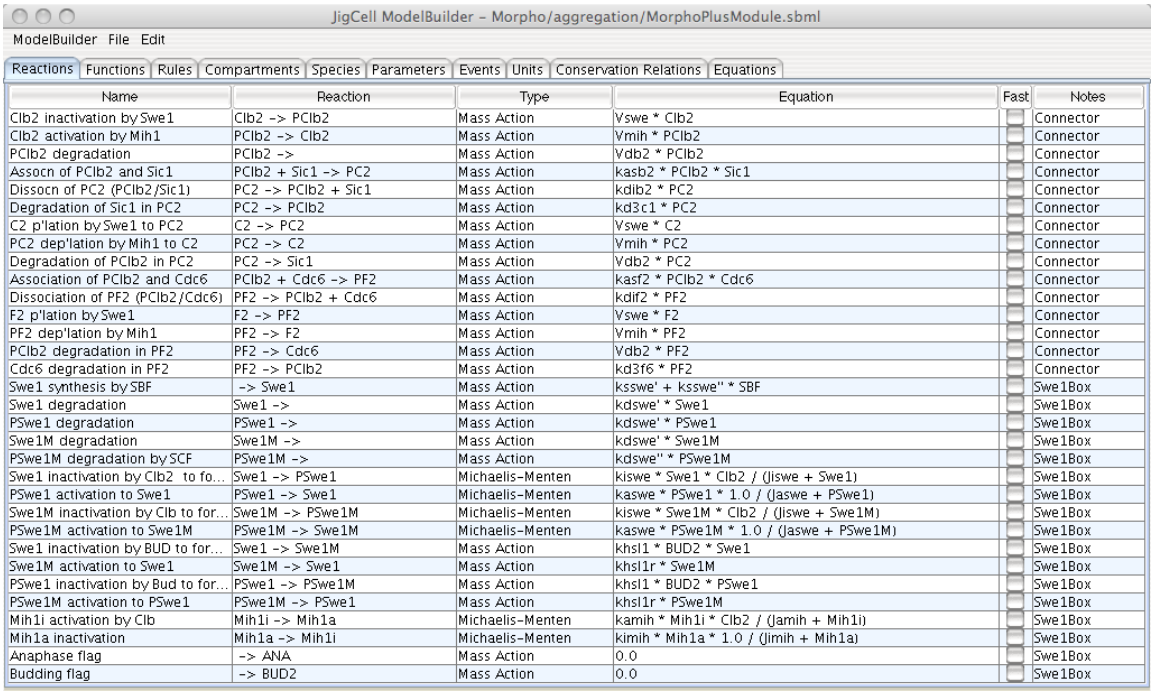
Variable	Type	Equation	Notes
	Algebraic	Esp1 + PE - T4	
	Algebraic	APCP + APC - T5	
	Algebraic	Cdc15 + Cdc15i - T6	
	Algebraic	Tem1GDP + Tem1GTP - T7	
Bck2	Assignment	B0 * mass	
Cln3	Assignment	C0 * Dn3 * mass / (Jn3 + Dn3 * mass)	
Clb2T	Assignment	Clb2 + C2 + C2P + F2 + F2P + PC2 + PClb2 + PF2	
Clb5T	Assignment	Clb5 + C5 + C5P + F5 + F5P	
Cdc14T	Assignment	RENTP + RENT + Cdc14	
Net1T	Assignment	Net1 + Net1P + RENT + RENTP	
Sic1T	Assignment	Sic1 + Sic1P + C2 + C2P + C5 + C5P + PC2	
Cdc6T	Assignment	Cdc6 + Cdc6P + F2 + F2P + F5 + F5P + PF2	
CKIT	Assignment	Sic1T + Cdc6T	
Vdb2	Assignment	kdb2' + kdb2'' * Cdh1 + kdb2p * Cdc20A	
Vdb5	Assignment	kdb5' + kdb5'' * Cdc20A	
Vasbf	Assignment	kasbf * (esbfn2 * Cln2 + esbfn3 * (Cln3 + Bck2) + esbfb5 * Clb5)	
Visbf	Assignment	kisbf' + kisbf'' * Clb2 + kisbf''' * PClb2	
SBF	Assignment	GK(Vasbf,Visbf,Jasbf,Jisbf)	
Mcm1	Assignment	GK(kamcm * Clb2,kimcm,Jamcm,Jimcm)	
Vkpc1	Assignment	kd1c1 + kd2c1 * (ec1n3 * Cln3 + ec1k2 * Bck2 + ec1n2 * Cln2 + ec1b5 * Clb5 + ec1b2 * Clb2) / (Jd2c1 + Sic1T)	
Vkpf6	Assignment	kd1f6 + kd2f6 * (ef6n3 * Cln3 + ef6k2 * Bck2 + ef6n2 * Cln2 + ef6b5 * Clb5 + ef6b2 * Clb2) / (Jd2f6 + Cdc6T)	
Vacdh	Assignment	kacdh' + kacdh'' * Cdc14	
Vicdh	Assignment	kicdh' + kicdh'' * (ecdhn3 * Cln3 + ecdhn2 * Cln2 + ecdhb2 * Clb2 + ecdhb5 * Clb5)	
Vkpnet	Assignment	(kkpnet' + kkpnet'' * Cdc15) * mass	
Vppnet	Assignment	kppnet' + kppnet'' * PPX	
Vdpds	Assignment	kd1pds' + kd2pds'' * Cdc20A + kd3pds'' * Cdh1	
Vdppx	Assignment	kdppx' + kdppx'' * (J20ppx + Cdc20A) * (Jpds / (Jpds + Pds1))	
D	Assignment	1.026 / kg - 32.0	
f	Assignment	exp((-kg) * D)	

Figure 6.26: Rules spreadsheet for the cell cycle module in the JigCell ModelBuilder.



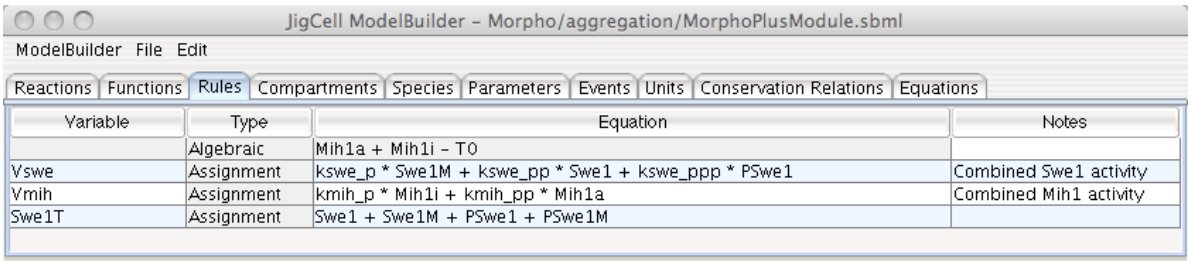
Name	Trigger	Delay	Time Units	Assignments	Notes
event_1	Clb2 + Clb5 - kez2 < 0.0	0	time	ORI=0.0	
event_2	ORI - 1.0 > 0.0	0	time	mad2=mad2h;bub2=bub2h	
event_3	SPN - 1.0 > 0.0	0	time	mad2=mad2l;Ite1=Ite1h;bub2=bub2l	
event_4	Clb2 - kez < 0.0	0	time	Ite1=Ite1l;BUD=0.0;SPN=0.0;mass=f * mass;BUD2=0.0;ANA=1.0	

Figure 6.27: Events spreadsheet for the cell cycle module in the JigCell ModelBuilder.



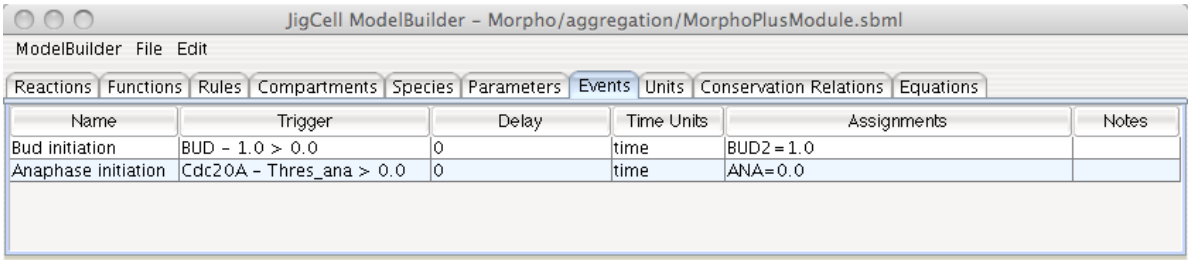
Name	Reaction	Type	Equation	Fast	Notes
Clb2 inactivation by Swe1	Clb2 -> PClb2	Mass Action	Vswe * Clb2		Connector
Clb2 activation by Mih1	PClb2 -> Clb2	Mass Action	Vmih * PClb2		Connector
PClb2 degradation	PClb2 ->	Mass Action	Vdb2 * PClb2		Connector
Assocn of PClb2 and Sic1	PClb2 + Sic1 -> PC2	Mass Action	kasb2 * PClb2 * Sic1		Connector
Dissocn of PC2 (PClb2/Sic1)	PC2 -> PClb2 + Sic1	Mass Action	kdib2 * PC2		Connector
Degradation of Sic1 in PC2	PC2 -> PClb2	Mass Action	kd3c1 * PC2		Connector
C2 p'lation by Swe1 to PC2	C2 -> PC2	Mass Action	Vswe * C2		Connector
PC2 dep'lation by Mih1 to C2	PC2 -> C2	Mass Action	Vmih * PC2		Connector
Degradation of PClb2 in PC2	PC2 -> Sic1	Mass Action	Vdb2 * PC2		Connector
Association of PClb2 and Cdc6	PClb2 + Cdc6 -> PF2	Mass Action	kasf2 * PClb2 * Cdc6		Connector
Dissociation of PF2 (PClb2/Cdc6)	PF2 -> PClb2 + Cdc6	Mass Action	kdif2 * PF2		Connector
F2 p'lation by Swe1	F2 -> PF2	Mass Action	Vswe * F2		Connector
PF2 dep'lation by Mih1	PF2 -> F2	Mass Action	Vmih * PF2		Connector
PClb2 degradation in PF2	PF2 -> Cdc6	Mass Action	Vdb2 * PF2		Connector
Cdc6 degradation in PF2	PF2 -> PClb2	Mass Action	kd3f6 * PF2		Connector
Swe1 synthesis by SBF	-> Swe1	Mass Action	ksswe' + ksswe'' * SBF		Swe1Box
Swe1 degradation	Swe1 ->	Mass Action	kdswe' * Swe1		Swe1Box
PSwe1 degradation	PSwe1 ->	Mass Action	kdswe' * PSwe1		Swe1Box
Swe1M degradation	Swe1M ->	Mass Action	kdswe' * Swe1M		Swe1Box
PSwe1M degradation by SCF	PSwe1M ->	Mass Action	kdswe'' * PSwe1M		Swe1Box
Swe1 inactivation by Clb2 to fo...	Swe1 -> PSwe1	Michaelis-Menten	kiswe * Swe1 * Clb2 / (Jiswe + Swe1)		Swe1Box
PSwe1 activation to Swe1	PSwe1 -> Swe1	Michaelis-Menten	kaswe * PSwe1 * 1.0 / (Jaswe + PSwe1)		Swe1Box
Swe1M inactivation by Clb to for...	Swe1M -> PSwe1M	Michaelis-Menten	kiswe * Swe1M * Clb2 / (Jiswe + Swe1M)		Swe1Box
PSwe1M activation to Swe1M	PSwe1M -> Swe1M	Michaelis-Menten	kaswe * PSwe1M * 1.0 / (Jaswe + PSwe1M)		Swe1Box
Swe1 inactivation by BUD to for...	Swe1 -> Swe1M	Mass Action	khsl1 * BUD2 * Swe1		Swe1Box
Swe1M activation to Swe1	Swe1M -> Swe1	Mass Action	khsl1r * Swe1M		Swe1Box
PSwe1 inactivation by Bud to for...	PSwe1 -> PSwe1M	Mass Action	khsl1 * BUD2 * PSwe1		Swe1Box
PSwe1M activation to PSwe1	PSwe1M -> PSwe1	Mass Action	khsl1r * PSwe1M		Swe1Box
Mih1i activation by Clb	Mih1i -> Mih1a	Michaelis-Menten	kamih * Mih1i * Clb2 / (Jamih + Mih1i)		Swe1Box
Mih1a inactivation	Mih1a -> Mih1i	Michaelis-Menten	kimih * Mih1a * 1.0 / (Jimih + Mih1a)		Swe1Box
Anaphase flag	-> ANA	Mass Action	0.0		Swe1Box
Budding flag	-> BUD2	Mass Action	0.0		Swe1Box

Figure 6.28: Reactions spreadsheet for the morphogenesis module in the JigCell ModelBuilder.



Variable	Type	Equation	Notes
	Algebraic	Mih1a + Mih1i - T0	
Vswe	Assignment	kswe_p * Swe1M + kswe_pp * Swe1 + kswe_ppp * PSwe1	Combined Swe1 activity
Vmih	Assignment	kmih_p * Mih1i + kmih_pp * Mih1a	Combined Mih1 activity
Swe1T	Assignment	Swe1 + Swe1M + PSwe1 + PSwe1M	

Figure 6.29: Rules spreadsheet for the morphogenesis module in the JigCell ModelBuilder.



Name	Trigger	Delay	Time Units	Assignments	Notes
Bud initiation	BUD - 1.0 > 0.0	0	time	BUD2 = 1.0	
Anaphase initiation	Cdc20A - Thres_ana > 0.0	0	time	ANA=0.0	

Figure 6.30: Events spreadsheet for the morphogenesis module in the JigCell ModelBuilder.

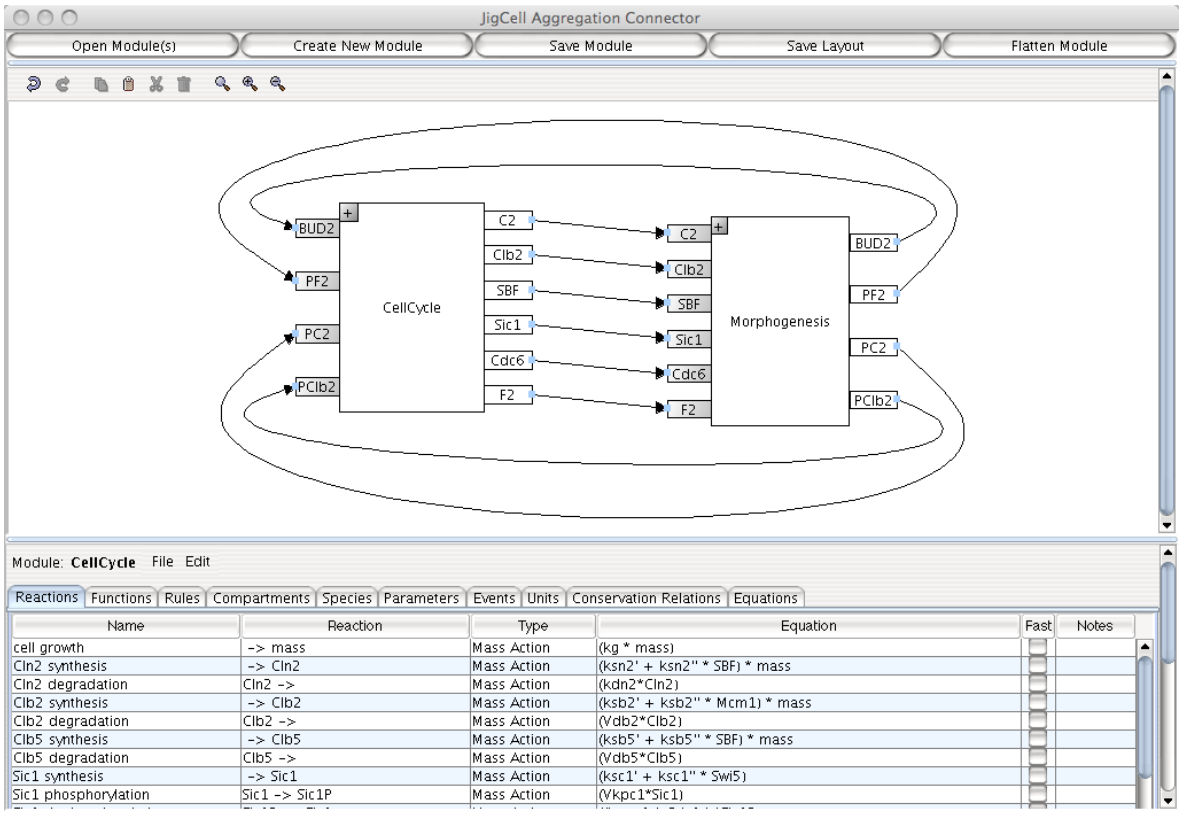


Figure 6.31: Screenshot of the $Aggregated\_Cell\_Morpho$ in the JigCell Aggregation Connector.

6.6 Summary

Sections 6.3, 6.4, and 6.5 produced the $Fused\_Cell\_Morpho$, $Composed\_Cell\_Morpho$, and $Aggregated\_Cell\_Morpho$ models, respectively. In order to judge whether the three model construction processes produced valid models, the modeler needed to verify, through simulation, that their dynamic properties represented the observed behavior of growing-dividing yeast cells in expected ways. This was accomplished by comparing the simulation results of the combined model to the results for the individual models published in [12] and [11]. The combined model was able to explain behavior that the two individual models could not explain by themselves. Since our current simulators require standard SBML (Level 2) input, the $Composed\_Cell\_Morpho$, and the $Aggregated\_Cell\_Morpho$ models needed to be flattened before they could be simulated. The JigCell flattening algorithm was used to automate this process and produced two single flat models that were then sent to a simulator.

6.6.1 Verification

Modeling and Simulation (M&S) Verification and Validation (V&V) typically focuses on the assessment of M&S transformational accuracy (verification) and representational/behavioral accuracy (validation) [5]. In this section we focus on transformational accuracy (verification) to verify the accuracy of our models, their simulation results, and the modeling processes used. In aggregation, no verification is performed as the modeler defines what to connect and how to connect. However, model testing activities are performed during aggregation to ensure a consistent model is produced. It should be noted that in this context consistent does not refer to biological consistency but rather whether the model follows the aggregation schema (e.g. input ports only containing a single linkage).

Model verification techniques can be applied to the fused and composed models. In this case the fused model is compared with the flattened version of the composed model, as the two processes are meant to give the same model. The two versions (fused and flattened composed model) will generate the same SBML Level2 model which can be verified manually or by checking the rate laws generated. Once again, both versions of the model will produce the same rate laws as the same information and transformations were provided by the modeler to produce the two models. Simulating the two models produces graphs which can be compared by graphical comparison (a V&V technique) by:

1. checking reflection points of the graphs
2. checking the skewness of the plots
3. superimposing the graphs

Yet another way to confirm that the results are not only are similar but match exactly is to generate a time series data from the simulation. Using time series data the modeler can quickly and definitively state whether the two simulation plots are identical.

