SBML Level 3 Package Proposal

Hierarchical Model Composition

Authors

Lucian P. Smith
lpsmith@u.washington.edu
Department of Bioengineering
University of Washington
Seattle, WA, US

Michael Hucka
mhucka@caltech.edu
Engineering and Applied Science
California Institute of Technology
Pasadena, CA, US

Contributors

Stefan Hoops
shoops@vbi.vt.edu
Virginia Bioinformatics Institute
Blacksburg, VA, US

Nicolas Le Novère
lenov@ebi.ac.uk
European Bioinformatics Institute
Wellcome Trust Genome Campus
Hinxton, Cambridge, UK

Andrew Finney
andrew.finney@oxfordcc.co.uk
(independent)
UK

Ion Moraru
moraru@panda.uchc.edu
Center for Cell Analysis and Modeling
University of Connecticut Health Center
Farmington, CT 06030, US

Martin Ginkel
ginkel@mpi-magdeburg.mpg.de
Max Planck Institute for Dynamics of Complex Technical Systems
Magdeburg, DE

Wolfram Leibermeister
lieberme@molgen.mpg.de
Max Planck Institute for Molecular Genetics
Berlin, DE

Ranjit Randawa
rrandhawa@vt.edu
Department of Computer Science
Virginia Tech
Blacksburg, VA, US

Jonathan Webb
jwebb@bbn.com
BBN Technologies
10 Moulton Street
Cambridge, MA, US

Draft of 11 May 2011

Disclaimer: This is a working draft of the SBML Level 3 "comp" package proposal. It is not a normative document. Please send comments and other feedback to the mailing list sbml-discuss@caltech.edu.

Proposal tracking number

Number 2404771 in the SBML issue tracking system.

Version information

Version number and date of public release

Version 1 released 9 October 2010.

Version 2 released 25 April 2011.

URL for this version of the proposal

https://sbml.svn.sourceforge.net/svnroot/sbml/trunk/specifications/sbml-level-3/version-1/comp

URL for previous versions of this proposal

http://sbml.svn.sourceforge.net/viewvc/sbml/trunk/specifications/sbml-level-3/version-1/comp/Hierarchical%20Model%20Composition%20Proposal.pdf?view=log

There have been numerous previous proposals for Hierarchical Model Composition. The most recent proposal preceding the current document, and serving as a source of inspiration and ideas, is the Hoops et al. 2007 proposal, available at the following URL:

http://sbml.org/Community/Wiki/SBML_Level_3_Proposals/Hierarchical_Model_Composition_%28Hoops_2007%29

1. Introduction and motivation

In the context of SBML, hierarchical model composition refers to the ability to include models as submodels inside other models. The goal is to support the ability of modelers and software tools to do such things as (1) decompose larger models into smaller ones, as a way to manage model complexity; (2) incorporate multiple instances of the same component model within one or more enclosing models, to avoid literal duplication of repeated elements; and (3) create libraries of vetted models, much as is done in software development and other engineering fields. This document describes a proposal for an SBML Level 3 package to support hierarchical model composition. Figure 1 illustrates some of the scenarios targeted by this proposal.

