Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

THE ROLE OF COMPOSITION AND AGGREGATION IN MODELING MACROMOLECULAR REGULATORY NETWORKS

Clifford A. Shaffer*, Ranjit Randhawa*, and John J. Tyson**

Departments of Computer Science* and Biological Sciences**
Virginia Tech
Blacksburg, VA 24061
shaffer|rrandhawa|tyson@vt.edu

ABSTRACT

Today's macromolecular regulatory network models are small compared to the amount of information known about a particular cellular pathway, in part because current modeling languages and tools are unable to handle significantly larger models. Thus, most pathway modeling work today focuses on building small models of individual pathways since they are easy to construct and manage. The hope is someday to put these pieces together to create a more complete picture of the underlying molecular machinery. While efforts to make large models benefit from reusing existing components, unfortunately, there currently exists little tool or representational support for combining or composing models. We have identified four distinct modeling processes related to model composition: fusion, composition, aggregation, and flattening. We present concrete proposals for implementing all four processes in the context of the Systems Biology Markup Language (SBML).

1 REGULATORY NETWORK MODELING

Macromolecular regulatory network models attempt to deduce the physiological properties of a cell from wiring diagrams of its control systems. These networks of interacting proteins are intrinsically dynamic. They describe the molecular mechanisms by which a cell changes in space and time to respond to stimuli, grow and reproduce, differentiate, and do all the other remarkable tricks that are necessary to stay alive and propagate the species.

A simple example of a regulatory network is the set of reactions controlling the activity of MPF (mitosis promoting factor) in Xenopus oocyte extracts (Marlovits et al. 1998), which we refer to herein as the frog egg model (see Fig. 1). Such networks are often represented as graphs where vertices represent substrates and products (collectively referred to as species), and labeled directed edges connecting vertices represent the reactions. Chemical reactions cause the concentrations of the chemical species (C_i) to change in time according to the equation

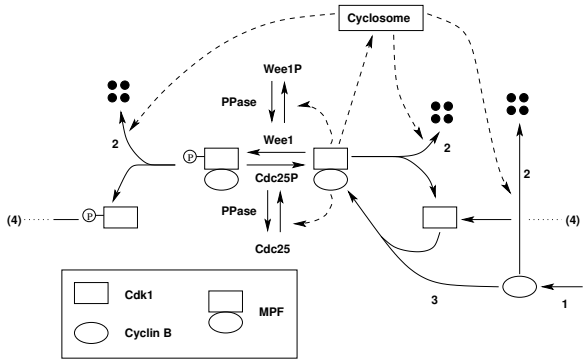

Figure 1: Pathway diagram for the frog egg cell cycle. Cyclin B, synthesized in reaction 1, combines with Cdk1 (reaction 3) to form active MPF. MPF is inactivated by phosphorylation of the Cdk1 subunit by Wee1. Cdc25P reverses the phosphorylation step, converting inactive MPF back to active MPF. Finally, a protein complex (called the cyclosome) degrades cyclin B protein (reaction 2).

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

where R is the number of reactions, v_j is the velocity of the jth 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 does not take part in reaction j).

The full set of rate equations is a mathematical representation of the temporal behavior of the regulatory network. Modelers are faced with many computational problems: accurately and efficiently solving equations when velocities are characterized by widely varied time constants, finding steady state solutions, estimating rate constants by fitting numerical solutions to experimental data, and identifying bifurcation points in the multi-dimensional parameter space.

A realistic model of the budding yeast cell cycle consists of about 30 differential equations containing 100 rate constants (Chen et al. 2004). The parameters are estimated from the cell-cycle behavior of more than 100 mutants defective in the regulatory network. Simulating the entire set takes a few minutes on a desktop PC for one choice of kinetic constants. To fit the model to the mutant data by nonlinear

regression requires thousands of repetitions of the full calculations. A model of such complexity (10-100 equations) is approaching the limit of what a dedicated modeler can produce and analyze with the tools available today. 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. To adequately describe fundamental physiological processes (such as the control of cell division) in mammalian cells will require models of at least 100-1000 equations. To handle this next generation of dynamical models will require sophisticated software to automate the modeling cycle: network specification, equation generation, simulation and data management, and parameter estimation. Ongoing efforts such as the DARPA BioSPICE initiative (DARPA 2005) and the DOE Genomes to Life project (DOE 2005) aspire to support models at least one order of magnitude larger than are currently used.