6.6.2 Results

The two processes (fusion and composition plus flattening) are intended to produce identical results. The third process (aggregation plus flattening) is also intended to produce identical results depending on how the aggregated model is created and connected. Using time series data and the simulation output of the $Fused\_Cell\_Morpho$ model, the flattened version of the $Composed\_Cell\_Morpho$ model, and the flattened version of the $Aggregated\_Cell\_Morpho$ model, the modeler was able to accurately state that the simulation results of all three models exactly match with each other (Figures 6.32 and 6.33). In addition, the simulation output produced also closely matched the simulation output from Chen's (Figure 2 in [11]) and Ciliberto's model (Figure 3 in [12]) as expected, demonstrating that the results from the combined models are superior to the individual models.

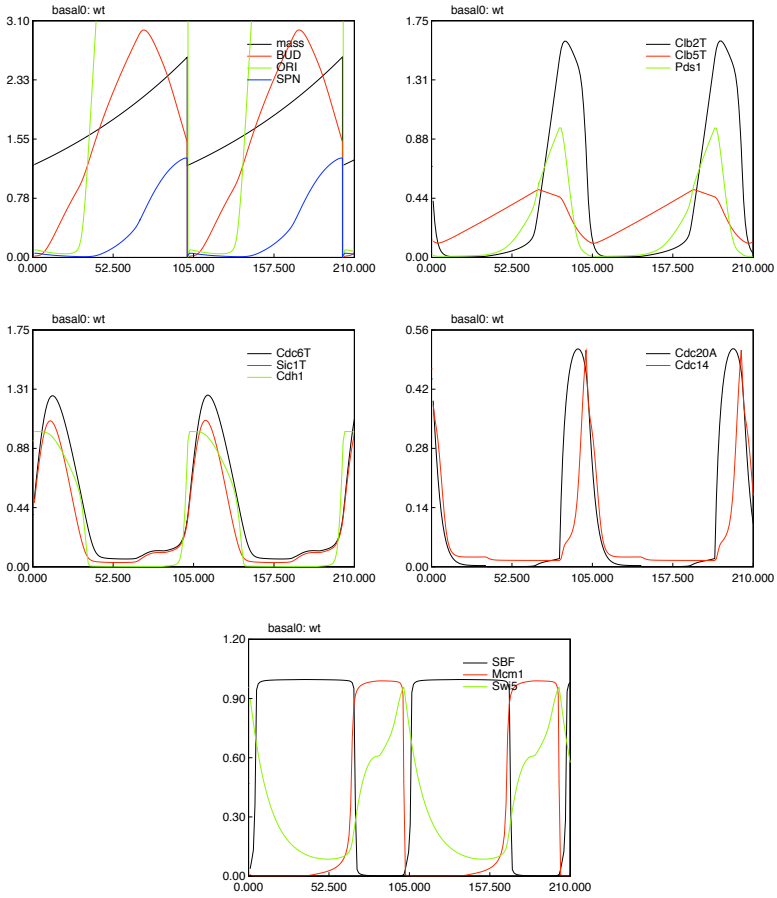


Figure 6.32: Simulation of the combined model using PET which closely matches Figure 2 in [11]. All three versions (the $Fused\_Cell\_Morpho$ model, the flattened version of the $Composed\_Cell\_Morpho$ model, and the flattened version of the $Aggregated\_Cell\_Morpho$ model) generated the same graphs. This was verified by confirming that the time series simulation output of the three models was identical.

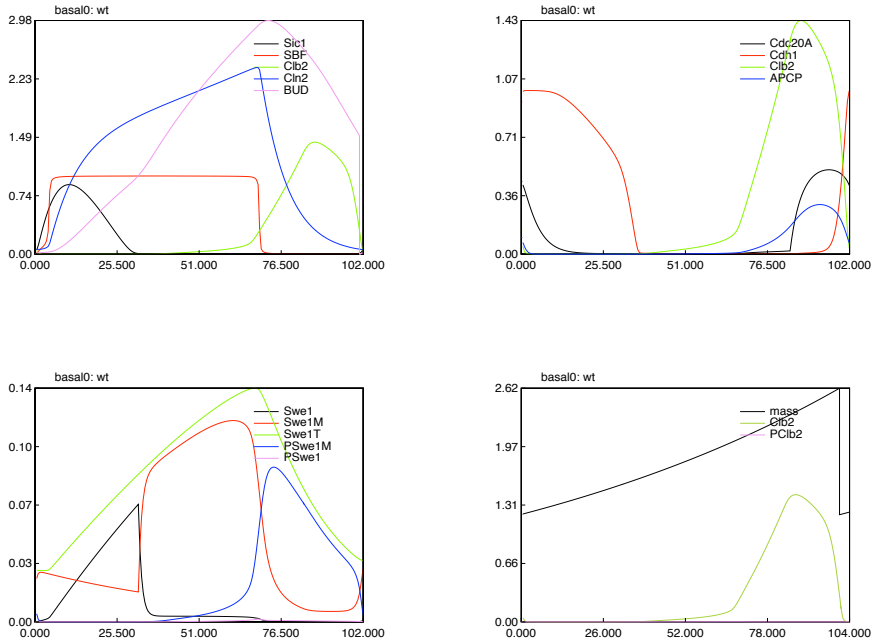


Figure 6.33: Further simulation of the combined model using PET which closely matches Figure 3 in [12]. All three versions (the $Fused\_Cell\_Morpho$ model, the flattened version of the $Composed\_Cell\_Morpho$ model, and the flattened version of the $Aggregated\_Cell\_Morpho$ model) generated the same graphs. This was verified by confirming that the time series simulation output of the three models was identical.

Our experiences lead us to believe that aggregation is a faster and easier way to create models than either fusion or composition. In fusion and composition, a lot of time and effort is used in identifying and dealing with redundancies across the (sub)models. In aggregation there are no such redundancies across submodels (or modules) as all submodels are customized before they are connected together. While it might take longer initially to create models for aggregation, once done, it is relatively straightforward to connect them together through their ports to create larger models. The expert modeler noted that composition was the slowest of the three options to create models. In fusion, only relevant and important information is added to the final model. By this we mean that only components that will be present in the fused model are handled by the modeler and explicitly added in the tool. In contrast, composition requires a modeler to document not just components that should be included in the composed model, but also components that should be excluded from the composed model. The slow down comes from the fact that composition strives to create a more complete record of model development by documenting the inclusion/exclusion of components within the composition hierarchy.

Chapter 7

Conclusions And Future Work

The computational software that this dissertation presents solves an open problem in model composition using pathway modeling. Model composition (and aggregation) was a particular bottleneck in the model development process and models were thus limited in their size and scope. Alleviation of the model composition (and aggregation) bottleneck should lead to the construction of larger and more complex pathway models than previously possible and a deeper understanding of biological systems under investigation.

This dissertation describes the construction of large models of biochemical reaction networks through hierarchical composition. Using previously existing modeling tools, a skilled modeler could create accurate mathematical models of biological systems with moderate complexity and gain an understanding of dynamical processes. However, this method for building complex models is already reaching its limits of practicality. The ability to construct large biological models provides the potential for better insights into the physiological properties of cells, if only we can handle the complexity involved. The level of complexity is difficult to deal with as the number of models and their sizes increase.

Further progress in biological modeling requires the use of better tools and modeling processes, as existing modeling processes and tools do not scale to models of this size. Investigations of complex eukaryotic organisms requires models at least an order of magnitude larger than the current state of the art. This dissertation describes how to fuse, compose, and aggregate collections of models to create models of greater complexity than previously possible.

The remainder of this section summarizes our major contributions and conclusions. Section 7.1 summarizes the contributions of this dissertation both in terms of the modeling community, the modeling process, and the SBML language. Section 7.2 examines future avenues of research.

Contents

7.1 Contributions

This research provides the following contributions to the modeling process for regulatory networks and the SBML modeling community.

Extends the ability of modelers to construct larger models

Chapters 3, 4, and 5 described three distinct modeling processes to extend the ability of modelers to decompose models and use the resulting components to construct larger models. Chapter 6 provided a practical demonstration that these novel modeling processes can help biological modelers build models. The example we provided in Chapter 6 is already more sophisticated than previous examples of composed or aggregated models, and the resulting model is clearly larger than any hand-constructed pathway model. The results of this case study demonstrate that these approaches are applicable to real models, reduce development time for models at the limit to the state of the art in biology, and work for large-scale models with complex outputs.

Provides modeling software that can handle and combine large-scale biological models

We have developed working implementations that demonstrate the feasibility of fusion, composition, and aggregation. Sections 3.2, 4.3, 4.6.1, and 5.3 described the additions to the JigCell modeling environment for building models of reaction networks. The biological modeling community needed tools for efficiently, reliably, and repeatably building (and combining) large models of biochemical reaction networks. The new JigCell applications support combining larger models than comparable modeling tools, making these JigCell applications especially suitable for particular modeling efforts.

Maintains information about inter-relations between models

The SBML language features described in Sections 4.1 and 5.1 indirectly maintain a record of how models are assembled. This record explicitly documents overlaps and redundancies which are stored implicitly in the additional language features (or "glue") of the composed or aggregated forms of a model. The overall structure of a composed or aggregated model provide the following information and features not present in flattened versions of the same model:

1. How the composed or aggregated model is constructed from smaller components.
2. The ability to deconstruct a composed or aggregated model into its distinct subparts.
3. The amount of overlap between components.

Provides an ability to reuse existing models

The arrangement of composed and aggregated models naturally favors model reuse. Models, submodels, and modules can all be reused either within the current composed or aggregated model they reside in or within

other models in different SBML files. This is possible due to the use of the <instance> structures which enable modelers to use the same model more than once in a composed or aggregated model.

Provides backwards compatibility to use existing tools with composed or aggregated models

Complex models can be created, stored, updated and maintained in their composed or aggregated forms. Modelers can make use of a wide variety of existing tools for analysis, simulation, etc by flattening their composed or aggregated models into their respective flattened forms. The backwards compatibility ensures that composed and aggregated models can be developed in conjunction with older tools that provide little or no support for these types of models.

Worked with SBML community to develop model composition (and aggregation) that ensures this is a practical solution

This dissertation has provided a real solution to model composition in regulatory network modeling. The SBML language features described in Sections 4.1 and 5.1 provide an implementation-independent framework for managing the complexity when building models. We have interacted with the SBML community to standardize our extensions. We are working with the SBML editors (in particular, Stephan Hoops) to document our solutions and approaches into the SBML Level3 proposal for Hierarchical Modeling (Appendix B), using our experiences and processes with respect to model composition and aggregation.

7.1.1 Work Completed

1. Fusion
   (a) Implemented in the Fusion Wizard
   (b) Distributed as part of the JigCell package.
2. Composition
   (a) Implemented and combined with the Fusion Wizard to create the Fusion/Composition Wizard
   (b) Distributed as part of the JigCell package.
3. Aggregation
   (a) Implemented in the Aggregation Connector
   (b) Distributed as part of the JigCell package.
4. Flattening
   (a) Implemented as the Flattening algorithm in the JigCell SBML parser library
   (b) Distributed as part of the JigCell package.
5. SBML
   (a) Formed the bulk of the official SBML Proposal for composition and aggregation titled Hierarchical Modeling.

(b) Got proposals implemented into SBML

6. Use Cases: Fused, Composed and Aggregated the following use cases successfully.
   (a) Tyson and Novak 2001 Basic Cell Cycle Model
   (b) Chen 2004 cell cycle model and Ciliberto 2003 morphogenesis checkpoint model.

7.2 Future Work

The ideas, processes, frameworks, and practices described in this dissertation provide a solid foundation for future avenues of research. Developing composed (and aggregated) models with the existing state of the art fails to address one major issue: When can a modeler can reliably and with some confidence state they are done? Introducing more detailed and developed verification and validation activities to improve the credibility and reliability of a model will greatly benefit the composition and aggregation processes. A checker tool that validates the results of composition and aggregation and provides some metrics and measurements when composition and aggregation have been accurately completed would provide a quantifiable demonstration that the model construction processes and the software applications developed help biological modelers build models.

The motivation for solving composition and aggregation was to provide a framework for dealing with complexity when model size increased. However what happens when the size and complexity of composed or aggregated models also increases beyond some viable point? This situation is imminently possible as composed and aggregated models can become fairly large and verbose, due to all the additional SBML language constructs. Developing a short-hand (human readable) notation for composed and aggregated models would provide an added layer of abstraction between the model and the modeler. The short-hand notation should ensure that there is no significant loss of information when reducing the verbosity of composed or aggregated models.

Model construction processes would greatly benefit from some form of automation. While an automated form of fusion, composition, or aggregation was not within the scope of this dissertation, there exist novel efforts within the systems biology community that focus on model curation and annotation of quantitative models of biological systems, such as MIRIAM (Minimum Information Requested in the Annotation of biochemical Models) [45] and BioModels [44], that might someday make this a real possibility. Even a limited form of automation would provide two main advantages over current practices. First model development time would decrease, and second, model verification activities would benefit greatly from curation and annotation standards.

Enhancing our applications and frameworks would provide a better modeling experience both to the novice and expert modeler. Three important areas of improvement are providing interoperability between SBML and other modeling languages, providing better integration between fusion and composition (in the Fusion/Composition Wizard) and model construction (in the ModelBuilder), and extending libSBML [8]. Facilitating interoperability between SBML and other modeling languages, such as CellML [35] would open

a whole modeling community to the advances made in this dissertation. In this way a composed or aggregated model might someday be constructed from both SBML and CellML models. The obvious benefit would be the construction of even more complex models than previously possible, that describe biological processes in terms of their biochemistry and physiology. Tighter integration of JigCell applications would provide a better model construction experience. Currently, fusion and composition are separate and distinct processes with respect to model construction. A tighter association between the Fusion/Composition Wizard and the ModelBuilder would enable a modeler to perform both tasks in one environment and thus would greatly speed up the model development process. The language features presented in this dissertation should be added to libSBML, an application programming interface library for reading, writing, manipulating and validating content expressed in SBML in order to gain the largest audience possible within the systems biology community.

In the immediate future modelers in John Tyson's Group at Virginia Tech will continue the efforts started in the case study (Chapter 6) by adding additional components to the combined models of Chen [11] and Ciliberto [12] and by making existing models more suitable for stochastic simulation by automatically converting deterministic models into stochastic models [76]. Extending the large-scale combined model developed in the case study by adding more components, such as a polo kinase module and a osmotic shock module, and by creating stochastic formulations of deterministic models will produce a more complete picture of the underlying biology in budding yeast.

Bibliography

[1] S. Addanki, R. Cremonini, and J.S. Penberthy. Graphs of models. *Artificial Intelligence*, 51(1–3):145–177, 1991.

[2] 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*, 7(3):285–299, 2003.

[3] J. Aronson and P. Bose. A model-based approach to simulation composition. In *Proceedings of the 1999 Symposium on Software Reusability*, pages 73–82, 1999.

[4] O. Balci. Verification, validation, and accreditation. In *Proc. 1998 Winter Simulation Conf.*, pages 41–48, 1998.

[5] O. Balci. Quality assessment, verification, and validation of modeling and simulation applications. In *Proc. 2004 Winter Simulation Conf.*, pages 122–129, 2004.

[6] O. Balci and R.G. Sargent. A Methodology for Cost-Risk Analysis in the Statistical Validation of Simulation Models. *Communications of the ACM*, 24(4):190–197, 1981.

[7] Darpa BioSPICE website. `http://community.biospice.org`, 2005.

[8] B.J. Bornstein, S.M. Keating, A. Jouraku, and M. Hucka. LibSBML: An API Library for SBML. *Bioinformatics*, pages btn051+, February 2008.

[9] T. Bulatewicz, J. Cuny, and M. Warman. The potential coupling interface: Metadata for model coupling. In *Proc. 2004 Winter Simulation Conf.*, pages 183–190, 2004.

[10] Y. Cao, H. Li, and L. Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *J. Chem. Phys.*, 121:4059–4067, 2004.

[11] K.C. Chen, L. Calzone, A. Csikasz-Nagy, F.R. Cross, B. Novak, and J.J. Tyson. Integrative analysis of cell cycle control in budding yeast. *Mol. Biol. Cell*, 15:3841–3862, 2004.

[12] A. Ciliberto, B. Novak, and J.J. Tyson. Mathematical model of the morphogenesis checkpoint in budding yeast. *J. Cell Biol.*, 163:1243–1254, 2003.

[13] A. Csikasz-Nagy, D. Battogtokh, K.C. Chen, B. Novak, and J.J. Tyson. Analysis of a generic model of eukaryotic cell-cycle regulation. *Biophys. J.*, 90(12):4361–4379, 2006.

[14] P.K. Davis and R.H. Anderson. Improving the composability of DoD models and simulations. *J. Defense Modeling and Simulation*, 1(1):5–17, 2004.

[15] S. DeRose, E. Maler, and D. Orchard. XML Linking Language (XLink) Version 1.0 W3C Recommendation. Available at `http://www.w3.org/TR/xlink`, 2001.

[16] D.T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.*, 115:1716–1733, 2001.

[17] B. Falkenhainer and K.D. Forbus. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51(1–3):95–143, 1991.

[18] A. Finney. Systems Biology Markup Language (SBML) Level 3 Proposal: Model Composition Features. Available at `http://www.sbml.org/forums/index.php?t=tree&goto=171&rid=0`, 2003.

[19] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch or why it's hard to build systems out of existing parts. In *Intl. Conf. on Softw. Eng.*, pages 179–185, 1995.

[20] M. Ginkel. Modular SBML Proposal for an Extension of SBML towards Level 2. In *Proc. 5^{th} Forum on Software Platforms for Systems Biology*, 2003.

[21] M. Ginkel, A. Kremling, T. Nutsch, R. Rehner, and E.D. Gilles. Modular modeling of cellular systems with ProMoT/Diva. *Bioinformatics*, 19(9):1169–1176, 2003.

[22] P. Grosso, E. Maler, J. Marsh, and N. Walsh. XPointer Framework. Available at `http://www.w3.org/TR/xptr-framework/`, 2003.

[23] L.H. Hartwell, J.J. Hopfield, S. Leibler, and A.W. Murray. From molecular to modular cell biology. *Nature*, 402:C47–C52, 1999.

[24] A.C. Hindmarsh. ODEPACK: A systematized collection of ODE solvers. In R.S. Stepleman, editor, *Scientific Computing*, pages 55–64. North Holland Publishing Company, 1983.

[25] 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

Novére, 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.

[26] Y. Iwasaki. Causal ordering in a mixed structure. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 313–318, August 1988.

[27] JigCell website. `http://jigcell.biol.vt.edu/`, 2007.

[28] S. Kasputis and H.C. Ng. Model composability: Formulating a research thrust: Composable simulations. In *Proc. 2000 Winter Simulation Conf.*, pages 1577–1584, 2000.

