FUSING AND COMPOSING MACROMOLECULAR REGULATORY NETWORK MODELS

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

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

Keywords: Computational Biology, Systems Biology Markup Language (SBML), pathway models

Abstract

Today's macromolecular regulatory network models are small compared to the amount of information known about the corresponding cellular pathways, in part because current modeling languages and tools are unable to handle significantly larger models. Most pathway models are small models of individual pathways which are relatively 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 can benefit from reusing existing components, there currently exists little tool or representational support for combining or composing models. In this paper we present a tool for merging two or more models (we call this process *model fusion*) and a concrete proposal for implementing composition in the context of the Systems Biology Markup Language (SBML).

REGULATORY NETWORK MODELING

Macromolecular regulatory network models attempt to deduce physiological properties of a cell from wiring diagrams of its control systems. An example is the set of reactions controlling the activity of MPF (mitosis promoting factor) in Xenopus oocyte extracts [16], which we refer to herein as the frog egg model (see Figure 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

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

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 takes no part in reaction j).

The full set of rate equations is a mathematical representation of the temporal behavior of the regulatory network. A

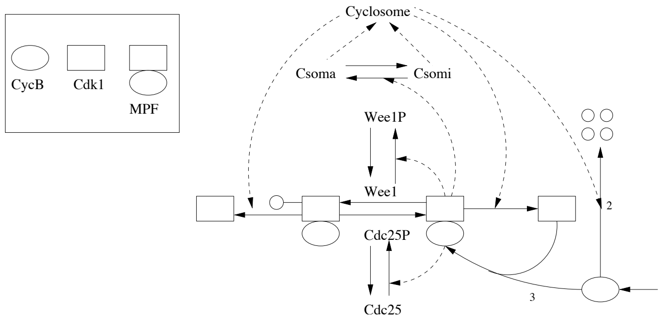


Figure 1. Pathway diagram for the frog egg cell cycle. Cyclin B, synthesized in reaction 1, combines with Cdk1 (reaction 3) to form active MPF. MPF is inactivated by phosphorylation of the Cdk1 subunit by Wee1. Cdc25P reverses the phosphorylation step, converting inactive MPF back to active MPF. Finally,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

realistic model of the budding yeast cell cycle consists of over 30 differential equations and 100 rate constants [3]. 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. Ongoing efforts such as the DARPA BioSPICE initiative [4] and the DOE Genomes to Life project [7] aspire to support models at least one order of magnitude larger than are currently used.

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 the corresponding organism or cellular pathway/process, on the order of 10s of species and/or reactions. 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, increasing the size and complexity by an order of magnitude, 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 since it is done manually. The level of complexity is difficult to deal with as the number of models and their sizes increase. Efficiently running simulations and parameter estimation for models becomes even more of a concern as model size and complexity increase. Our work is intended to be a first step for scaling up to larger problems. When making large models it is helpful to start from existing models and reuse information, rather than start from scratch. Using existing models also ensures that the newly created model will be consistent with the experimental data. One can assume that each of the submodels used in creating the larger model is in fact a validated model with experimental data that fixes its parameters. The main motivation for creating larger models is because there exists new data that the current (sub)models cannot explain or describe. A simple yet effective method to verify the biological accuracy of the newly created model is to ensure that it is consistent with both the older submodel data as well as the new data.

Modeling languages and tools help modelers construct their models by providing a computational environment 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 techniques that are intended to enable modelers to create larger models than previously possible. Our prior work has identified a number of modeling processes related to model composition [18]. In this paper we describe two distinct modeling processes whose purpose is to support the construction of larger models: Fusion and Composition.

Model Fusion is a process that combines two or more models in an irreversible manner. In fusion, the identities of the individual models (called submodels) being combined are lost, but the aggregated information remains the same. Fusion enables modelers to incorporate information from one model into another model, thereby creating larger models. Eventually, fused models will become too large to grasp and manage as single entities. Large models will ultimately need to be made up of distinct components to infer any meaningful insight into their underlying biology. Thus, while model fusion as a useful tool for manipulating small to mid-sized models, it is 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. 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.