2 BUILDING LARGE NETWORKS

There is a correlation between the size of a model and the amount of biological information it represents. The ability to construct large biological models provides the potential for better insights into the workings of a cell under investigation, if only we can handle the complexity involved. Models that exist today are small compared to the amount of information known about a particular organism or cellular pathway/process. 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. 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 to start from scratch. This is analogous to adding features to an existing program rather than having to completely re-write it to incorporate new functionality.

Modeling languages and 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 small to medium 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. In this paper we describe features that are intended to enable modelers to create larger models than previously possible. We describe four distinct modeling processes whose purpose is to support the construction of larger models: Fusion, Composition, Aggregation, and Flattening.

Model Fusion is a process that combines two or more models in an irreversible manner. The identities of the individual models (called submodels) being combined are lost, however the aggregate information remains the same. Fusion enables modelers to incorporate information from another model into an existing model, thereby creating larger models. Eventually, fused models will become too large to grasp and manage as single entities. Large models will need to be made up of distinct components to infer any meaningful insight into their underlying biology. Thus, we view model fusion as a useful tool for manipulating small to mid-sized models, but not a viable solution in the long run.

Model Composition provides a potential solution to our limited ability to comprehend larger pathway models. With composition, one can think of models not as monolithic entities, but as collections of smaller components (submodels) joined together in a particular manner. A *Submodel* is a complete model, not an element such as a species, parameter, etc. Given two or more submodels, a composed model is built by describing their inter-model relationships/interactions. Composition is a reversible process, in that removing the inter-model interactions that holds the composed model together recovers the individual submodels.

We distingusih *Model Aggregation* as a restricted form of composition. A collection of model elements is represented as a single entity (a "module"). A module contains a list of input and output ports that link to internal species and parameters. These ports define the module's interface, which provides restricted access to the components in the module. The process of aggregation (connecting modules via their interfaces) 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 result is equivalent to fusing the submodels. However, the relationship information provided by the composition and/or aggregation process should be sufficient to allow the flattening to take place without human intervention (such intervention is needed in the fusion process). The relationships used to describe the interaction between the models and submodels are lost, as the composed or aggregated model is converted into a single large (fused) model. Flattening a model allows us to use existing tools that have no support for composition or aggregation.

3 CONTEXT AND PRIOR WORK

The XML-based Systems Biology Markup Language (SBML) (Hucka et al. 2003) has become widely supported within the pathway modeling community. Thus, we choose to present concrete implementations for the various modeling processes through added SBML language constructs that express the necessary glue that connects submodels to-

gether. It is not necessary that our proposals be implemented in SBML, but doing so provides clear reference implementations in the same way that an algorithm is often expressed in a particular programming language. Fusion is presented in terms of a tool to aid modelers hand-compose large models from smaller components, while flattening a composed or aggregated model will result in a valid SBML Level 2 model without our added language features.

A number of authors find that successful composition or reuse requires components that were designed for composition or reuse (Davis and Anderson 2004, Garlan, Allen, and Ockerbloom 1995, Kasputis and Ng 2000, Malak and Paredis 2004, Spiegel, Reynolds, and Brogan 2005). Bulatewicz, Cuny, and Warman (2004) suggest using a coupling interface for model coupling and provide a number of solutions, from a brute force technique to using frameworks designed to support coupling. Liang and Paredis (2003) describe a port ontology for automated model composition. While automating composition is outside the scope of our work, the ontology for representing ports is useful in detailing the different roles and functions port structures can take.

Proposals have been made within the SBML community (Finney 2003, Ginkel 2003, Schroder and Weimar 2003) that describe the mechanics of composition through additional language features for SBML, as we will do. The common idea among the various pathway modeling composition 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.

Finney (2003) proposes that a model can be composed of instances of submodels, which themselves are full-fledged models. 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. Finney's <Port> structure creates interfaces of components within a model. The <Direct Link> structure enables direct access to components within a submodel without having to define it beforehand. 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 (Schroder and Weimar 2003) 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 dependencies are due to the XSD schema; while semantic dependencies exist due to variations in intuition in English text.