[29] E. Klipp, B. Nordlander, R. Kruger, P. Gennemark, and S. Hohmann. Integrative model of the response of yeast to osmotic shock. *Nat. Biotechnol.*, 23(8):975–982, 2005.

[30] B. Kofahl and E. Klipp. Modelling the dynamics of the yeast pheromone pathway. *Yeast*, 21(10):831–850, 2004.

[31] K.W. Kohn. Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Mol. Biol. Cell*, 10(8):2703–2734, 1999.

[32] D.A. Lauffenburger. Cell signaling pathways as control modules: Complexity for simplicity? *Proceedings of the National Academy of Science*, 97(10):5031–5033, 2000.

[33] A.Y. Levy, Y. Iwasaki, and R. Fikes. Automated model selection for simulation based on relevance reasoning. *Artificial Intelligence*, 96(2):351–394, 1997.

[34] V.C. Liang and C.J.J. Paredis. Foundations of multi-paradigm modeling and simulation: A port ontology for automated model composition. In *Proc. 2003 Winter Simulation Conf.*, pages 613–622, 2003.

[35] C.M. Lloyd, M.D.B. Halstead, and P.F. Nielsen. CellML: its future, present and past. *Progress in Biophysics and Molecular Biology*, 85(2-3):433–450, 2004.

[36] M. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, 104:1876–1889, 2000.

[37] R.J. Malak and C.J.J. Paredis. Foundations of validating reusable behavioral models in engineering design problems. In *Proc. 2004 Winter Simulation Conf.*, pages 420–428, 2004.

[38] G. Marlovits, C.J. Tyson, B. Novak, and J.J. Tyson. Modeling M-phase control in Xenopus oocyte extracts: the surveillance mechanism for unreplicated DNA. *Biophysical Chemistry*, 72:169–184, 1998.

[39] P. Mendes. Biochemistry by numbers: Simulation of biochemical pathways with Gepasi 3. *Trends in Biochem. Sci.*, 22:361–363, 1997.

[40] Microsoft. Microsoft Hails 10 Years of Publisher. Available at `http://www.microsoft.com/presspass/press/2001/oct01/10-15TenYearsPublisherPR.mspx`, 2005.

[41] P.P. Nayak. Casual approximations. *Artificial Intelligence*, 70(1–2):277–334, 1994.

[42] P.P. Nayak and L. Joskowicz. Efficient compositional modeling for generating causal explanations. *Artificial Intelligence*, 83(2):193–227, 1996.

[43] B. Novak and J.J. Tyson. Modelling the controls of the eukaryotic cell cycle. *Biochem. Soc. Trans.*, 31(6):1526–1529, 2003.

[44] N. Le Novère, B. Bornstein, A. Broicher, M. Courtot, M. Donizelli, H. Dharuri, L. Li, H. Sauro, M. Schilstra, B. Shapiro, J.L. Snoep, and M. Hucka. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Res.*, 34:D689–D691, 2006.

[45] N. Le Novère, A. Finney, M. Hucka, U. Bhalla, F. Campagne, J. Collado-Vides, E. Crampin, M. Halstead, E. Klipp, P. Mendes, P. Nielsen, H. Sauro, B. Shapiro, J.L. Snoep, H.D. Spence, and B.L. Wanner. Minimum information requested in the annotation of biochemical models (MIRIAM). *Nature Biotechnology*, 23(12):1509–1515, 2005.

[46] US Department of Energy. Genomes to Life website. Available at `http://doegenomestolife.org/`, 2005.

[47] E.H. Page, R. Briggs, and J.A. Tufarolo. Toward a familiy of maturity models for the simulation interconnection problem. In *Proceedings of the 2004 Spring Simulation Interoperability Workshop*, pages 18–23, 2004.

[48] E.H. Page and J.M. Opper. Observations on the complexity of composable simulation. In *Winter Simulation Conference*, pages 553–560, 1999.

[49] T.D. Panning, L.T. Watson, C.A. Shaffer, and J.J. Tyson. A Mathematical Programming Formulation for the Budding Yeast Cell Cycle. *Simulation*, 83(7):497–514, 2007.

[50] Pathwaybuilder website. `http://biospice.lbl.gov/PathwayBuilder/`, 2003.

[51] M.D. Petty and E.W. Weisel. A composability lexicon. In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, April 2003.

[52] M.D. Petty and E.W. Weisel. A formal basis for a theory of semantic composability. In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, September 2003.

[53] M.D. Petty, E.W. Weisel, and R.R. Mielke. Computational complexity of selecting components. for composition. In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, pages 517–525, September 2003.

[54] M.D. Petty, E.W. Weisel, and R.R. Mielke. A formal approach to composability. In *Proceedings of the 2003 Interservice/Industry Training, Simulation and Education Conference (I/ITSEC)*, December 2003.

[55] Promotdiva website. `http://www.mpi-magdeburg.mpg.de/research/projects/1002/comp_bio/promot/distrib`, 2002.

[56] R. Randhawa, C.A. Shaffer, and J.J. Tyson. Fusing and composing macromolecular regulatory network models. In *Proc. 2007 High Performance Computing Symp.*, pages 337–344, 2007.

[57] R. Randhawa, C.A. Shaffer, and J.J. Tyson. Model composition in macromolecular regulatory networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2008.

[58] R. Randhawa, C.A. Shaffer, and J.J. Tyson. Model aggregation: a building-block approach to creating large macromolecular regulatory networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, submitted.

[59] J. Rickel and B. Porter. Automated modeling for answering prediction questions: Selecting the time scale. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1191–1198, August 1994.

[60] J. Rickel and B. Porter. Automated modeling of complex systems to answer prediction questions. *Artificial Intelligence*, 93(1–2):201–260, 1997.

[61] R.G. Sargent. Validation and verification of simulation models. In *Proc. 2003 Winter Simulation Conf.*, pages 17–28, 2004.

[62] H.M. Sauro, M. Hucka, A. Finney, C. Wellock, H. Bolouri, J. Doyle, and H. Kitano. Next generation simulation tools: The Systems Biology Workbench and BioSPICE integration. *OMICS*, 7:355–372, Winter 2003.

[63] J.C. Schaff, B.M. Slepchenko, Y. Choi, J.M. Wagner, D. Resasco, and L.M. Loew. Analysis of Non-Linear Dynamics on Arbitrary Geometries with the Virtual Cell. *Chaos*, 11:115–131, 2001.

[64] D. Schroder and J. Weimar. Modularization of SBML. Available at `http://www.sbml.org/workshops/ninth/VortragSBMLForum.pdf`, 2003.

[65] M. Schulz, J. Uhlendorf, E. Klipp, and W. Liebermeister. SBMLmerge, a system for combining biochemical network models. *Genome Informatics*, 17(1):62–71, 2006.

[66] C.A. Shaffer, R. Randhawa, and J.J. Tyson. The role of composition and aggregation in modeling macromolecular regulatory networks. In *Proc. 2006 Winter Simulation Conf.*, December 2006.

[67] J.C. Sible and J.J. Tyson. Mathematical modeling as a tool for investigating cell cycle control networks. *Methods*, 41(2):238–247, 2007.

[68] J.L. Snoep, F. Bruggeman, B.G. Olivier, and H.V. Westerhoff. Towards building the silicon cell: A modular approach. *Biosystems*, 83:207–216, 2006.

[69] M. Spiegel, P.F. Reynolds, and D.C. Brogan. A case study of model context for simulation composability and reusability. In *Proc. 2005 Winter Simulation Conf.*, pages 437–444, 2005.

[70] K. Takahashi, N. Ishikawa, Y. Sadamoto, H. Sasamoto, S. Ohta, A. Shiozawa, F. Miyoshi, Y. Naito, Y. Nakayama, and M. Tomita. E-Cell 2: Multi-platform E-Cell simulation system. *Bioinformatics*, 19:1727–1729, 2003.

[71] Teranode design suite website. `http://teranode.com/products/index.php`, 2005.

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

[73] J.J. Tyson, K.C. Chen, and B. Novak. Sniffers, buzzers, toggles and blinkers: Dynamics of regulatory and signaling pathways in the cell. *Curr Opin Cell Biol*, 15(2):221–231, 2003.

[74] J.J. Tyson and B. Novak. Regulation of the eukaryotic cell cycle: Molecular antagonism, hysteresis, and irreversible transitions. *J. Theoretical Biology*, 210:249–263, 2001.

[75] 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 Trans. Comput. Biol. Bioinform.*, 3(2):155–164, 2006.

[76] P. Wang, R. Randhawa, C.A. Shaffer, Y. Cao, and B. Baumann. Automatic conversion of macromolecular regulatory models from deterministic to stochastic formulation. In *Proceedings of the 2008 High Performance Computing Symposium (HPCS 2008)*, 2008.

[77] J. Webb. BioSpice MDL Model Composition and Libraries. Available at `http://bio.bbn.com/biospice/mdl/design/compose.html`, 2003.

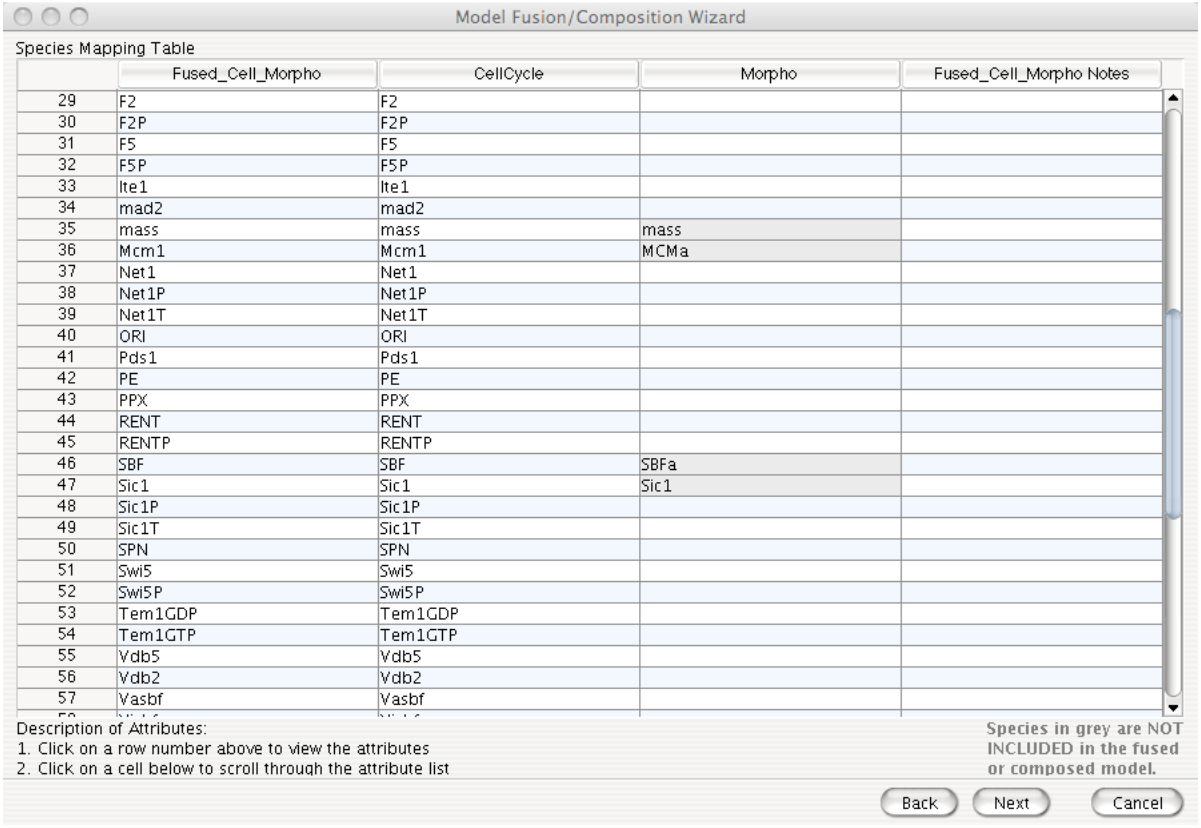
[78] E.W. Weisel, M.D. Petty, and R.R. Mielke. Validity of models and classes of models in semantic composability. In *Proceedings of the Fall 2003 Simulation Interoperability Workshop*, pages 526–536, September 2003.

[79] L. Yilmaz. On the need for contextualized introspective simulation models to improve reuse and composability of defense simulations. *Journal of Defense Modeling and Simulation*, 1(3):135–145, 2004.

[80] L. Yilmaz and T.I. Oren. Prospective issues in simulation model composability: Basic concepts to advance theory, methodology, and technology. *MSIAC's M&S Journal Online*, 6(3), 2004.

Appendix A

Fusion/Composition Wizard Screenshots



	Fused_Cell_Morpho	CellCycle	Morpho	Fused_Cell_Morpho Notes
29	F2	F2		
30	F2P	F2P		
31	F5	F5		
32	F5P	F5P		
33	Ite1	Ite1		
34	mad2	mad2		
35	mass	mass	mass	
36	Mcm1	Mcm1	MCMa	
37	Net1	Net1		
38	Net1P	Net1P		
39	Net1T	Net1T		
40	ORI	ORI		
41	Pds1	Pds1		
42	PE	PE		
43	PPX	PPX		
44	RENT	RENT		
45	RENTP	RENTP		
46	SBF	SBF	SBFa	
47	Sic1	Sic1	Sic1	
48	Sic1P	Sic1P		
49	Sic1T	Sic1T		
50	SPN	SPN		
51	Swi5	Swi5		
52	Swi5P	Swi5P		
53	Tem1GDP	Tem1GDP		
54	Tem1GTP	Tem1GTP		
55	Vdb5	Vdb5		
56	Vdb2	Vdb2		
57	Vasbf	Vasbf		

Figure A.1: 2^{nd} of 3 screenshots of the species mapping table of the morphogenesis and cell cycle combined model

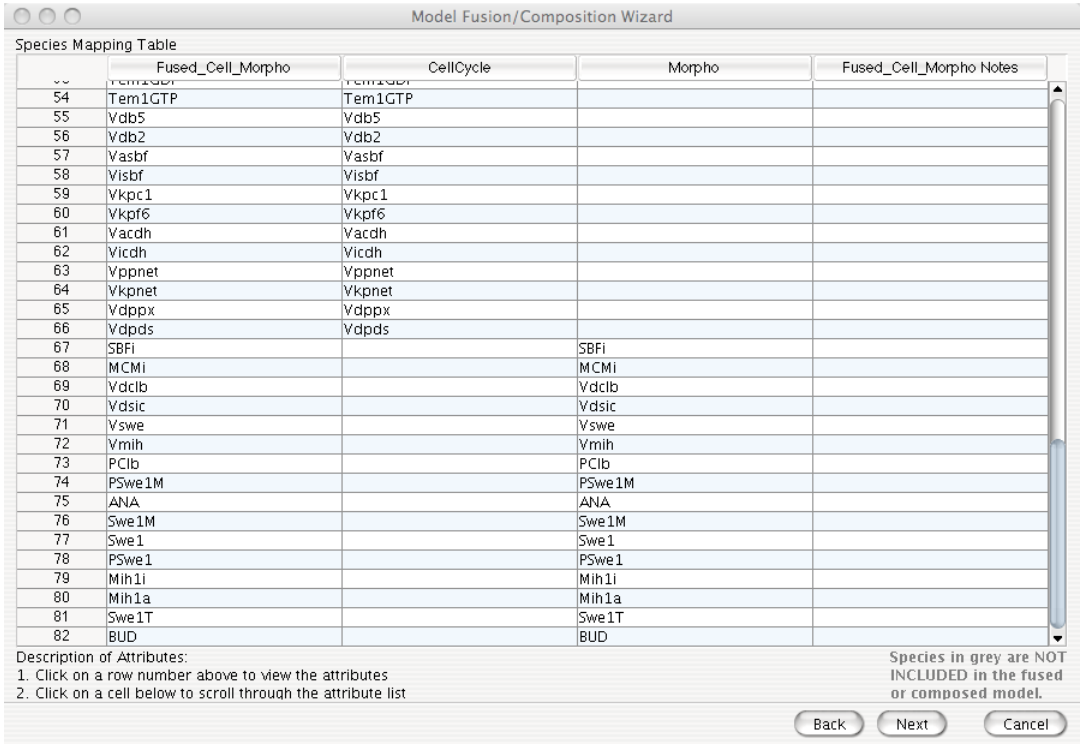


Figure A.2: 3^{rd} of 3 screenshots of the species mapping table of the combined model