CONTEXT AND PRIOR WORK

The XML-based Systems Biology Markup Language (SBML) [9, 12] 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 together. It is not necessary that our proposals be implemented in SBML, but doing so provides clear reference implementations in the same way as an algorithm expressed in a particular programming language. Fusion is presented in terms of a tool to aid modelers hand compose large models from smaller components.

A number of authors find that successful composition or reuse requires components that were designed for the purpose [5, 10, 13, 15, 19]. Bulatewicz, et al. [2] 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 [14] 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 [8, 11, 17] that describe the mechanics of composition through additional language features for SBML, as we will do. However, we note that 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 in this application domain, or any publications describing proprietary features in commercial applications. Model composition for pathway models remains very much an open problem.

MODEL FUSION

Model Fusion is an iterative process to make larger models by merging two or more submodels together. Unlike composition (where submodels are referenced but not modified), fusion takes the the submodels and actually makes changes to them as part of the process of combining them together. 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.

Sample models

The chromosome cycle is divided into four phases (G1, S, G2 and M), with two irreversible transitions (*Start* and *Finish*). The two transitions are irreversible due to the creation and destruction of stable steady states of the molecular regulatory mechanism by dynamic bifurcations [1, 20]. A network of molecular signals control events in the cell cycle (cyclin-dependent protein kinases). The *Start* transition separates G1 from S; once the cell passes this transition it commits itself to DNA synthesis. *Start* is triggered by the protein kinase, Cdk (referred to using its cyclin partner CycB in the sample models). At *Start*, cyclin synthesis is induced and cyclin degradation is inhibited. This causes a rise in Cdk activity which is needed for DNA synthesis. The *Finish* transition separates M from G1, and occurs when DNA replication is complete. Once the cell enters the *Finish* transition, it commits itself to cell division. *Finish* is accomplished by activating a group of proteins that make up the anaphase-promoting complex (APC; also known as the cyclosome), which labels specific proteins for degradation. The APC contains two auxiliary proteins, Cdc20 and Cdh1, whose role (when active) is to recognize cyclins and present them to the complex for labeling (and degradation), which allows the system to return to G1. Cdc20 and Cdh1 are controlled differently by cyclin-Cdk, which activates Cdc20 and inhibits Cdh1.

The two sample models in Figure 2 were obtained from [20] and describe how the cell cycle engine is regulated in eukaryotic cells. *Model*1 describes the effects of Cdh1/APC, Cdc20/APC and cyclin-Cdk on each other. *Model*2 describes the effects of a cyclin-dependent kinase inhibitor (CKI) on CycB.

Consider the two sample models, *Model*1 and *Model*2 which will be fused together to produce model (*FusedModel*). The modeler does this by producing a mapping table for the various SBML component types. During this processing we must avoid dependencies across components which might exist as some components are referenced in other components. For examples, we must resolve the identities of compartments (which represent the bounded

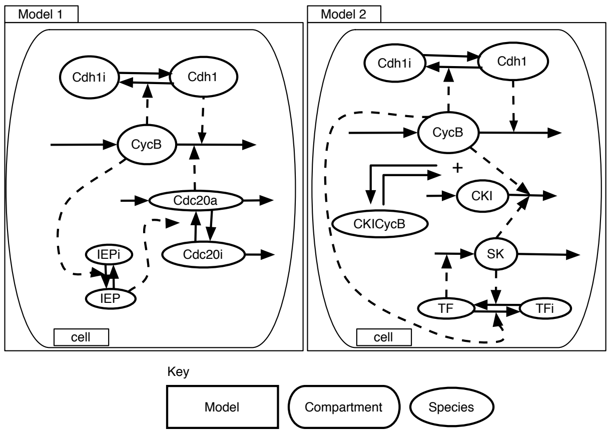


Figure 2. Sample models

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 such conflicting dependencies: (1) Compartments; (2) Species; (3) Function Definitions; (4) Rules; (5) Events; (6) Units; (7) Reactions; and (8) Parameters.