Ginkel (2003) highlights the advantages of modularity and describes features 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. <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 small number of tools exist that provide support for composition (in the context of pathway models) in some form or another. However none of these tools provides any support for composition in SBML. Pathway Builder (Gilman 2003) skirts the issue of model composition: while a user can arrange elements hierarchically in the diagram, the actual model is kept flat. In ProMoT (Ginkel and Krasnyk 2002), 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. Teranode Design Suite (Li 2005) provides support for grouping models together as model collections. 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.

4 MODEL FUSION

Model Fusion is an iterative process to make larger models by merging two or more submodels together. Unlike composition or aggregation (where submodels are not modified but only referenced), fusion takes the the submodels and actually combines them together. In fusion there is no glue (additional SBML language constructs), which describes how submodels are combined. The goal of fusion is to combine submodels into a single unified model containing all the information of the original collection, without any redundancies that might occur across submodels in the original collection. Our approach to fusion is to provide tools that aid modelers attempting to perform the fusion process.

Consider two models, m_1 and m_2, each containing two chemical species (*A* and *B* in m_1, *A* and *D* in m_2). They will be fused together to produce model (m_f). The modeler does this by producing a mapping table for each of the eight SBML component types. They are processed in the following order to avoid conflicting dependencies: (1) Compartments; (2) Species; (3) Function Definitions; (4) Rules; (5) Events; (6) Units; (7) Reactions; and (8) Parameters. A column in a mapping table represents a model and a row represents an SBML component in that model. Duplicate names within a model are not allowed, therefore a species name will only occur once in any particular column. The first column in the mapping table is reserved for the fused model and is referred to as m_f). The two actions available to the modeler during fusion are:

1. define two or more SBML components to be equivalent
2. remove the link/association between two or more SBML components (which have previously been incorrectly linked together) across the different submodels.

The construction of the species mapping table using the example models m_1 and m_2 is shown in Tables 1 and 2; the other mapping tables are constructed using the same process. Each row in the species mapping table corresponds to a distinct species in some submodel. 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 name *A* appears in both models (m_1 and m_2). m_f initially assumes these are the same species in both models (Table 1, Row 1). This might or might not be correct, and can be changed by the modeler if desired. The name of the species in m_f can also be changed. Species *B* and *D* each appear in only one model.

When a modeler defines two species with different names to be equivalent to each other, the two rows are combined. The resulting empty row is automatically deleted, and the modeler selects which name to give this species in the fused model. In our example the modeler defines a new species name *C* for row 2 in the fused model. The resulting m_f contains the species *A* and *C*. If two species (say species *A* in m_1 and species *A* in m_2) were incorrectly identified by the computer as being equivalent to each other, the user can separate them into separate species, each with distinct names. The results of these changes are shown in Table 2.

Table 1: Initial Species Map (with shared species names initially on the same row).

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

Table 2: A completed mapping table for species.

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

5 MODEL COMPOSITION

Model composition describes the process of connecting models, called submodels, together to generate a hierarchy of models (called a composed model) that interact with each other. Larger models can be thought of as a collage of smaller submodels held together by new language features. The language additions for SBML described in this section allow modelers to compose models from submodels, and include support for multiple instances of a given submodel. The features both describe the hierarchy of the submodels, and represent the interactions, relationships, links and reactions between the submodels.

To illustrate model 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 the $Global$ model that represents $x \rightarrow y$, by referring to *x* and *y* in A and B respectively. The $Global$ model consists of a model with only one reaction. The names of reactants and products for that reaction refer to the corresponding species in the two submodels.

Models can be composed together in a number of ways. The first step is to assign or select the global model (the root node in the model tree hierarchy), which can either be one of the submodels or a new model. Once the global model has been assigned, the next step is to specify its list of submodels using the <listOfSubmodels> and <submodels> structures. After the list of submodels have been declared in the global model, the modeler needs to instantiate the submodels in order to use/access them using the <Instance> structure. Finally, different components (species, reactions, etc) within either the submodels or the global model are con-

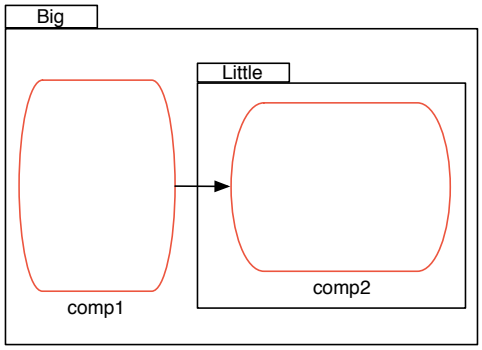

Figure 2: Submodel example showing a link between two compartments

nected/accessed using <link> structures.