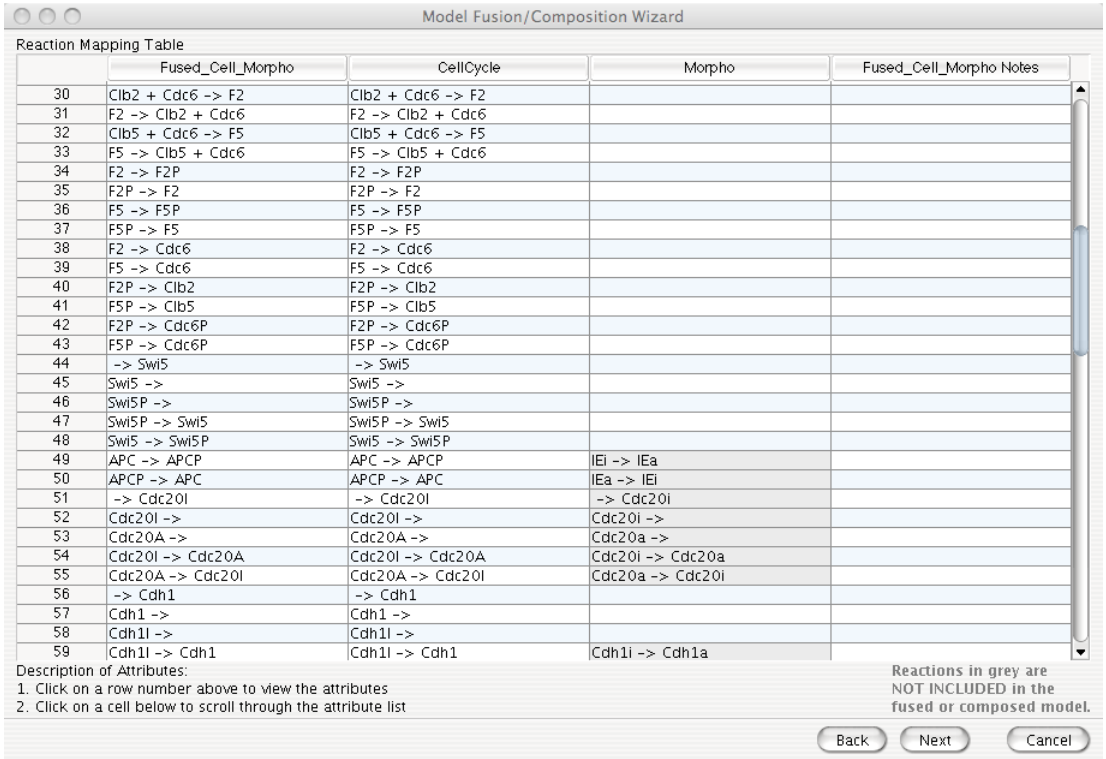
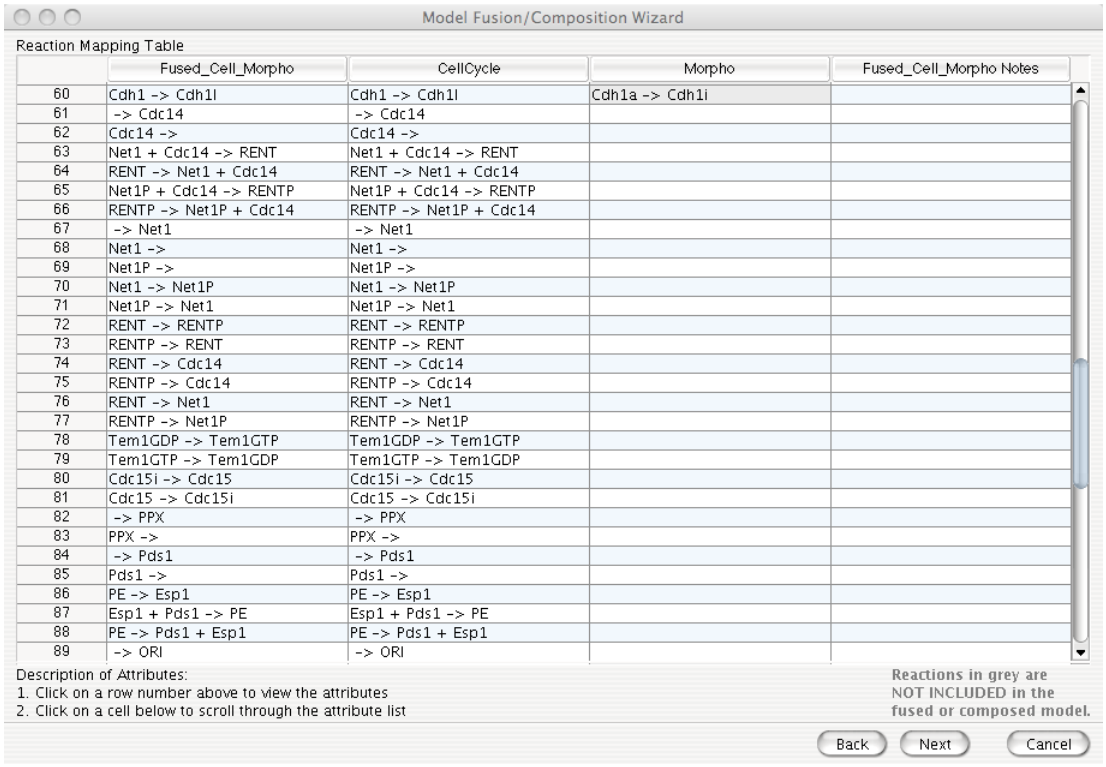
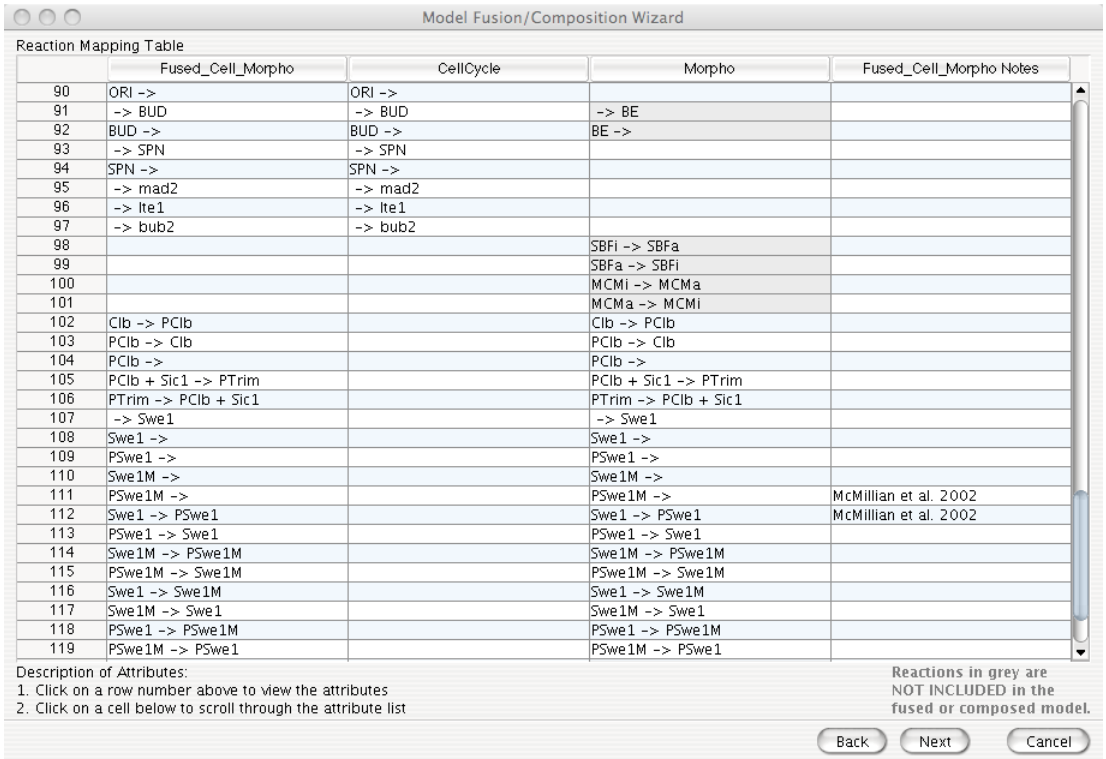


Figure A.3: 2^{nd} of 5 screenshots of the reaction mapping table of the morphogenesis and cell cycle combined model



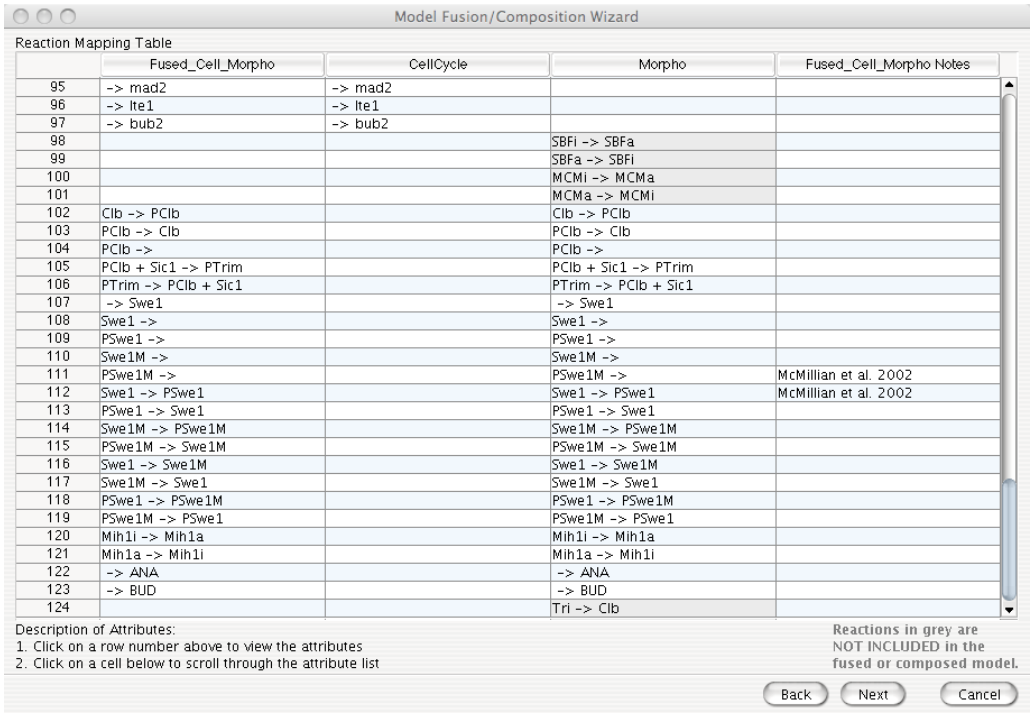
	Fused_Cell_Morpho	CellCycle	Morpho	Fused_Cell_Morpho Notes
60	Cdh1 -> Cdh1I	Cdh1 -> Cdh1I	Cdh1a -> Cdh1i	
61	-> Cdc14	-> Cdc14		
62	Cdc14 ->	Cdc14 ->		
63	Net1 + Cdc14 -> RENT	Net1 + Cdc14 -> RENT		
64	RENT -> Net1 + Cdc14	RENT -> Net1 + Cdc14		
65	Net1P + Cdc14 -> RENTP	Net1P + Cdc14 -> RENTP		
66	RENTP -> Net1P + Cdc14	RENTP -> Net1P + Cdc14		
67	-> Net1	-> Net1		
68	Net1 ->	Net1 ->		
69	Net1P ->	Net1P ->		
70	Net1 -> Net1P	Net1 -> Net1P		
71	Net1P -> Net1	Net1P -> Net1		
72	RENT -> RENTP	RENT -> RENTP		
73	RENTP -> RENT	RENTP -> RENT		
74	RENT -> Cdc14	RENT -> Cdc14		
75	RENTP -> Cdc14	RENTP -> Cdc14		
76	RENT -> Net1	RENT -> Net1		
77	RENTP -> Net1P	RENTP -> Net1P		
78	Tem1GDP -> Tem1GTP	Tem1GDP -> Tem1GTP		
79	Tem1GTP -> Tem1GDP	Tem1GTP -> Tem1GDP		
80	Cdc15i -> Cdc15	Cdc15i -> Cdc15		
81	Cdc15 -> Cdc15i	Cdc15 -> Cdc15i		
82	-> PPX	-> PPX		
83	PPX ->	PPX ->		
84	-> Pds1	-> Pds1		
85	Pds1 ->	Pds1 ->		
86	PE -> Esp1	PE -> Esp1		
87	Esp1 + Pds1 -> PE	Esp1 + Pds1 -> PE		
88	PE -> Pds1 + Esp1	PE -> Pds1 + Esp1		
89	-> ORI	-> ORI		

Figure A.4: 3^{rd} of 5 screenshots of the reaction mapping table of the combined model



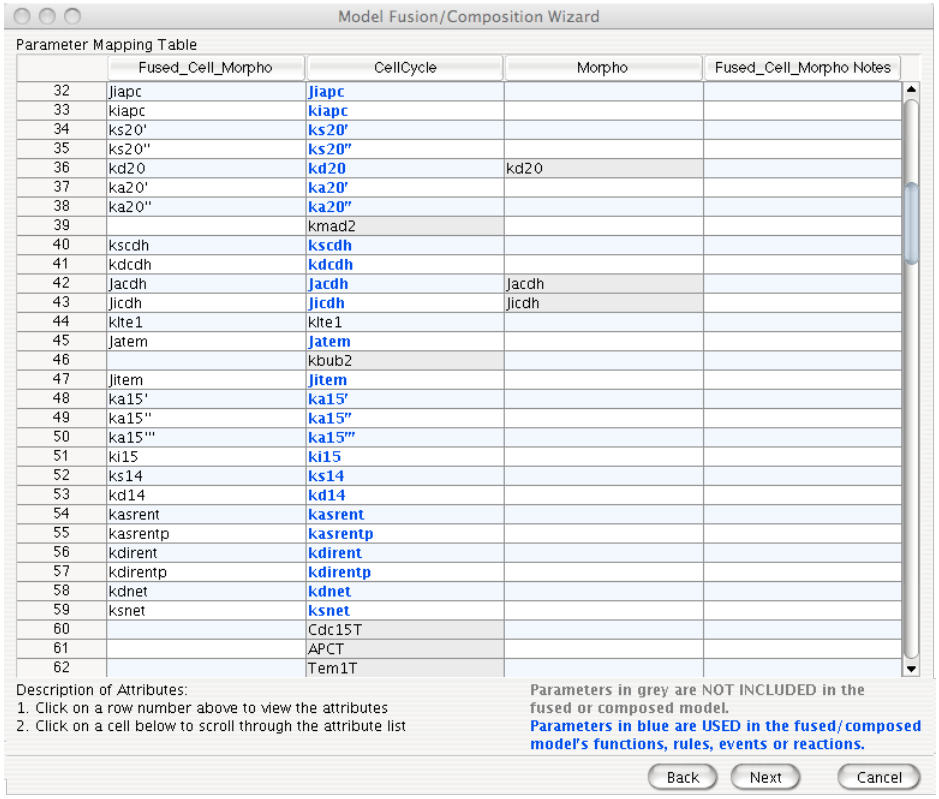
	Fused_Cell_Morpho	CellCycle	Morpho	Fused_Cell_Morpho Notes
90	ORI ->	ORI ->		
91	-> BUD	-> BUD	-> BE	
92	BUD ->	BUD ->	BE ->	
93	-> SPN	-> SPN		
94	SPN ->	SPN ->		
95	-> mad2	-> mad2		
96	-> Ite1	-> Ite1		
97	-> bub2	-> bub2		
98			SBFi -> SBFa	
99			SBFa -> SBFi	
100			MCMi -> MCMa	
101			MCMa -> MCMi	
102	Clb -> PClb		Clb -> PClb	
103	PClb -> Clb		PClb -> Clb	
104	PClb ->		PClb ->	
105	PClb + Sic1 -> PTrim		PClb + Sic1 -> PTrim	
106	PTrim -> PClb + Sic1		PTrim -> PClb + Sic1	
107	-> Swe1		-> Swe1	
108	Swe1 ->		Swe1 ->	
109	PSwe1 ->		PSwe1 ->	
110	Swe1M ->		Swe1M ->	
111	PSwe1M ->		PSwe1M ->	McMillian et al. 2002
112	Swe1 -> PSwe1		Swe1 -> PSwe1	McMillian et al. 2002
113	PSwe1 -> Swe1		PSwe1 -> Swe1	
114	Swe1M -> PSwe1M		Swe1M -> PSwe1M	
115	PSwe1M -> Swe1M		PSwe1M -> Swe1M	
116	Swe1 -> Swe1M		Swe1 -> Swe1M	
117	Swe1M -> Swe1		Swe1M -> Swe1	
118	PSwe1 -> PSwe1M		PSwe1 -> PSwe1M	
119	PSwe1M -> PSwe1		PSwe1M -> PSwe1	

Figure A.5: 4^{th} of 5 screenshots of the reaction mapping table of the combined model



	Fused_Cell_Morpho	CellCycle	Morpho	Fused_Cell_Morpho Notes
95	-> mad2	-> mad2		
96	-> Ite1	-> Ite1		
97	-> bub2	-> bub2		
98			SBFi -> SBFa	
99			SBFa -> SBFi	
100			MCMi -> MCMa	
101			MCMa -> MCMi	
102	Clb -> PClb		Clb -> PClb	
103	PClb -> Clb		PClb -> Clb	
104	PClb ->		PClb ->	
105	PClb + Sic1 -> PTrim		PClb + Sic1 -> PTrim	
106	PTrim -> PClb + Sic1		PTrim -> PClb + Sic1	
107	-> Swe1		-> Swe1	
108	Swe1 ->		Swe1 ->	
109	PSwe1 ->		PSwe1 ->	
110	Swe1M ->		Swe1M ->	
111	PSwe1M ->		PSwe1M ->	McMillian et al. 2002
112	Swe1 -> PSwe1		Swe1 -> PSwe1	McMillian et al. 2002
113	PSwe1 -> Swe1		PSwe1 -> Swe1	
114	Swe1M -> PSwe1M		Swe1M -> PSwe1M	
115	PSwe1M -> Swe1M		PSwe1M -> Swe1M	
116	Swe1 -> Swe1M		Swe1 -> Swe1M	
117	Swe1M -> Swe1		Swe1M -> Swe1	
118	PSwe1 -> PSwe1M		PSwe1 -> PSwe1M	
119	PSwe1M -> PSwe1		PSwe1M -> PSwe1	
120	Mih1i -> Mih1a		Mih1i -> Mih1a	
121	Mih1a -> Mih1i		Mih1a -> Mih1i	
122	-> ANA		-> ANA	
123	-> BUD		-> BUD	
124			Tri -> Clb	

Figure A.6: 5^{th} of 5 screenshots of the reaction mapping table of the combined model



	Fused_Cell_Morpho	CellCycle	Morpho	Fused_Cell_Morpho Notes
32	Jiapc	**Jiapc**		
33	kiapc	**kiapc**		
34	ks20'	**ks20'**		
35	ks20''	**ks20''**		
36	kd20	**kd20**	kd20	
37	ka20'	**ka20'**		
38	ka20''	**ka20''**		
39		kmad2		
40	kscdh	**kscdh**		
41	kdcdh	**kdcdh**		
42	Jacdh	**Jacdh**	Jacdh	
43	Jicdh	**Jicdh**	Jicdh	
44	klte1	klte1		
45	Jatem	**Jatem**		
46		kbub2		
47	Jitem	**Jitem**		
48	ka15'	**ka15'**		
49	ka15''	**ka15''**		
50	ka15'''	**ka15'''**		
51	ki15	**ki15**		
52	ks14	**ks14**		
53	kd14	**kd14**		
54	kasrent	**kasrent**		
55	kasrentp	**kasrentp**		
56	kdirent	**kdirent**		
57	kdirentp	**kdirentp**		
58	kdnet	**kdnet**		
59	ksnet	**ksnet**		
60		Cdc15T		
61		APCT		
62		Tem1T		

Figure A.7: 2^{nd} of 7 screenshots of the parameter mapping table of the morphogenesis and cell cycle combined model.

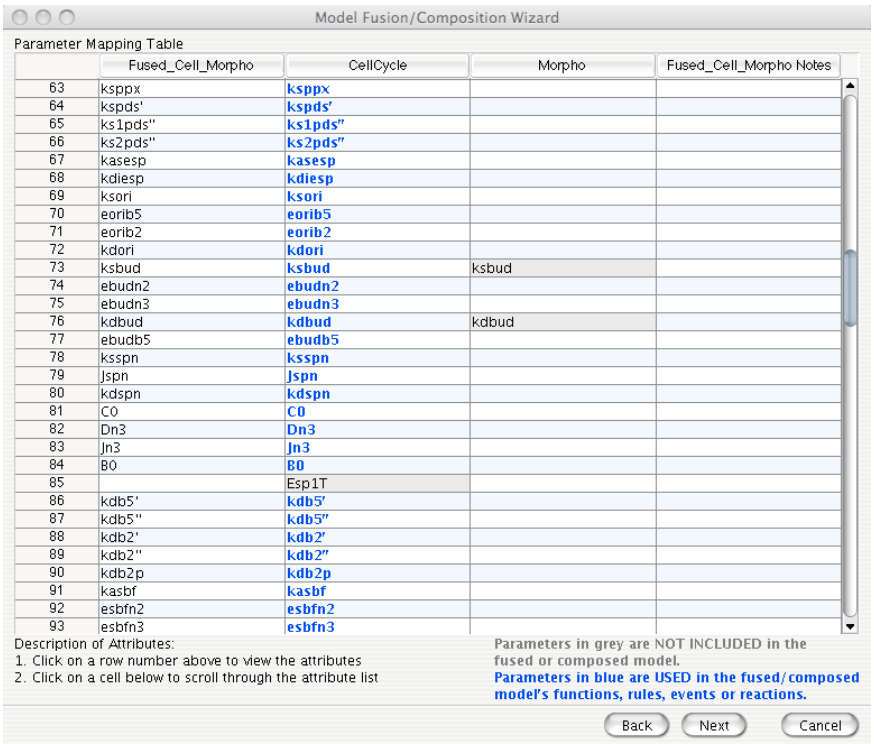


Figure A.8: 3^{rd} of 7 screenshots of the parameter mapping table of the combined model.