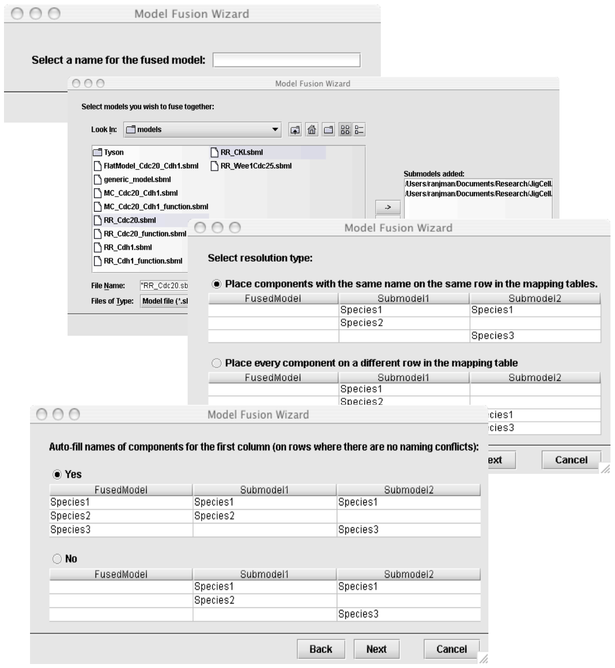
A column in a mapping table represents a model, and each row represents an SBML component in that model. Duplicate names within a model are not allowed. Therefore, a species name will only occur once in any particular column. The first column in the mapping table is reserved for the fused model and is referred to as *FusedModel*). 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.

Fusion Prototype

The fusion prototype in Figure 3 follows a wizard interface paradigm, where information is solicited from the user in a step by step process. Fusion consists of two parts: setup and resolution. During setup the modeler is guided through various steps that initialize the application. The modeler first assigns a name to the fused model, then selects the list of models to fuse together from a file chooser. The next screen allows the modeler to select a control option for the mapping tables: the system either (1) places components of the same name on the same row or (2) places each component on a different row. The auto-fill screen attempts to minimize the amount of work needed by the modeler 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 change them.

Once the environment has been initialized, the modeler



● Place components with the same name on the same row in the mapping tables.

FusedModel	Submodel1	Submodel2
	Species1	Species1
	Species2	
		Species3

○ Place every component on a different row in the mapping table

FusedModel	Submodel1	Submodel2
	Species1	
	Species2	
		cies1
		cies3

Auto-fill names of components for the first column (on rows where there are no naming conflicts):

● Yes

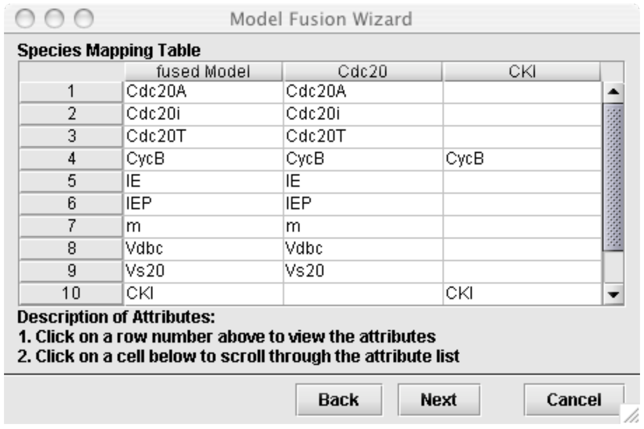
FusedModel	Submodel1	Submodel2
Species1	Species1	Species1
Species2	Species2	
Species3		Species3

○ No

FusedModel	Submodel1	Submodel2
	Species1	Species1
	Species2	
		Species3

Figure 3. Fusion wizard setup screens.

starts resolving the naming conflicts in the various submodels. Resolving names is vital to be able to unambiguously identify all the distinct entities within a model and their relationships with each other. The initial species mapping table is shown in Figure 4.