We adopt a naming convention to enable modelers to uniquely identify an SBML component (e.g. species, parameters, etc) within a model (or submodel). Our format is:

```
<link>
  <from object="comp1">
  <to object="little">
     <subobject="comp2">
  </to>
</link>
```

We also describe this 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.

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 be a composed model. 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 2) shows how model Big contains a submodel called $Little$, and both models contains a single compartment (*comp1* and *comp2* respectively).

Each <instance> refers to a particular <model>. An <instance> indicates that a copy of a submodel is being instantiated within the current model. Models can be composed of more than one instance of a particular submodel. The instance structure will use the XML Linking Language (XLink) (DeRose, Maler, and Orchard 2001) to refer to submodels, as it is a standard mechanism for linking XML elements inside and outside a given SBML document. XLinks describe links between XML documents. An instance of submodel $Little$ (called *Submodel_Little*) can be made in model Big in order to use/access submodel $Little$ in model Big. The <instance> structure contains attributes `id` (the unique identifier for the <instance>), the XLink's `type`, and the XLink's `href` (an XPointer string that points to either an SBML model document or a model element within the current SBML 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 purposes we only need to link together two objects (resources) and so the value of the *type* field will be $simple$.

A <link> links two entities in separate submodels of a composed model. A <link> should be able to link two <species>, <parameters>, <reactions>, or <compartments> to each other. 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 subobjects of the components using our object/suboject naming convention. The following example shows how the two compartments in Big and $Little$ can be linked together (Figure 2).

```
<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=Little])"/>
  </listOfInstances>
  <listOfLinks>
    <link>
      <from object="comp1"/>
      <to object="Submodel_Little">
        <subobject object="comp2"/>
      </to>
    </link>
  </listOfLinks>
</model>
```

The above example shows an *href* attribute where the submodel $Little$ occurs within the same SBML document. If the submodel $Little$ occured 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.

A <link> structure contains a $merge$ attribute. A value of `true` indicates a $merge$ link; `false` indicates 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 $A = 1.0$. *S1* in Model T has attributes $A = 2.0$ and $B = 3.0$. 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*. Thus, the result is that *S1* has attributes $A = 1.0$ and $B = 3.0$ since it keeps its old value for A and gains the definition for B. If *S1* in R is linked to *S1* in T using a replacement link, then only R.*S1*'s attributes are used. Thus, the result will be that *S1* will have attribute $A = 1.0$.

The <link> structure can link certain combinations of differing SBML component types to each other, such as species $\leftrightarrow$ parameters and rules $\leftrightarrow$ species/parameters. A link can take a <species> structure as the *from object* and a <parameter> structure as the *to object*, and vice versa. An example of this type of link is found when composing the two sample models sharing a degradation reaction *CycB* ($CycB \rightarrow$). In $Model1$ this reaction contains the modifier *Cdc20a*, but in $Model2$, this species does not exist so the reaction instead contains the parameter *A*. In the composed model the species *Cdc20a* from $Model1$ will be linked to the parameter *A* in $Model2$. The reason for this link is because when $Model2$ was created, knowledge about *Cdc20a* was not known so the modeler used the entity (parameter) *A* in their model instead. When $Model1$ was created the modeler had knowledge about the effects of *Cdc20a* on *CycB* degradation. With this additional knowledge it is now desirable to replace *A* with *Cdc20a* when composing (or fusing) the two models together.

6 MODEL AGGREGATION

Naturally occuring molecular networks seem to arise from simpler modules that carry out specific tasks and combine together (Tyson, Chen, and Novak 2003). 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. *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. *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). The fundamental difference between aggregation and composition is the amount of access to model information. 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, model information is not hidden. A modeler can link to any variable or component in a submodel. In aggregation, model information is hidden, and a modeler can only link to variables or components that are explicitly made visible by a given module.