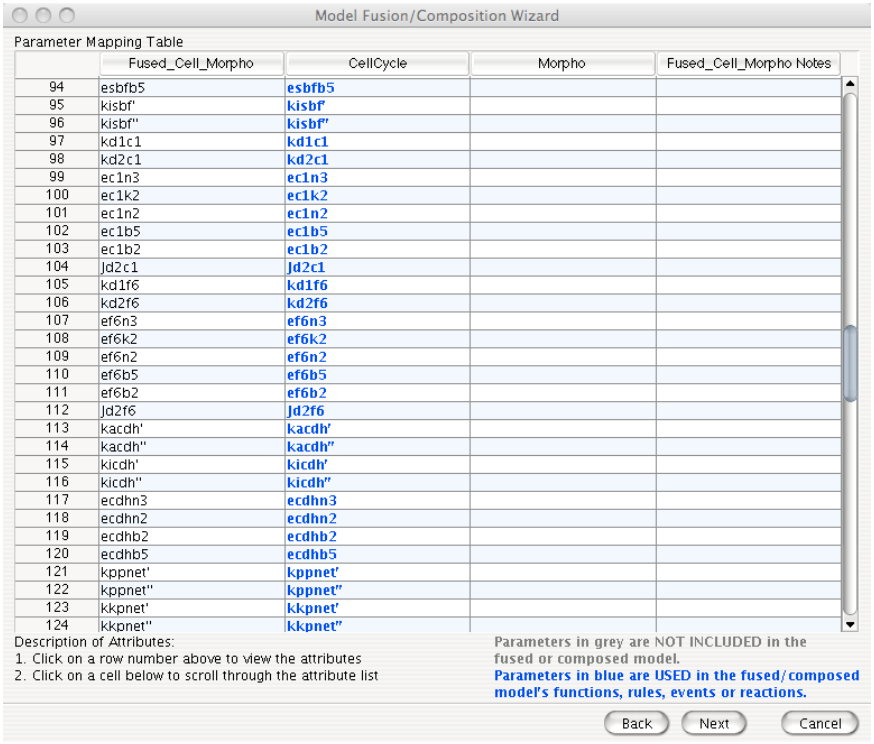


Figure A.9: 4^{th} of 7 screenshots of the parameter mapping table of the combined model.

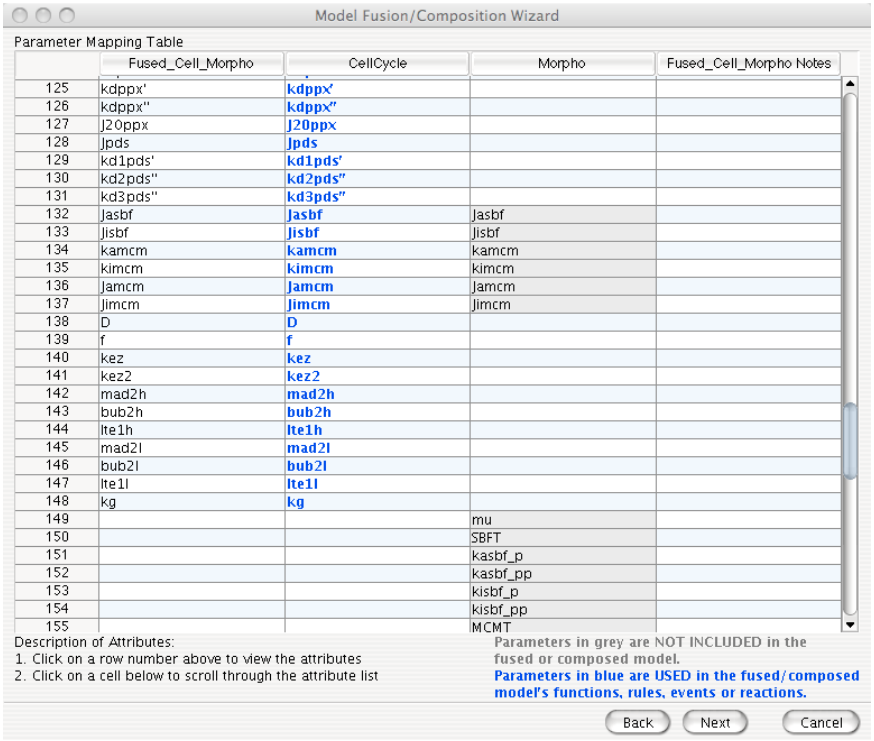


Figure A.10: 5^{th} of 7 screenshots of the parameter mapping table of the combined model.

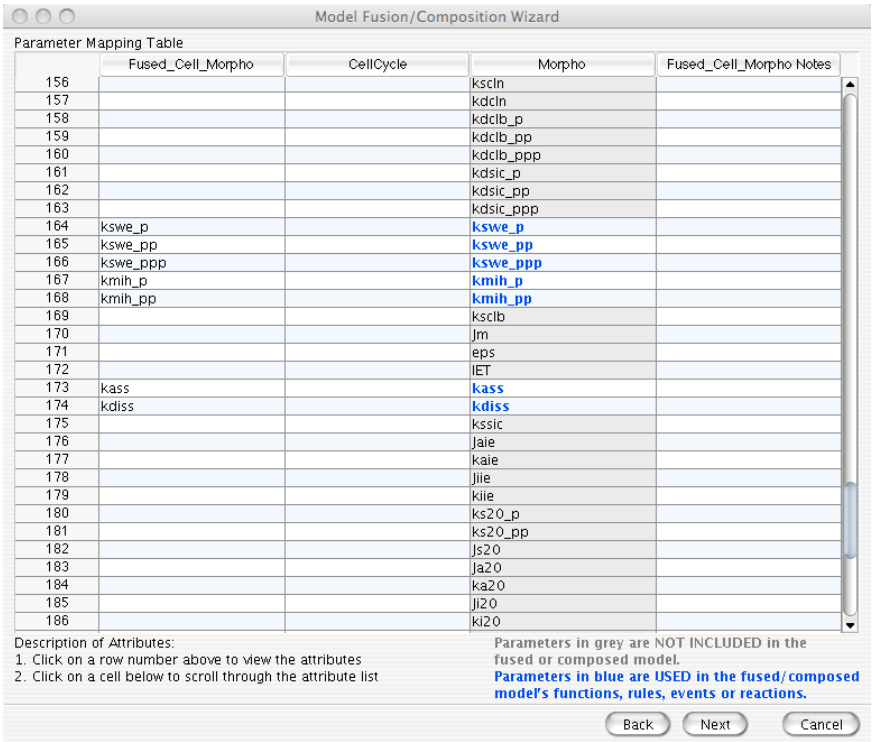


Figure A.11: 6^{th} of 7 screenshots of the parameter mapping table of the combined model.

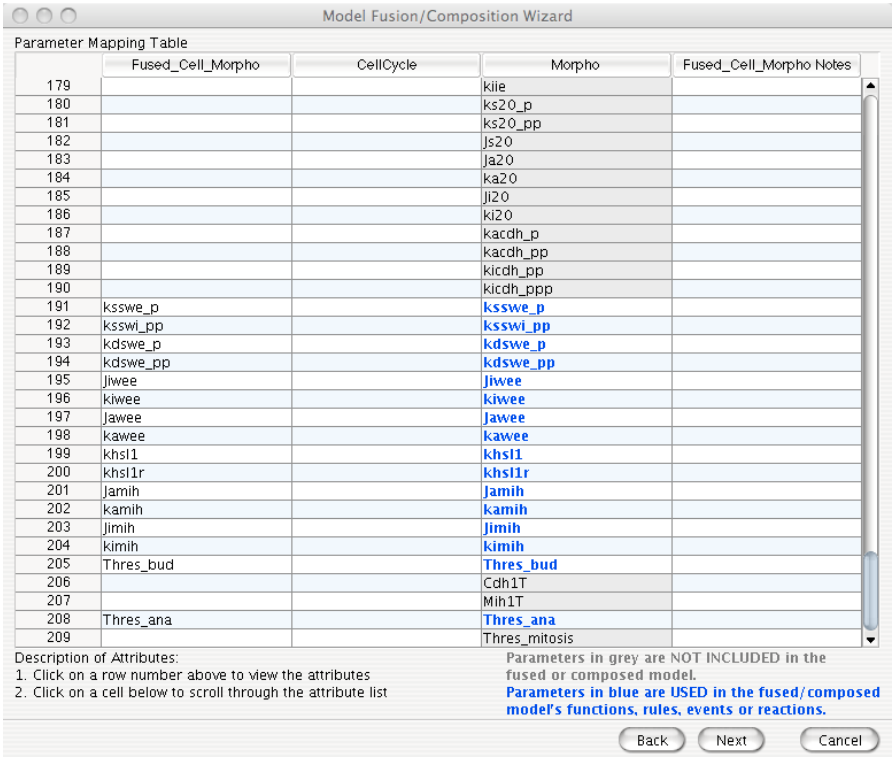


Figure A.12: 7^{th} of 7 screenshots of the parameter mapping table of the combined model.

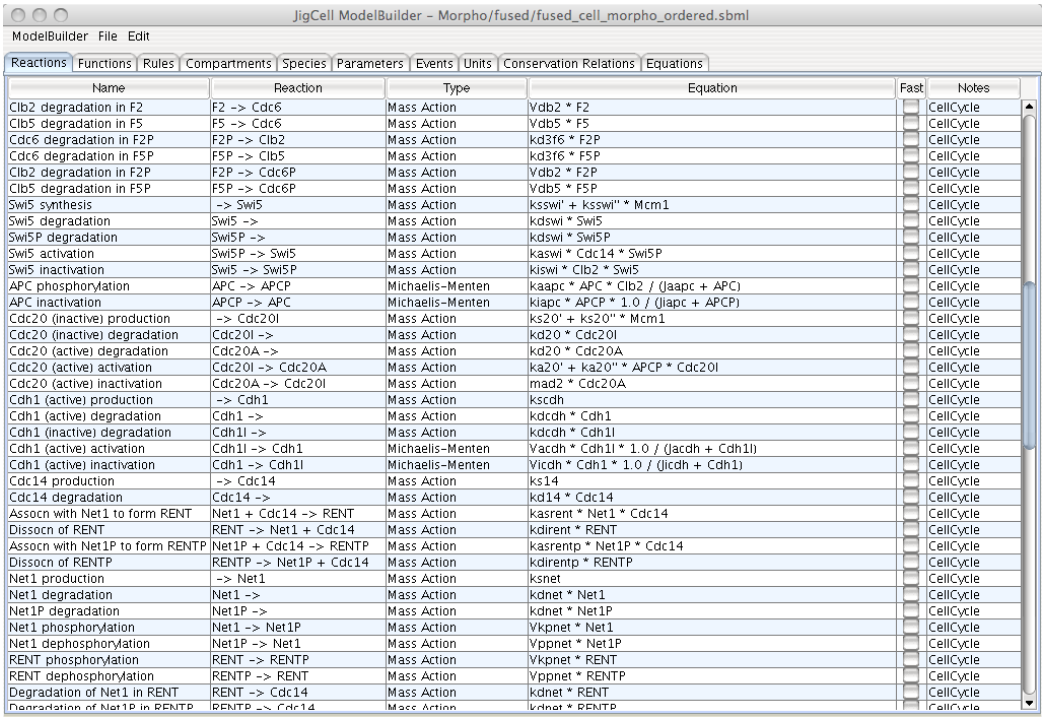
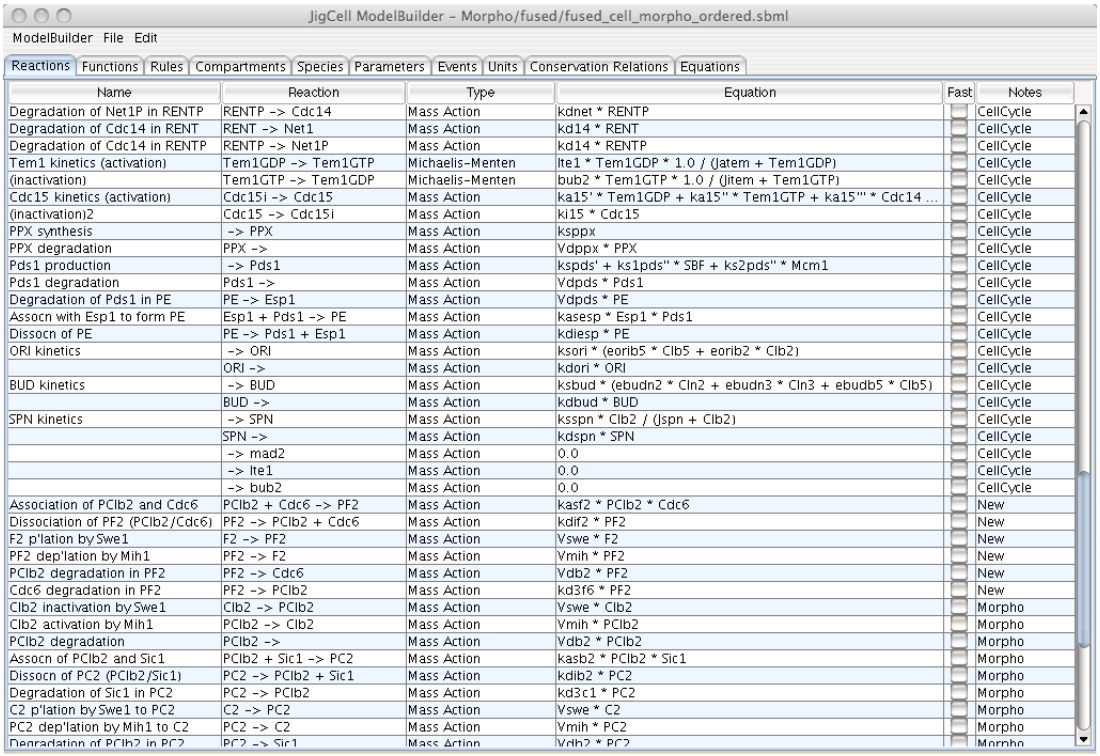


Figure A.13: 2^{nd} of 4 screenshots of the reactions spreadsheet of the $Fused\_Cell\_Morpho$ model in the JigCell ModelBuilder.