Species Mapping Table

	fused Model	Cdc20	CKI
1	Cdc20A	Cdc20A	
2	Cdc20i	Cdc20i	
3	Cdc20T	Cdc20T	
4	CycB	CycB	CycB
5	IE	IE	
6	IEP	IEP	
7	m	m	
8	Vdbc	Vdbc	
9	Vs20	Vs20	
10	CKI		CKI

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

Figure 4. Fusion application initial species mapping table. Shared species names are on the same row.

A simple example helps to explain how the mapping tables are created. 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). The initial species mapping table is shown in Table 1 (mapping tables for the other seven SBML components are constructed using the same process). Each row in the species mapping table corresponds to a distinct species in some submodel. The

Table 1. Initial Species Map.

	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

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, to replace B in m_1 and D in m_2. 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.

Once the SBML component mapping tables have been generated, the application uses this information to automatically merge the submodels together. The fused model is now created from the reaction networks of the submodels. Figure 5 shows how $Model1$ has been fused with $Model2$. Note that there are no duplicated species or reactions.

MODEL COMPOSITION

Another approach to building larger models is to connect submodels together to generate a hierarchy of models. We call this a composed model. Larger models can be thought of as a collage of smaller submodels. Within the context of SBML, we add new language features to describe the relationships between submodels. We refer to such constructs as "glue." 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

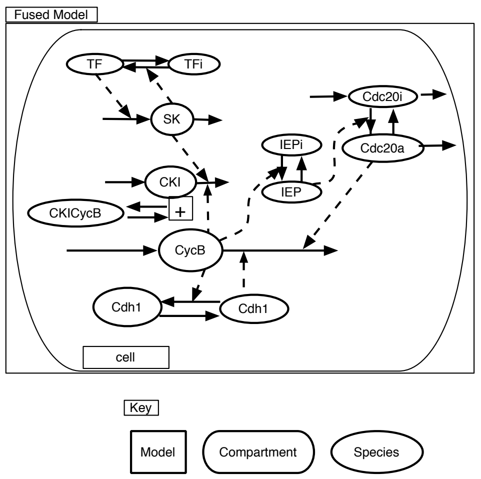


Figure 5. The fused model

(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. *Global* 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. It should be noted that adding a new reaction (or any new component) is not performed in the fusion tool, instead this action is accomplished in the model building environment used to create the (sub)models.

It turns out that there are significant similarities between model fusion and model composition, as we discovered during the process of developing the fusion tool. We had originally conceived of fusion and composition as fairly unrelated processes. However, the fusion process described in the last section defines a series of steps taken to merge two or more models together. This series of steps is captured by our fusion wizard tool, and 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 results, as the output of both fused and composed models should be identical during simulation. 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 composed/connected together.

The first step in composition 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. This requires extending the fusion wizard’s file chooser functionality to allow selecting which of the submodels will become the composed model.

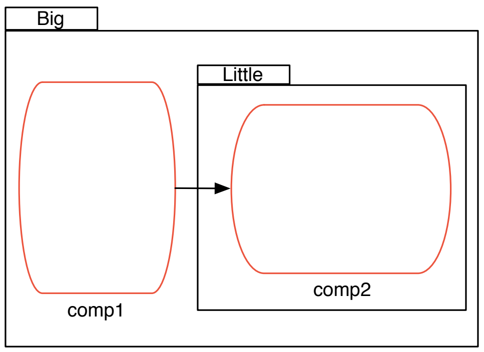


Figure 6. Submodel example showing a link between two compartments

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 be a composed model. Since there is no restriction on the number of submodels a model can contain, a <model> structure is enclosed in a <listOfSubmodels> structure. A simple example (Figure 6) shows how model *Big* contains a submodel called *Little*, and both models contains a single compartment (*comp1* and *comp2* respectively).