Our models can be viewed as graphs with nodes and edges, where reactions are edges and reactants, products, and parameters are nodes. Interfaces allow access information outside the model they occur in. 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. Like composition, we propose to implement aggregation by means of new language features added to SBML. Most of the language features for composition are used also for aggregation, with the addition of features that allow us to define an interface.

Modelers need not know all reactions that exist within a module to use it, only the list of inputs and outputs. Constraints exist on what can be defined as an input or output to a module. Inputs are parameters (with fixed values), while outputs are species (any reactant or product in any reaction in 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.

To construct a module, a modeler would take a set of reactions and group them together (e.g. by putting them in a new submodel). The modeler then needs to define the input and output ports for the module using the <listOfPorts> structure (discussed below). The input ports can be one or more of the parameters that appear within the set of reactions. The output ports can be one or more of the species that appear within the set of reactions. Once ports are selected, a modeler can use the module by itself or link modules together (by linking the output of one module to the input of another modules) to create more complex models. The link between two <port> structures is unidirectional and is made using the <link> structure from Section 5.

An example of creating and linking modules together to create a complex aggregated model is provided in the following figures from Tyson, Chen, and Novak (2003). A toggle switch (Figure 3.C), a type of two-way discontinuous switch (also referred to as hysteresis) is an example of a mutual inhibition signal response element. It is a model that can be created by linking together two simpler models, a linear response element (Figure 3.A) and a hyperbolic response element (Figure 3.B). Figure 4 shows the same components as icons with input and output ports. Note that the toggle switch created in Figure 4 can now be made into a new model with its own set of inputs and outputs (see Figure 5), by defining its set of port structures.

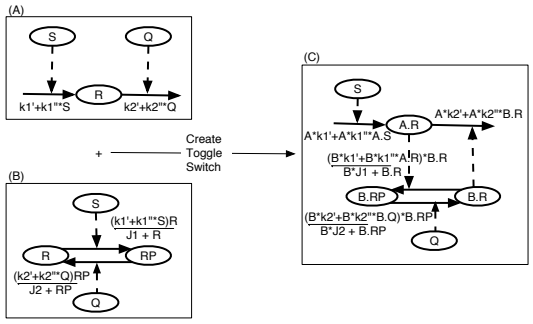

Figure 3: Toggle switch.

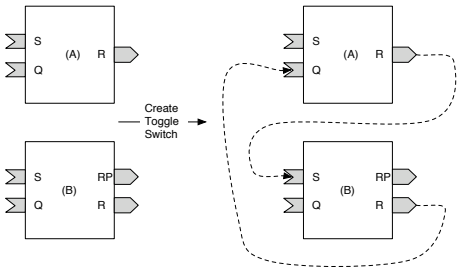

Figure 4: Iconified toggle switch with input and output ports. S and Q are signals, R and RP are responses