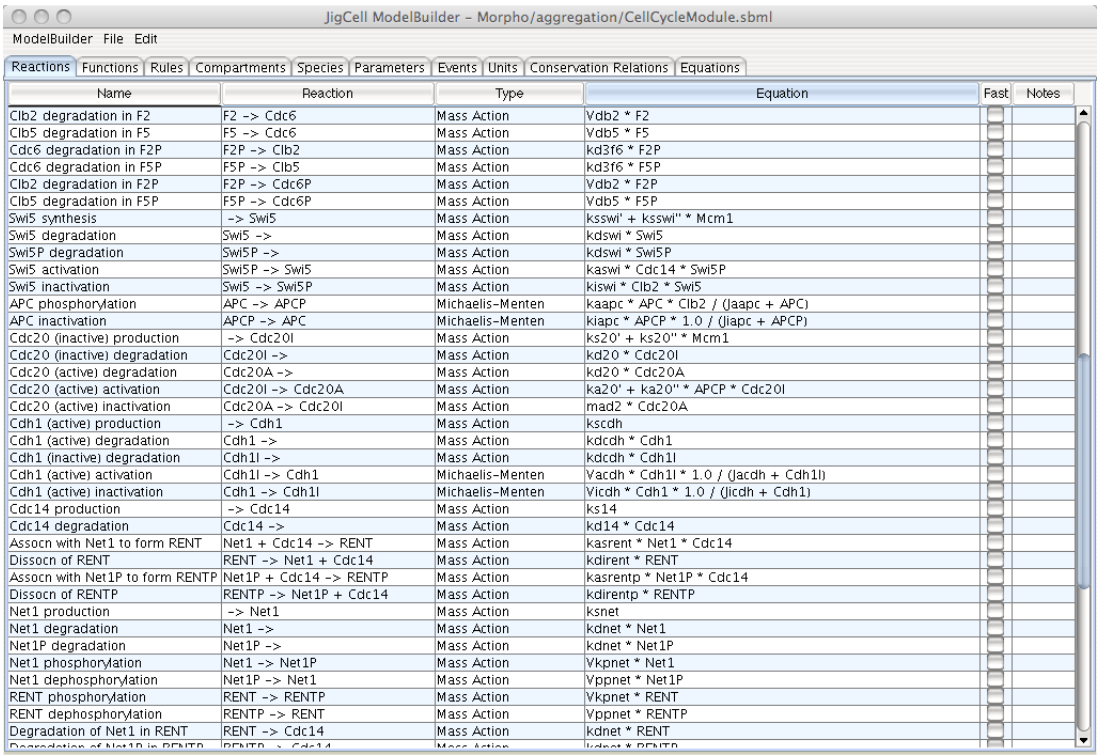


Name	Reaction	Type	Equation	Fast	Notes
Degradation of Net1P in RENTP	RENTP -> Cdc14	Mass Action	kdnet * RENTP		CellCycle
Degradation of Cdc14 in RENT	RENT -> Net1	Mass Action	kd14 * RENT		CellCycle
Degradation of Cdc14 in RENTP	RENTP -> Net1P	Mass Action	kd14 * RENTP		CellCycle
Tem1 kinetics (activation)	Tem1GDP -> Tem1GTP	Michaelis-Menten	kte1 * Tem1GDP * 1.0 / (Jatem + Tem1GDP)		CellCycle
(inactivation)	Tem1GTP -> Tem1GDP	Michaelis-Menten	kbub2 * Tem1GTP * 1.0 / (Jitem + Tem1GTP)		CellCycle
Cdc15 kinetics (activation)	Cdc15i -> Cdc15	Mass Action	ka15' * Tem1GDP + ka15'' * Tem1GTP + ka15''' * Cdc14 ...		CellCycle
(inactivation)2	Cdc15 -> Cdc15i	Mass Action	ki15 * Cdc15		CellCycle
PPX synthesis	-> PPX	Mass Action	ksppx		CellCycle
PPX degradation	PPX ->	Mass Action	Vdppx * PPX		CellCycle
Pds1 production	-> Pds1	Mass Action	kspds' + ks1pds'' * SBF + ks2pds'' * Mcm1		CellCycle
Pds1 degradation	Pds1 ->	Mass Action	Vdpds * Pds1		CellCycle
Degradation of Pds1 in PE	PE -> Esp1	Mass Action	Vdpds * PE		CellCycle
Assocn with Esp1 to form PE	Esp1 + Pds1 -> PE	Mass Action	kasesp * Esp1 * Pds1		CellCycle
Dissocn of PE	PE -> Pds1 + Esp1	Mass Action	kdiesp * PE		CellCycle
ORI kinetics	-> ORI	Mass Action	ksori * (eorib5 * Clb5 + eorib2 * Clb2)		CellCycle
	ORI ->	Mass Action	kdori * ORI		CellCycle
BUD kinetics	-> BUD	Mass Action	ksbud * (ebudn2 * Cln2 + ebudn3 * Cln3 + ebudb5 * Clb5)		CellCycle
	BUD ->	Mass Action	kdbud * BUD		CellCycle
SPN kinetics	-> SPN	Mass Action	kspn * Clb2 / (Jspn + Clb2)		CellCycle
	SPN ->	Mass Action	kdspn * SPN		CellCycle
	-> mad2	Mass Action	0.0		CellCycle
	-> Ite1	Mass Action	0.0		CellCycle
	-> bub2	Mass Action	0.0		CellCycle
Association of PClb2 and Cdc6	PClb2 + Cdc6 -> PF2	Mass Action	kasf2 * PClb2 * Cdc6		New
Dissociation of PF2 (PClb2/Cdc6)	PF2 -> PClb2 + Cdc6	Mass Action	kdif2 * PF2		New
F2 p'lation by Swe1	F2 -> PF2	Mass Action	Vswe * F2		New
PF2 dep'lation by Mih1	PF2 -> F2	Mass Action	Vmih * PF2		New
PClb2 degradation in PF2	PF2 -> Cdc6	Mass Action	Vdb2 * PF2		New
Cdc6 degradation in PF2	PF2 -> PClb2	Mass Action	kd3f6 * PF2		New
Clb2 inactivation by Swe1	Clb2 -> PClb2	Mass Action	Vswe * Clb2		Morpho
Clb2 activation by Mih1	PClb2 -> Clb2	Mass Action	Vmih * PClb2		Morpho
PClb2 degradation	PClb2 ->	Mass Action	Vdb2 * PClb2		Morpho
Assocn of PClb2 and Sic1	PClb2 + Sic1 -> PC2	Mass Action	kasb2 * PClb2 * Sic1		Morpho
Dissocn of PC2 (PClb2/Sic1)	PC2 -> PClb2 + Sic1	Mass Action	kdib2 * PC2		Morpho
Degradation of Sic1 in PC2	PC2 -> PClb2	Mass Action	kd3c1 * PC2		Morpho
C2 p'lation by Swe1 to PC2	C2 -> PC2	Mass Action	Vswe * C2		Morpho
PC2 dep'lation by Mih1 to C2	PC2 -> C2	Mass Action	Vmih * PC2		Morpho
Degradation of PClb2 in PC2	PC2 -> Sic1	Mass Action	Vdb2 * PC2		Morpho

Figure A.14: 3^{rd} of 4 screenshots of the reactions spreadsheet of the $Fused\_Cell\_Morpho$ model in the JigCell ModelBuilder.

Degradation of PClb2 in PC2	PC2 -> Sic1	Mass Action	Vdb2 * PC2		Morpho
Swe1 synthesis by SBF	-> Swe1	Mass Action	ksswe' + ksswe'' * SBF		Morpho
Swe1 degradation	Swe1 ->	Mass Action	kdswe' * Swe1		Morpho
PSwe1 degradation	PSwe1 ->	Mass Action	kdswe' * PSwe1		Morpho
Swe1M degradation	Swe1M ->	Mass Action	kdswe' * Swe1M		Morpho
PSwe1M degradation by SCF	PSwe1M ->	Mass Action	kdswe'' * PSwe1M		Morpho
Swe1 inactivation by Clb2 to fo...	Swe1 -> PSwe1	Michaelis-Menten	kiswe * Swe1 * Clb2 / (Jiswe + Swe1)		Morpho
PSwe1 activation to Swe1	PSwe1 -> Swe1	Michaelis-Menten	kaswe * PSwe1 * 1.0 / (Jaswe + PSwe1)		Morpho
Swe1M inactivation by Clb to for...	Swe1M -> PSwe1M	Michaelis-Menten	kiswe * Swe1M * Clb2 / (Jiswe + Swe1M)		Morpho
PSwe1M activation to Swe1M	PSwe1M -> Swe1M	Michaelis-Menten	kaswe * PSwe1M * 1.0 / (Jaswe + PSwe1M)		Morpho
Swe1 inactivation by BUD to for...	Swe1 -> Swe1M	Mass Action	khsl1 * BUD2 * Swe1		Morpho
Swe1M activation to Swe1	Swe1M -> Swe1	Mass Action	khsl1r * Swe1M		Morpho
PSwe1 inactivation by Bud to for...	PSwe1 -> PSwe1M	Mass Action	khsl1 * BUD2 * PSwe1		Morpho
PSwe1M activation to PSwe1	PSwe1M -> PSwe1	Mass Action	khsl1r * PSwe1M		Morpho
Mih1i activation by Clb	Mih1i -> Mih1a	Michaelis-Menten	kamih * Mih1i * Clb2 / (Jamih + Mih1i)		Morpho
Mih1a inactivation	Mih1a -> Mih1i	Michaelis-Menten	kimih * Mih1a * 1.0 / (Jimih + Mih1a)		Morpho
Anaphase flag	-> ANA	Mass Action	k1		Morpho
Budding flag	-> BUD2	Mass Action	0.0		Morpho

Figure A.15: 4^{th} of 4 screenshots of the reactions spreadsheet of the $Fused\_Cell\_Morpho$ model in the JigCell ModelBuilder.



Name	Reaction	Type	Equation	Fast	Notes
Clb2 degradation in F2	F2 -> Cdc6	Mass Action	Vdb2 * F2		
Clb5 degradation in F5	F5 -> Cdc6	Mass Action	Vdb5 * F5		
Cdc6 degradation in F2P	F2P -> Clb2	Mass Action	kd3f6 * F2P		
Cdc6 degradation in F5P	F5P -> Clb5	Mass Action	kd3f6 * F5P		
Clb2 degradation in F2P	F2P -> Cdc6P	Mass Action	Vdb2 * F2P		
Clb5 degradation in F5P	F5P -> Cdc6P	Mass Action	Vdb5 * F5P		
Swi5 synthesis	-> Swi5	Mass Action	kswi' + kswi'' * Mcm1		
Swi5 degradation	Swi5 ->	Mass Action	kdswi * Swi5		
Swi5P degradation	Swi5P ->	Mass Action	kdswi * Swi5P		
Swi5 activation	Swi5P -> Swi5	Mass Action	kaswi * Cdc14 * Swi5P		
Swi5 inactivation	Swi5 -> Swi5P	Mass Action	kiswi * Clb2 * Swi5		
APC phosphorylation	APC -> APCP	Michaelis-Menten	kaapc * APC * Clb2 / (Jaapc + APC)		
APC inactivation	APCP -> APC	Michaelis-Menten	kiapc * APCP * 1.0 / (Jiapc + APCP)		
Cdc20 (inactive) production	-> Cdc20I	Mass Action	ks20' + ks20'' * Mcm1		
Cdc20 (inactive) degradation	Cdc20I ->	Mass Action	kd20 * Cdc20I		
Cdc20 (active) degradation	Cdc20A ->	Mass Action	kd20 * Cdc20A		
Cdc20 (active) activation	Cdc20I -> Cdc20A	Mass Action	ka20' + ka20'' * APCP * Cdc20I		
Cdc20 (active) inactivation	Cdc20A -> Cdc20I	Mass Action	mad2 * Cdc20A		
Cdh1 (active) production	-> Cdh1	Mass Action	kscdh		
Cdh1 (active) degradation	Cdh1 ->	Mass Action	kdcdh * Cdh1		
Cdh1 (inactive) degradation	Cdh1I ->	Mass Action	kdcdh * Cdh1I		
Cdh1 (active) activation	Cdh1I -> Cdh1	Michaelis-Menten	Vacdh * Cdh1I * 1.0 / (Jacdh + Cdh1I)		
Cdh1 (active) inactivation	Cdh1 -> Cdh1I	Michaelis-Menten	Vicdh * Cdh1 * 1.0 / (Jicdh + Cdh1)		
Cdc14 production	-> Cdc14	Mass Action	ks14		
Cdc14 degradation	Cdc14 ->	Mass Action	kd14 * Cdc14		
Assocn with Net1 to form RENT	Net1 + Cdc14 -> RENT	Mass Action	kasrent * Net1 * Cdc14		
Dissocn of RENT	RENT -> Net1 + Cdc14	Mass Action	kdirent * RENT		
Assocn with Net1P to form RENTP	Net1P + Cdc14 -> RENTP	Mass Action	kasrentp * Net1P * Cdc14		
Dissocn of RENTP	RENTP -> Net1P + Cdc14	Mass Action	kdirentp * RENTP		
Net1 production	-> Net1	Mass Action	ksnet		
Net1 degradation	Net1 ->	Mass Action	kdnet * Net1		
Net1P degradation	Net1P ->	Mass Action	kdnet * Net1P		
Net1 phosphorylation	Net1 -> Net1P	Mass Action	Vkpnet * Net1		
Net1 dephosphorylation	Net1P -> Net1	Mass Action	Vppnet * Net1P		
RENT phosphorylation	RENT -> RENTP	Mass Action	Vkpnet * RENT		
RENT dephosphorylation	RENTP -> RENT	Mass Action	Vppnet * RENTP		
Degradation of Net1 in RENT	RENT -> Cdc14	Mass Action	kdnet * RENT		

Figure A.16: 2^{nd} of 3 screenshots of the reactions spreadsheet of the cell cycle module in the JigCell ModelBuilder.

Degradation of Net1P in RENTP	RENTP -> Cdc14	Mass Action	kdnet * RENTP		
Degradation of Cdc14 in RENT	RENT -> Net1	Mass Action	kd14 * RENT		
Degradation of Cdc14 in RENTP	RENTP -> Net1P	Mass Action	kd14 * RENTP		
Tem1 kinetics (activation)	Tem1GDP -> Tem1GTP	Michaelis-Menten	kte1 * Tem1GDP * 1.0 / (Jatem + Tem1GDP)		
(inactivation)	Tem1GTP -> Tem1GDP	Michaelis-Menten	kbub2 * Tem1GTP * 1.0 / (Jitem + Tem1GTP)		
Cdc15 kinetics (activation)	Cdc15i -> Cdc15	Mass Action	ka15' * Tem1GDP + ka15'' * Tem1GTP + ka15''' * Cdc14 * ...		
(inactivation)2	Cdc15 -> Cdc15i	Mass Action	ki15 * Cdc15		
PPX synthesis	-> PPX	Mass Action	ksppx		
PPX degradation	PPX ->	Mass Action	Vdppx * PPX		
Pds1 production	-> Pds1	Mass Action	kspds' + ks1pds'' * SBF + ks2pds'' * Mcm1		
Pds1 degradation	Pds1 ->	Mass Action	Vdpds * Pds1		
Degradation of Pds1 in PE	PE -> Esp1	Mass Action	Vdpds * PE		
Assocn with Esp1 to form PE	Esp1 + Pds1 -> PE	Mass Action	kasesp * Esp1 * Pds1		
Dissocn of PE	PE -> Pds1 + Esp1	Mass Action	kdiesp * PE		
ORI kinetics	-> ORI	Mass Action	ksori * (eorib5 * Clb5 + eorib2 * Clb2)		
	ORI ->	Mass Action	kdori * ORI		
BUD kinetics	-> BUD	Mass Action	ksbud * (ebudn2 * Cln2 + ebudn3 * Cln3 + ebudb5 * Clb5)		
	BUD ->	Mass Action	kdbud * BUD		
SPN kinetics	-> SPN	Mass Action	kspn * Clb2 / (Jspn + Clb2)		
	SPN ->	Mass Action	kdspn * SPN		
	-> mad2	Mass Action	0.0		
	-> kte1	Mass Action	0.0		
	-> bub2	Mass Action	0.0		

Figure A.17: 3^{rd} of 3 screenshots of the reactions spreadsheet of the cell cycle module in the JigCell ModelBuilder.

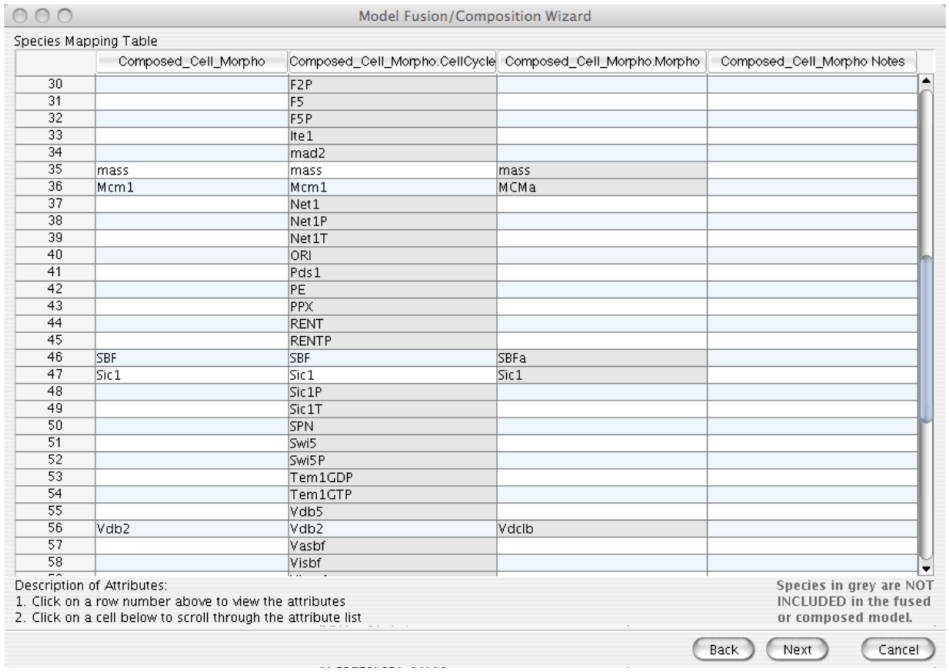


Figure A.18: 2^{nd} of 3 screenshots of the initial species composition mapping table of the morphogenesis and cell cycle combined model.

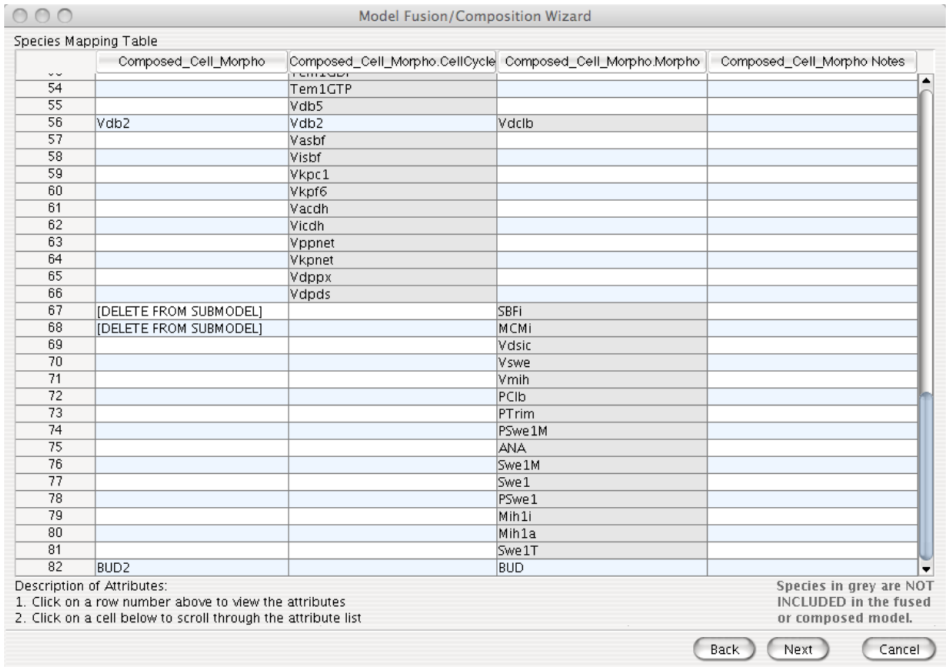


Figure A.19: 3^{rd} of 3 screenshots of the initial species composition mapping table of the morphogenesis and cell cycle combined model.

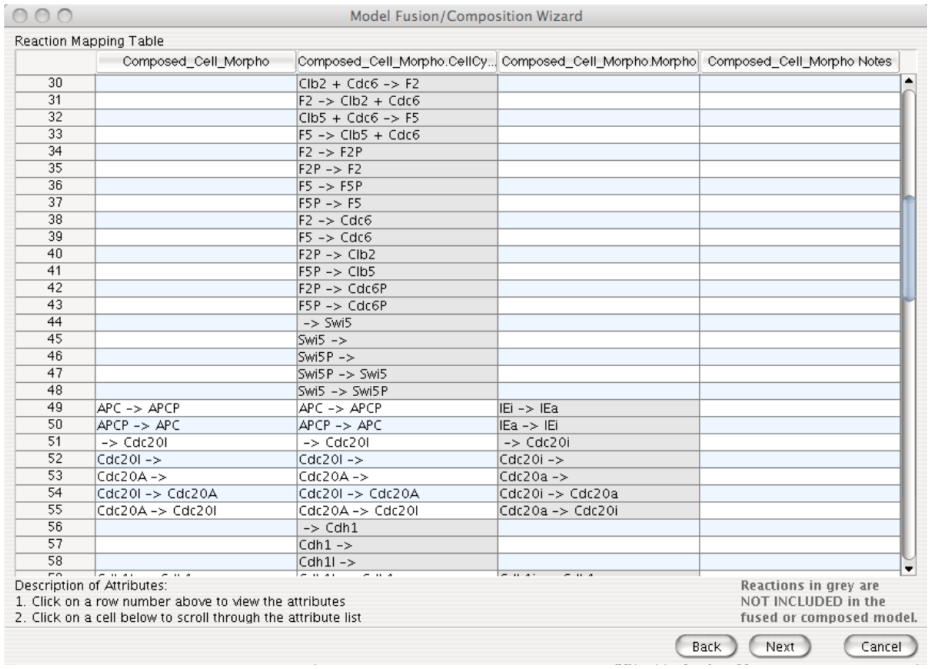


Figure A.20: 2^{nd} of 5 screenshots of the initial reaction composition mapping table of the morphogenesis and cell cycle combined model.