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 connected/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:

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

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.

Each <instance> (enclosed in a <listOfInstances> structure) 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 uses the XML Linking Language (XLink) [6] 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 example we only need to link together two objects (resources) and so the value of the *type* field will be *simple*.

A <link> (enclosed in a <listOfLinks>) 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. Linking components in composition can be achieved by using the mapping tables created during fusion. Components on the same row in the mapping table will be linked together. Functionality to describe the type of link must be added to the current fusion mapping table to better represent the unidirectional relationship between linked components. 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 6).

```
<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* 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.

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 $A = 1.0$. *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 (i.e., the *merge* attribute is false), then only *R.S1*'s attributes are used. Thus, the result will be that *S1* will have attribute $A = 1.0$. Specifying the type of link for composition requires including an additional 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 $\leftrightarrow$ 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 *Model*1 this reaction contains the modifier *Cdc20a*, but in *Model*2, this species does not exist so the reaction instead contains the parameter *A*. In the composed model the species *Cdc20a* from *Model*1 will be linked to the parameter *A* in *Model*2. The reason for this link is because when *Model*2 was created, knowledge about *Cdc20a* was not known so the modeler used the entity (parameter) *A* in their model instead. When *Model*1 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.

FUTURE PLANS

We will investigate two additional modeling processes whose purpose is to support the construction of larger models. *Model Aggregation* is 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. It is possible that model aggregation will prove to be more intuitive to modelers who are constructing large models from scratch with components designed to be aggregated, rather than composing existing models that have incompatibilities.

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 since this information is unknown to the fusion tool). 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.

REFERENCES

[1] M.T. Borisuk and J.J. Tyson. Bifurcation analysis of a model of mitotic control in frog eggs. *Journal of Theoretical Biology*, 195(1):69–85, 1998.

[2] T. Bulatewicz, J. Cuny, and M. Warman. The potential coupling interface: Metadata for model coupling. In *Proceedings of the 2004 Winter Simulation Conference*, pages 183–190, 2004.

[3] 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.

[4] DARPA. Darpa biospice website. Available at `community.biospice.org`, 2005.

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

[6] S. DeRose, E. Maler, and D. Orchard. Xml linking language (xlink) version 1.0 w3c recommendation. Available at `www.w3.org/TR/xlink`, 2001.

[7] DOE. Us department of energy genomes to life website. Available at `doegenomestolife.org/`, 2005.

[8] A. Finney. 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`, 2003.

[9] A. Finney, M. Hucka, and H. Bolouri. Systems biology markup language (sbml) level 2: Structures and facilities for model definitions. Available at `sbml.org/specifications/sbml-level-2/version-1/html/sbml-level-2.html`, 2002.

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

[11] M. Ginkel. Modular sbml proposal for an extension of sbml towards level 2. In *Proceedings of 5th Forum on Software Platforms for Systems Biology*, 2003.

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

[13] S. Kasputis and H.C. Ng. Model composability: formulating a research thrust: composable simulations. In *Proceedings of the 2000 Winter Simulation Conference*, pages 1577–1584, 2000.

[14] V.C. Liang and C.J.J. Paredis. Foundations of multiparadigm modeling and simulation: a port ontology for automated model composition. In *Proceedings of the 2003 Winter Simulation Conference*, pages 613–622, 2003.

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

[16] 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.

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

[18] C.A. Shaffer, R. Randhawa, and J.J. Tyson. The role of composition and aggregation in modeling macromolecular regulatory networks. In *Proceedings of the 2006 Winter Simulation Conference*, Dec. 2006.

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

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

AUTHOR BIOGRAPHIES

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.

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`.

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.