The <port> (enclosed in a <listOfPorts> structure) allows a modeler access to a particular species or parameter in a submodel for aggregation. A <port> is composed of two fields: *id* and *target*. The *id* gives a unique identifier to the port. The *target* specifies a single species or parameter by its SBML identifier or by an object reference. The syntax for a port structure is as follows:

```
<listOfPorts>
  <port id="PortId"
        target="SpeciesId or ParameterId"/>
</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 in Section 5. The <listOfSubmodels> and <listOfInstances>

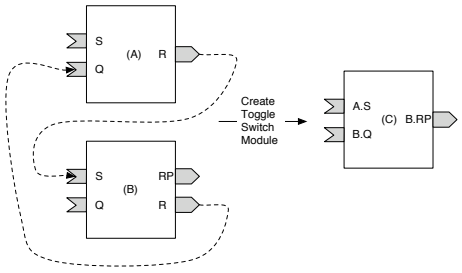

Figure 5: Iconified toggle switch model with its own input and output ports. S and Q are signals, R and RP are responses. From the naming convention mentioned earlier A.S refers to parameter R in model A and B.R refers to species R in model B.

structures are used to define the layout of the aggregated model. Connections between the submodels within the layout is made using the <listOfLinks> structures (Section 5) which can only connect <port> objects to each other.

7 MODEL FLATTENING

Model Flattening is the process of taking a model which contains our additional SBML language constructs (i.e. <listOfSubmodels>, <listOfInstances>, <listOfPorts> and <listOfLinks>), and generating a valid SBML Level 2 model (i.e., with our language constructs removed). The flattening process is done automatically, using the information provided by the composition/aggregation glue to perform the steps that otherwise are done by hand when a modeler fuses models (as described in Section 4).

Flattening has three steps: separation, saving and resolution. During separation, first 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 (Section 4). 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/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.

8 CONCLUSIONS

Model composition is widely viewed within the systems biology community as a prerequisite for building significantly larger pathway models. Such scaling up is necessary for building models of mammalian cells. However, we note that none of these proposals have been published in the peer-reviewed literature, nor to our knowledge have any been implemented, whether in terms of SBML extensions with suitable support tools, or in any other model representation language. 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 in this application domain. Model composition for pathway models remains very much an open problem.

Our proposals for model composition are unique in recognizing the distinctions beween model fusion, composition, aggregation, and flattening. We have begun implementing the fusion tool described above, and are implementing the SBML language features necessary to support model com-

position and aggregation. We hope to present these tools to the modeling community in the near future.

ACKNOWLEDGMENTS

We thank the reviewers of this paper, who greatly contributed to the clarity of our presentation. This work was supported by NSF Biocomplexity Program, Grant No. MCB-0083315, NIH Grant 1 R01 GM64339-01, DARPA and Air Force Research Lab, Air Force Materiel command, USAF, under agreement F30602-02-0572.

REFERENCES

Bulatewicz, T., J. Cuny, and M. Warman. 2004. The potential coupling interface: Metadata for model coupling. In *Winter Simulation Conference*, 183–190.

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

DARPA 2005. DARPA BioSPICE website. Available at `community.biospice.org`.

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

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

DOE 2005. US Department of Energy Genomes to Life website. Available at `doegenomestolife.org/`.

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

Garlan, D., R. Allen, and J. Ockerbloom. 1995. Architectural mismatch or why it's hard to build systems out of existing parts. In *International Conference on Software Engineering*, 179–185.

Gilman, A. 2003. PathwayBuilder. Available at `biospice.lbl.gov/PathwayBuilder/`.

Ginkel, M. 2003. Modular SBML Proposal for an Extension of SBML towards Level 2. In *Proceedings of 5th Forum on Software Platforms for Systems Biology*.

Ginkel, M., and M. Krasnyk. 2002. ProMoTDIVA. Available at `www.mpi-magdeburg.mpg.de/research/projects/1002/comp_bio/promot/distrib`.

Hucka, M., A. Finney, H. Sauro, and 40 additional authors. 2003. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* 19 (4): 524–531.

Kasputis, S., and H. C. Ng. 2000. Model composability: formulating a research thrust: composable simulations. In *Winter Simulation Conference*, 1577–1584.

Li, Z. 2005. Teranode Design Suite. Available at `teranode.com/products/index.php`.

Liang, V. C., and C. J. J. Paredis. 2003. Foundations of multi-paradigm modeling and simulation: a port ontology for automated model composition. In *Winter Simulation Conference*, 613–622.

Malak, R. J., and C. J. J. Paredis. 2004. Foundations of validating reusable behavioral models in engineering design problems. In *Winter Simulation Conference*, 420–428.

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

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

Spiegel, M., P. Reynolds, and D. Brogan. 2005. A case study of model context for simulation composability and reusability. In *Winter Simulation Conference*, 437–444.

Tyson, J. J., K. C. Chen, and B. Novak. 2003. Sniffers, Buzzers, Toggles and Blinkers: Dynamics of Regulatory and Signaling Pathways in the Cell. *Current Opinion in Cell Biology* 15:221–231.

AUTHOR BIOGRAPHIES

CLIFFORD A. SHAFFER is an associate professor in the Department of Computer Science at Virginia Tech since 1987. He received his PhD from University of Maryland in 1986. His current research interests include problem solving environments, bioinformatics, component architectures, visualization, algorithm design and analysis, and data structures. His Web address is `www.cs.vt.edu/shaffer`.

RANJIT RANDHAWA is a PhD candidate in the Department of Computer Science at Virginia Tech. He received BS degrees in Computer Science and Genetic Biology from Purdue University, and an MS degree in Computer Science from Virginia Tech. His research interests include software design, systems biology, synthetic biology, computational biology, bioinformatics and modeling and simulation.

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