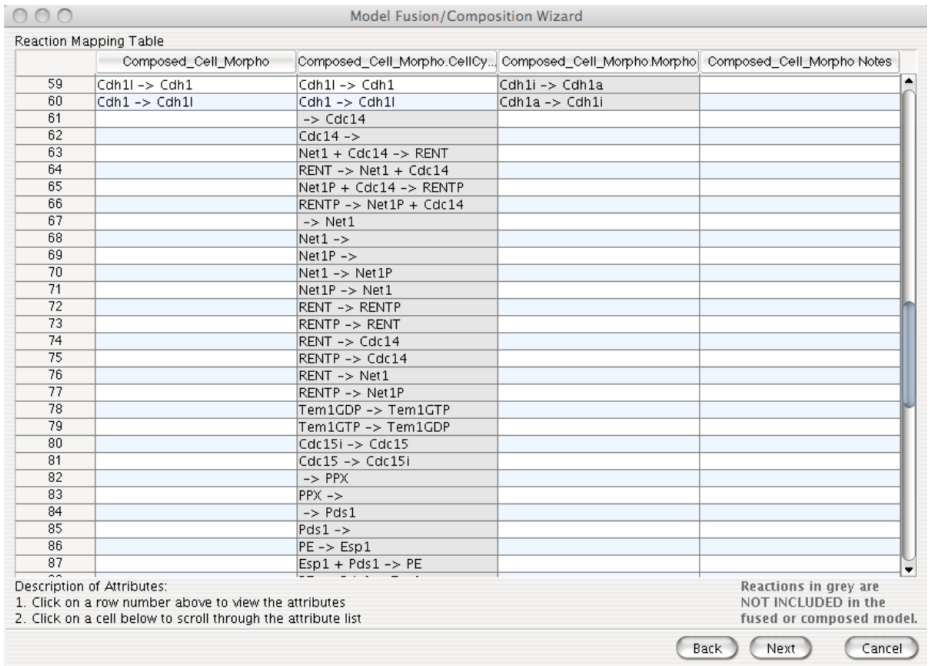


Figure A.21: 3^{rd} of 5 screenshots of the initial reaction composition mapping table of the morphogenesis and cell cycle combined model.

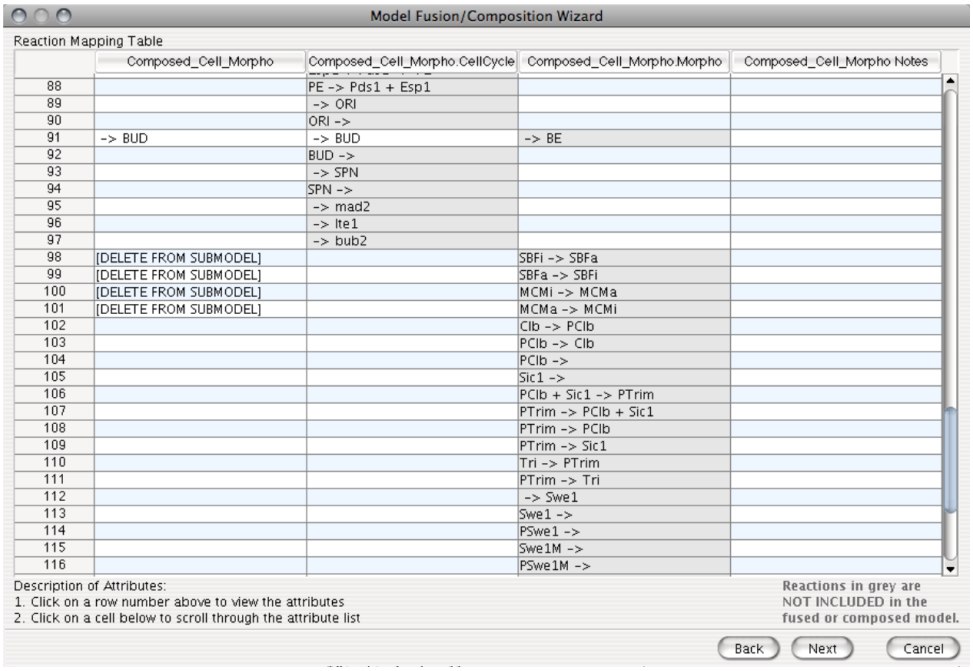


Figure A.22: 4^{th} of 5 screenshots of the initial reaction composition mapping table of the morphogenesis and cell cycle combined model.

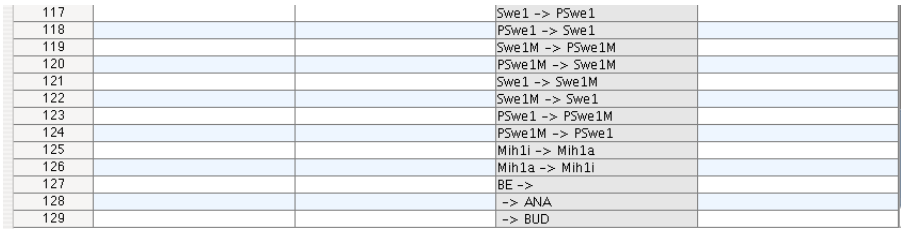


Figure A.23: 5^{th} of 5 screenshots of the initial reaction composition mapping table of the morphogenesis and cell cycle combined model.

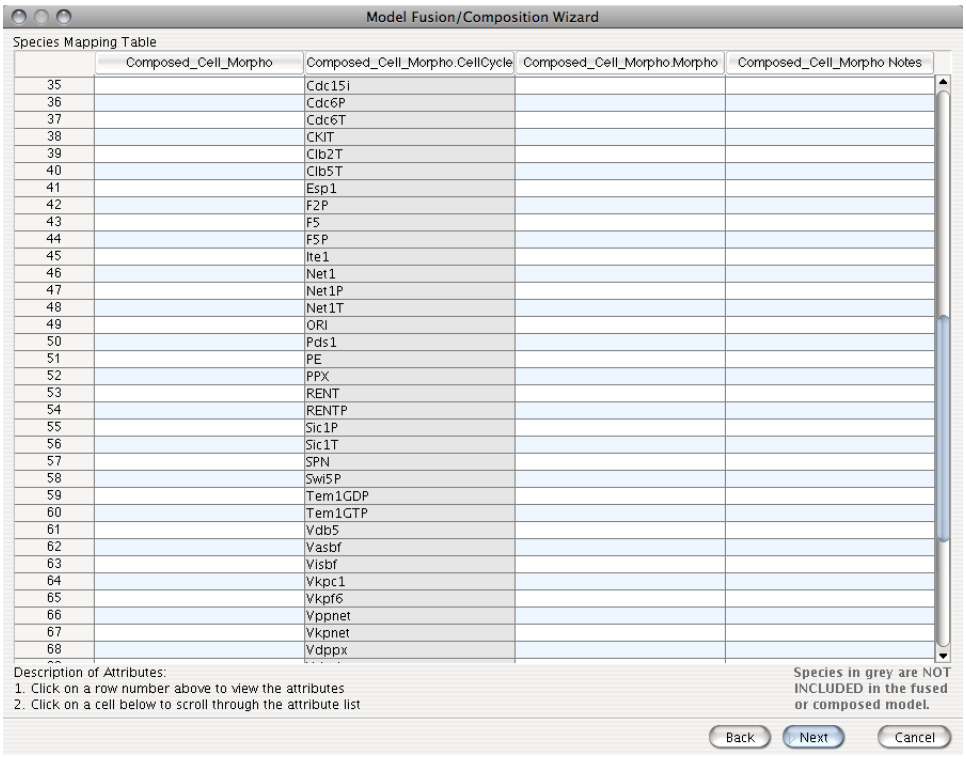


Figure A.24: 2^{nd} of 3 screenshots of the final species composition mapping table of the morphogenesis and cell cycle combined model.

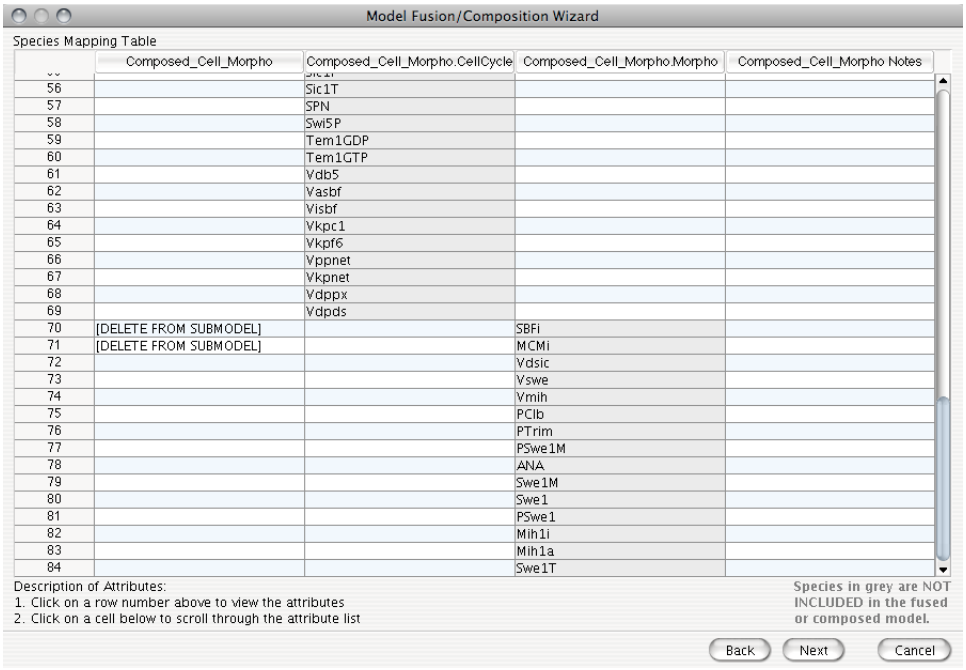


Figure A.25: 3^{rd} of 3 screenshots of the final species composition mapping table of the morphogenesis and cell cycle combined model.

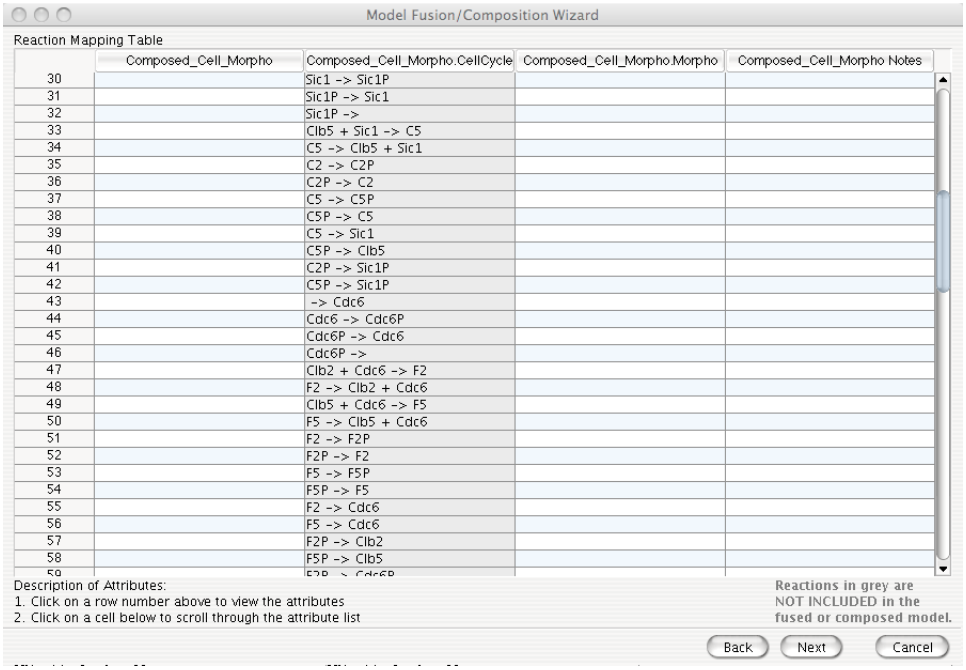


Figure A.26: 2^{nd} of 5 screenshots of the final reaction composition mapping table of the morphogenesis and cell cycle combined model.

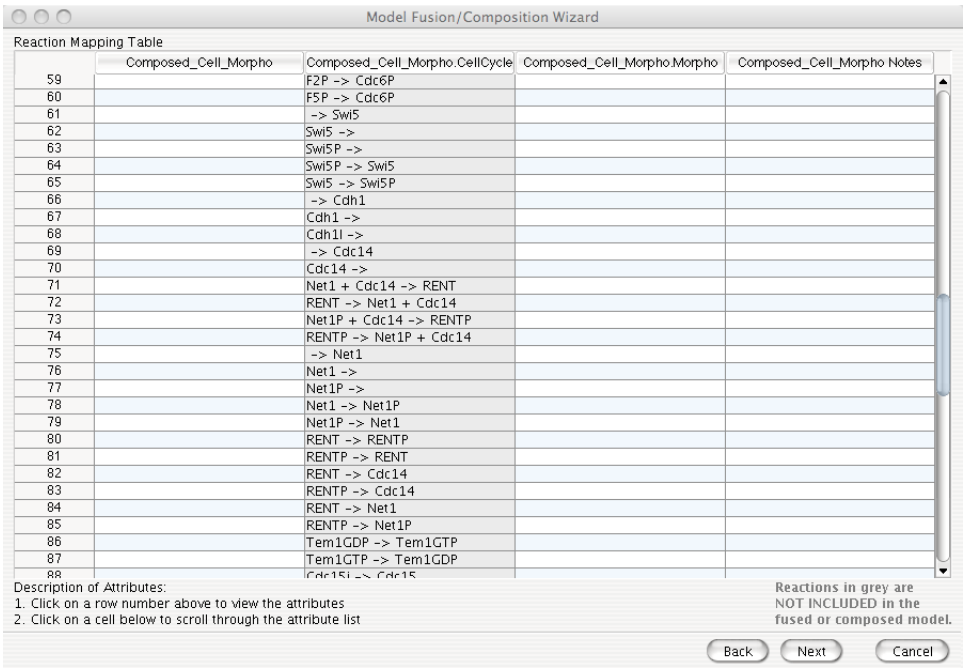


Figure A.27: 3^{rd} of 5 screenshots of the final reaction composition mapping table of the morphogenesis and cell cycle combined model.

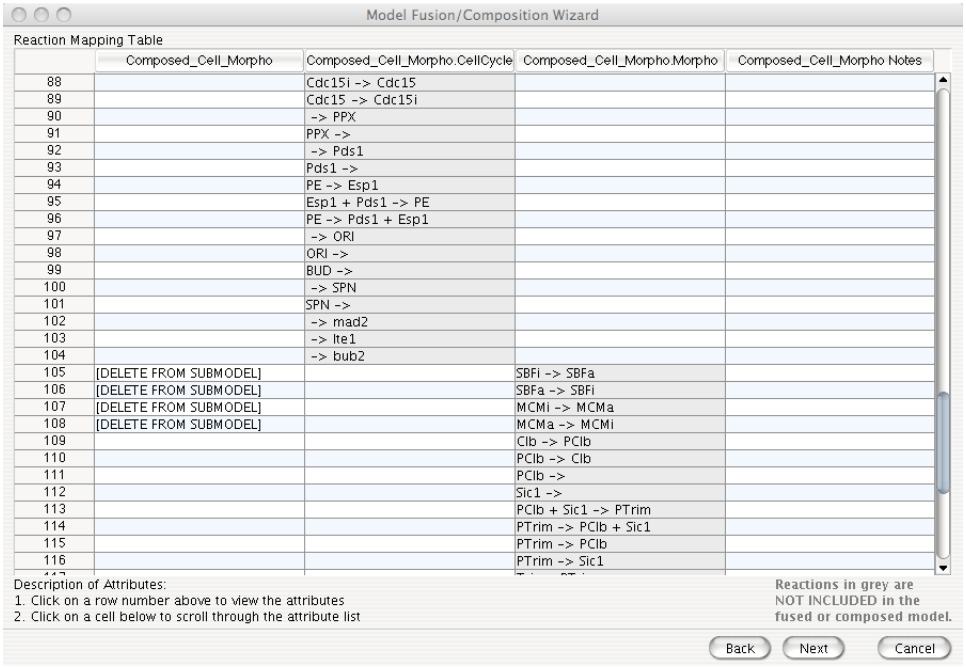


Figure A.28: 4^{th} of 5 screenshots of the final reaction composition mapping table of the morphogenesis and cell cycle combined model.

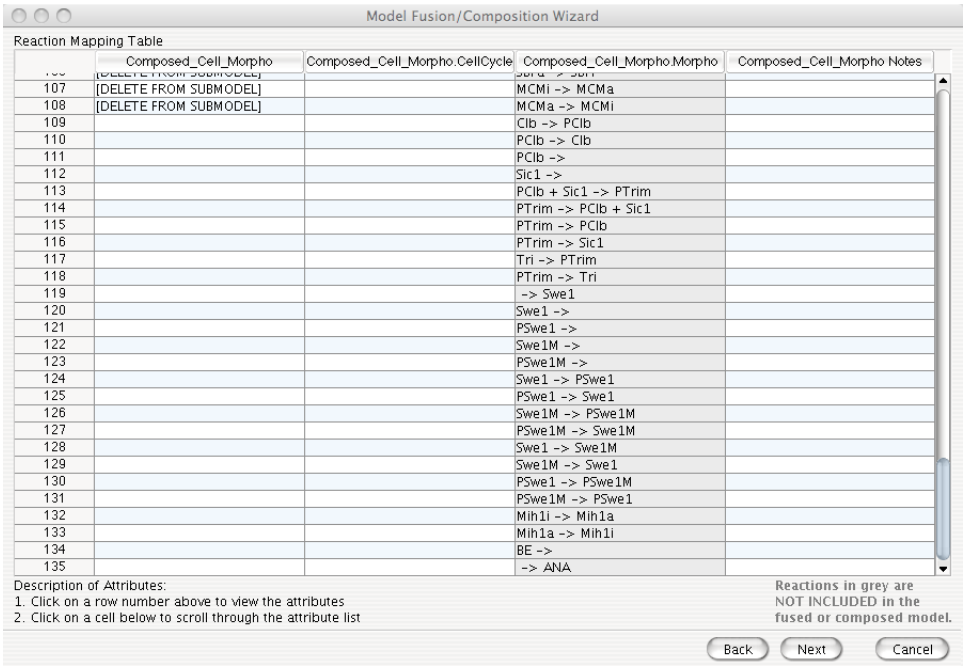


Figure A.29: 5^{th} of 5 screenshots of the final reaction composition mapping table of the morphogenesis and cell cycle combined model.

Appendix B

Hierarchical Modeling Proposal

The following is a proposal for the Hierarchical Modeling extension for SBML Level3. It is written as an outline of the sections we envision being important to put into the document and was prepared in its current form by Stefan Hoops.

B.1 Goals and Scope

The original objectives for the proposed SBML Composition extension were the following:

1. Facilitate the reuse of existing models (as components)
2. Manage complexity when building large models.

As various different approaches were analysed in considerable amount of detail during the September 2007 workshop on model composition, the goals, scope, and design of the extension evolved considerably:

1. There is a fundamental need for a simple algorithm for defining, referring to, and including, separate model components - e.g. chunks of SBML documents (or fragments thereof). This should be a "core" enhancement to SBML in Level 3. A separate proposal for such an enhancement (based on xlink) was drafted and is described here.
2. To define a model that can be arbitrarily complex, but purely hierarchical (when referring to its parts), is both required and sufficient for achieving all of the goals of the proposal. To reflect this, the proposed L3 extension has been renamed the Hierarchical Modeling extension. The syntactical elements needed are surprisingly few - as long as there is an unequivocal mechanism for separating"sub-model" namespaces and replacing ("overloading") model elements.
3. The proposed solution is simple, but has a number of quite powerful additional features (that were somewhat unforeseen):
   (a) The ability to create related collections of model variants - e.g. storing "multiple runs" of a model (with different kinetic parameters or initial conditions) - in a single SBML document

(b) The ability of almost complete backwards compatibility if libSBML will be enhanced to support this extension - e.g. Level2 software tools that use libSBML for document handling may be able to read in any such model as a “flattened” Level2 version3 model

(c) The ability to create “smart” visualization tools/algorithms - e.g. “black-box” display of components, selective zoom/editing of components, etc.

(d) The facilitation of tool interoperability with CellML language-encoded models and components

B.2 Syntax

B.2.1 HierarchicalModel Type

The HierachicalModel (Figure B.1) is the type of the <model> element, which is the top element of an SBML file containing a model which is composed from multiple submodels. The HierarchicalModel is derived from the Model type of the SBML core. The attributes of the <model> stay unchanged however the sequence of child elements is expanded. The added child elements are <listOfSubmodels>, <listOfReplacements>, and <listOfPorts>.

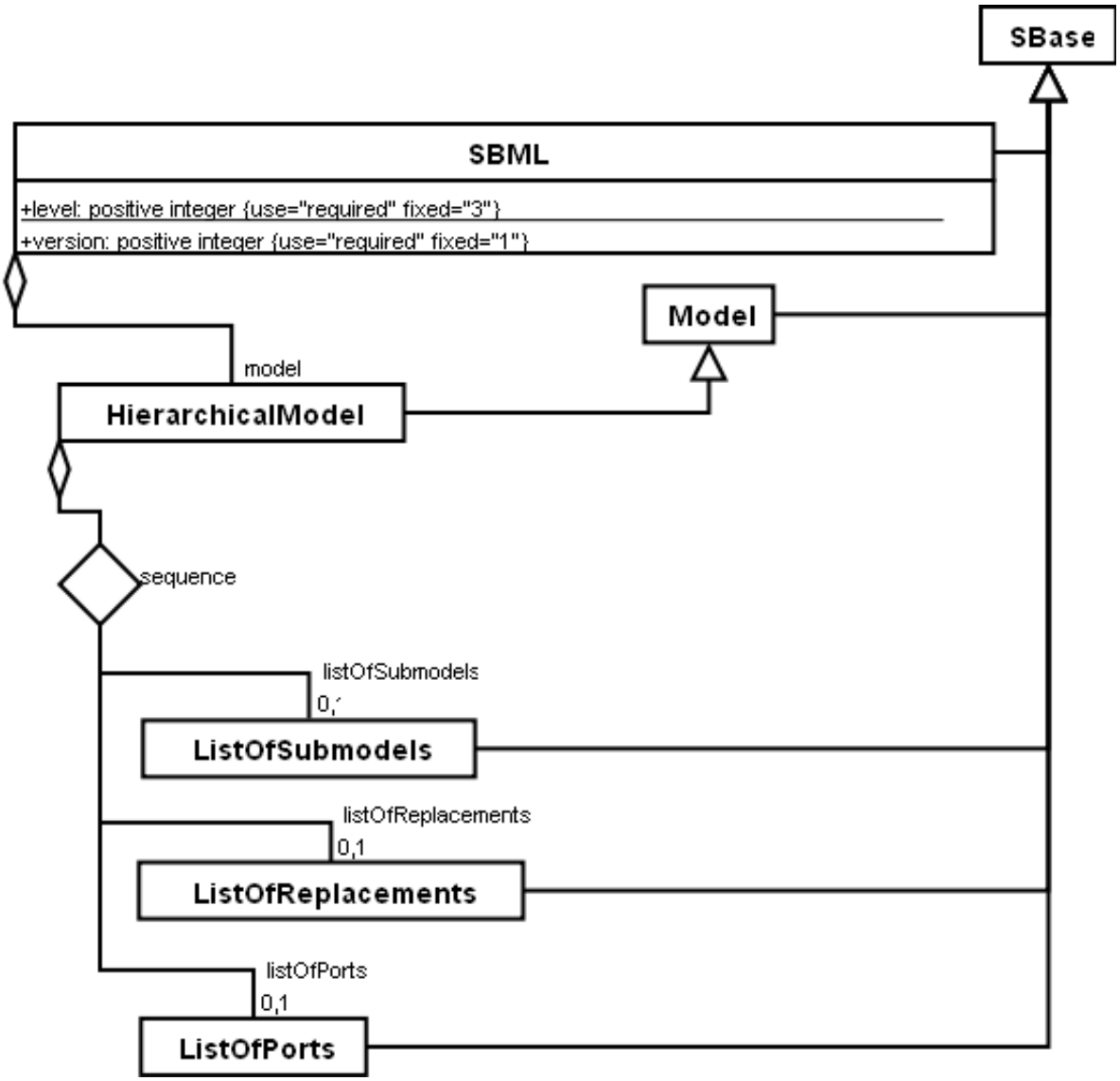


Figure B.1: Proposed HierarchicalModel Type for SBML Level3.

B.2.2 ListOfSubmodels Type

The ListOfSubmodels (Figure B.2) is the type of the <listOfSubmodels> element. This element contains the a list of submodels . The <model> sources which are instantiated may be defined by a model element, or locally and remotely (in another XML documnet) through the SBMLInclude type. The two alternate attribute lists of the $source$ element present restrictions of the SBMLInclude type (derived from $xi : include$) applicable for the local or remote model definitions. The scope of the SBML ID of the hierarchical model does exclude the child of the <submodel> element. This means that an instantiated model has its own namespace. It is possible to instantiate several copies of a submodel from the same source.

The resulting section in an SBML document would look like:

```
<listOfSubModels>
  <submodel id="Submodel_1">
    <substanceConversionFactor
      id="SCF_1"
      name="Substant Conversion Factor 1"
      value="1000"
      units="submodelSubstsanceUnitPerParentModelSubstanceUnit"
      constant="true" />
    <volumeConversionFactor
      id="VCF_1"
      name="Volume Conversion Factor 1"
      value="1000"
      units="submodelVolumeUnitPerParentModelVolumeUnit"
      constant="true" />
    <areaConversionFactor
      id="ACF_1"
      name="Area Conversion Factor 1"
      value="100"
      units="submodelAreaUnitPerParentModelAreaUnit"
      constant="true" />
    <lengthConversionFactor
      id="LCF_1"
      name="Length Conversion Factor 1"
      value="10"
      units="submodelLengthUnitPerParentModelLengthUnit"
      constant="true" />
    <timeConversionFactor
      id="TCF_1"
      name="Time Conversion Factor 1"
      value="60"
      units="submodelTimeUnitPerParentModelTimeUnit"
      constant="true" />
    <model id = "HierarichicalModel_1">
      ...
    </model>
  </submodel>
    ...
  <submodel id="Submodel_X">
    <model xref="..." xpointer="..." />
  </submodel >
</listOfSubmodels>
```

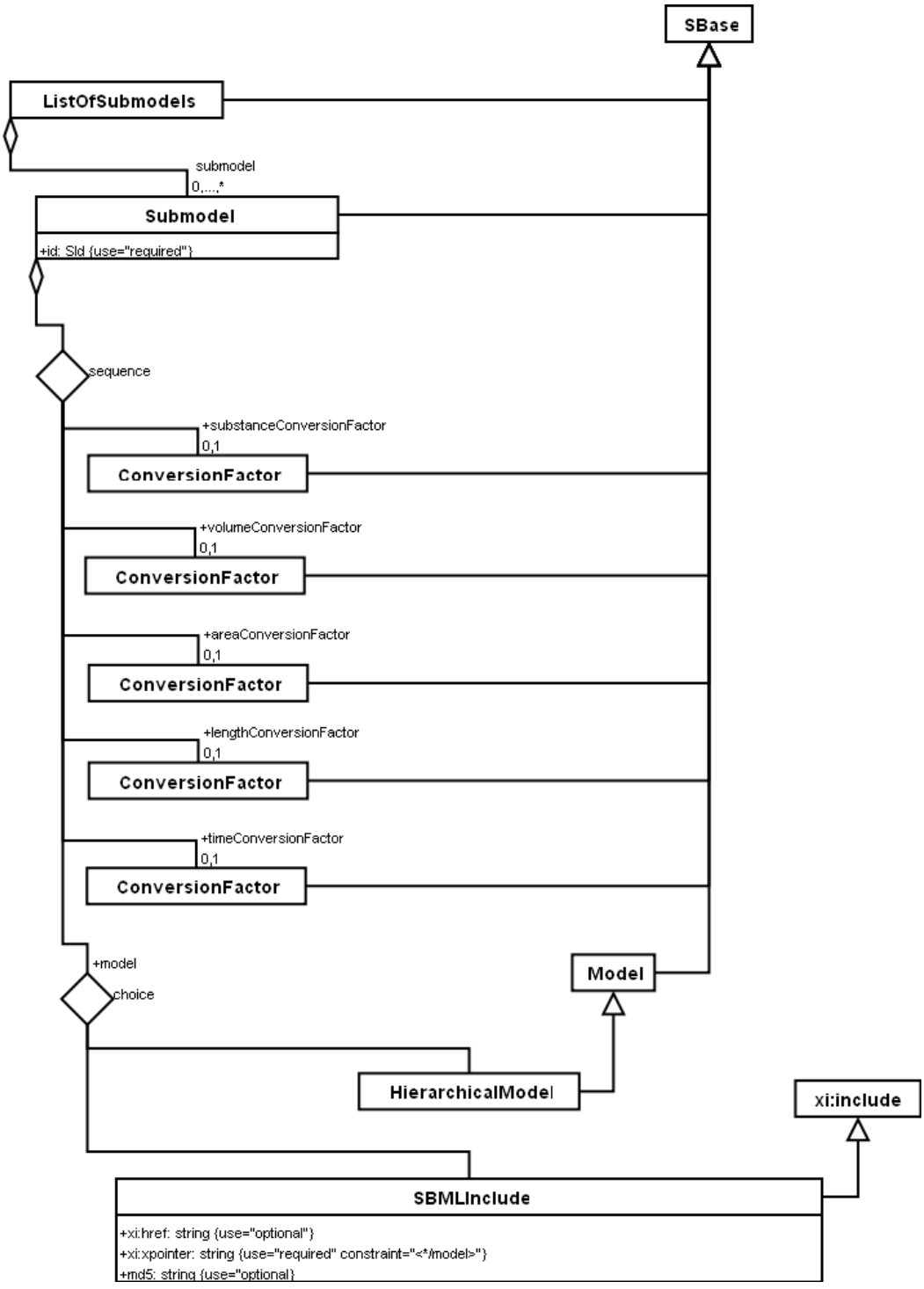


Figure B.2: Proposed ListOfSubmodels Type for SBML Level3.

B.2.3 ListOfReplacements Type

See Figure B.3.

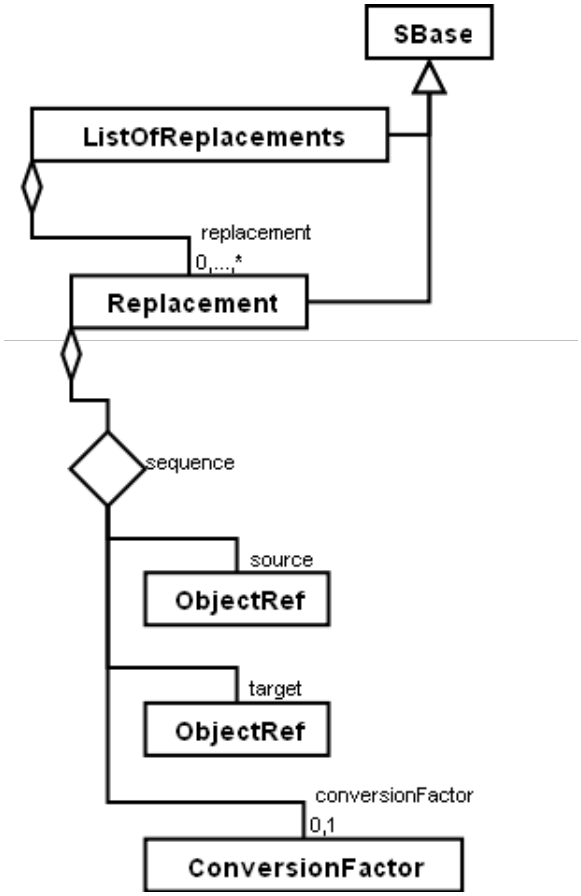


Figure B.3: Proposed ListOfReplacements Type for SBML Level3.

B.2.4 ListOfPorts Type

See Figure B.4.

B.2.5 ConversionFactor Type

The ConversionFactor type (Figure B.5) is used to specify the conversions required between the parent and sub models as well as the conversion required for replacements. The type is derived from the Parameter type of the SBML core with the restriction that the $constant$ attribute is fixed to the value $true$.

The conversion is such that following two MathML fragments are equivalent in value and in units.

```
<ci> target </ci>
```

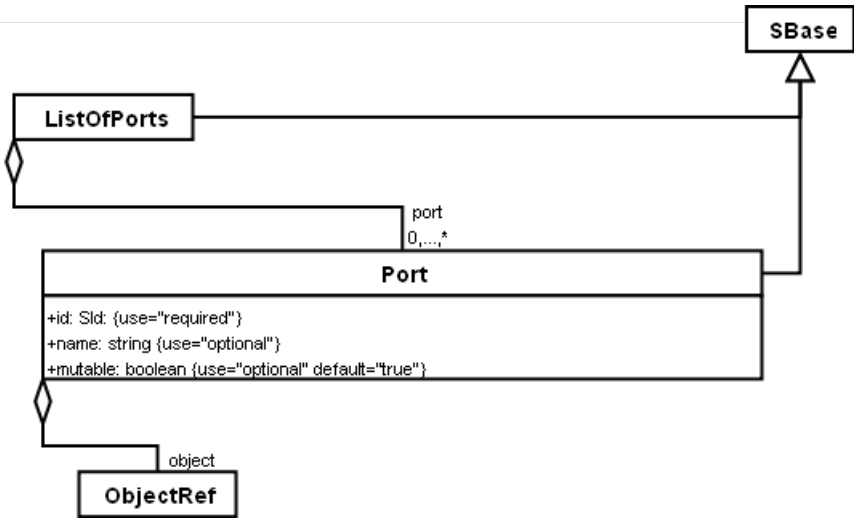


Figure B.4: Proposed ListOfPorts Type for SBML Level3.

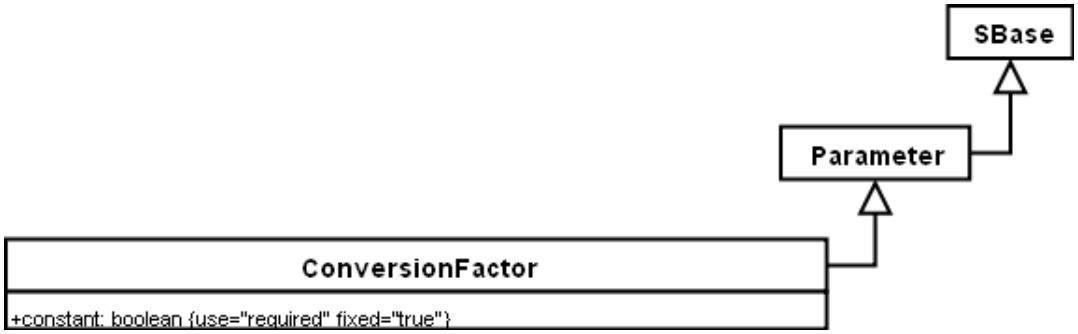


Figure B.5: Proposed ConversionFactor Type for SBML Level3.

```
<apply>
  <times/>
  <ci> conversionFactor </ci>
  <ci> source </ci>
</apply>
```

This means that when we flatten a hierarchical model to a SBML core model every occurrence of the first fragment will be replaced by the second. For this mechanism to work properly in the units space it is required that proper units for the conversion factors must be defined as part of the parent model.

B.2.6 ObjectReference Type

See Figure B.6.

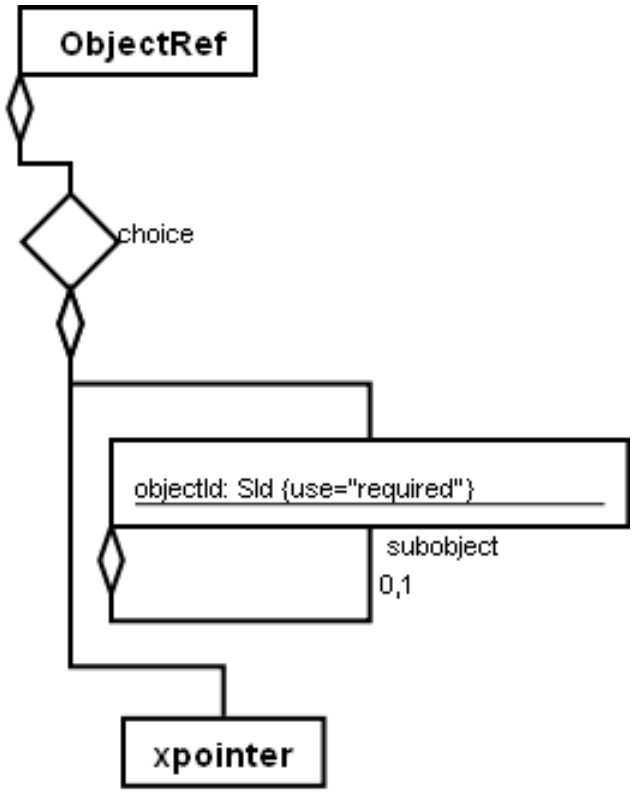


Figure B.6: Proposed ObjectReference Type for SBML Level3.

B.3 Expected impact on other extensions

Some possible other L3 extensions that might be impacted by this one are: modularity/inclusion, rule-based models, arrays, spatial geometry.