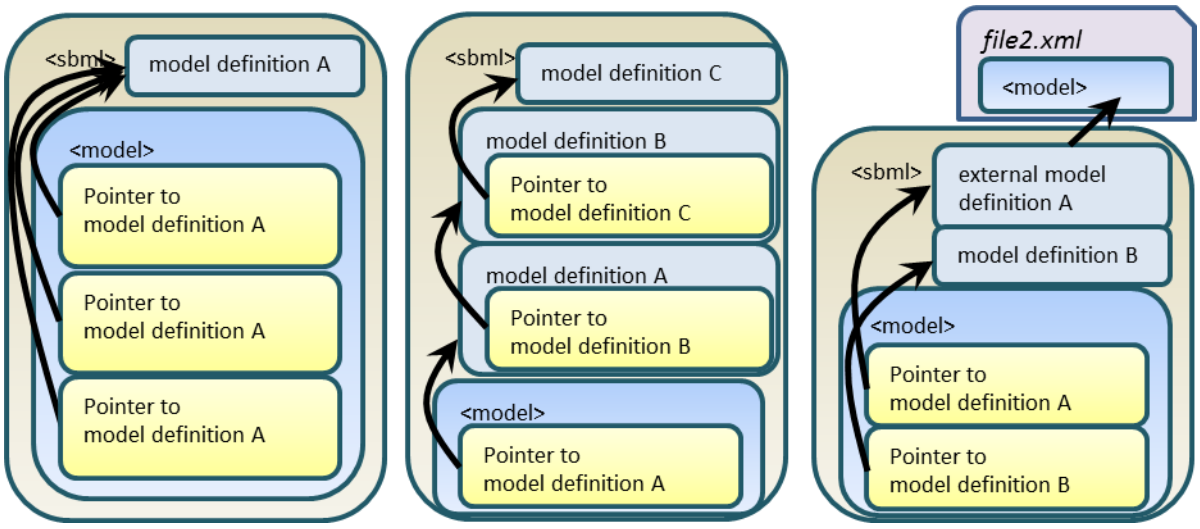

Figure 1: Model composition scenarios. From left to right: a model composed of multiple instances of a single, internally-defined submodel definition; a model composed of a submodel that is itself composed of submodels; and a model composed of submodels, one of which is defined in an external file.

The effort to create a hierarchical model composition mechanism in SBML has a long history, which we summarize in the next section. It has also been known by different names. Originally, it was called *modularity* because it allows a model to be divided into structural and conceptual modules. It was renamed *model composition* when it became apparent that the name "modularity" was too-easily confused with other notions modularity, particularly XHTML 1.1 modularity[1] (which concerns decomposition into separate files). To make clear that the purpose is structural *model* decomposition, regardless of whether the components are stored in separate files, the SBML community adopted the name SBML Hierarchical Model Composition.

To support a variety of composition scenarios, this package provides for optional black-box encapsulation by means of defined data communication interfaces (here called *ports*). In addition, it also separates model *definitions* (i.e., blueprints, or templates) from *instances* of those definitions, it supports optional external file storage, and it allows recursive model decomposition with arbitrary submodel nesting.

---

1 "XHTML 1.1—Module-based XHTML", http://www.w3.org/TR/xhtml11/, W3C, 31 May 2001.

2. Background

2.1 Problems with current SBML approaches

SBML Level 3 Core has no direct support for allowing a model to include other models as submodels. Software tools either have to implement their own schemes outside of SBML, or (in principle) one could also use annotations to augment a plain SBML Level 2 model with the necessary information to allow a software tool to compose a model out of submodels. However, such solutions would be proprietary and tool-specific, and not conducive to interoperability. There is a clear need for an official SBML language facility for hierarchical model composition.

2.2 Past work on this problem or similar topics

The SBML community has discussed the need to add model composition features to SBML since its very beginning, some ten years ago. In an internal discussion document titled "Possible extensions to the Systems Biology Markup Language"[2] principally authored by Andrew Finney (and, notably, written even before SBML Level 1 Version 1 was finalized in March of 2001), the first of the four titular possible extensions is for "submodels". In that document, the main model object can contain a list of submodels, each of which are model definitions only, and a list of submodel instantiations, each of which are references to elements in the submodel list. Finney's proposal also extends the syntax of SBML identifiers (the SId data type) to allow entity references using a dotted notation, in which `X.y` signifies element `y` of submodel instance `X`; the proposal also defines a form of linking model elements through "substitutions". The proposal also introduced the concept of validation through what it called the "expanded" version of the model (now commonly referred to as the "flattened" form, meaning translation to a plain SBML format that does not use composition features): if the flat version of the model is valid, then the model as a whole must also be valid.

In June of 2001, at the Third Workshop on Software Platforms for Systems Biology, Martin Ginkel and Jörg Stelling presented their proposal titled "XML Notation for Modularity"[3], complete with an accompanying proposal document and sample XML file, partially in response to deficiencies or missing elements they believed existed in the proposal by Finney. In their proposal, Ginkel and Stelling present a "classic view" of modularity, where models are packaged as black boxes with interfaces. One of their design goals is to support the substitution of one module for another with the same defined interface, thereby supporting the simplification or elaboration of models as needed. Their proposal emphasizes the reuse of models and with the possibility of developing libraries of models.

Martin Ginkel presented an expanded version of that proposal[4] in the July 2002 Fifth Workshop on Software Platforms for Systems Biology, in the hope that it could be incorporated into the

---

2 http://sbml.svn.sourceforge.net/viewvc/sbml/trunk/specifications/sbml-level-3/old/sbml-team-proposals/original/sbmlext.pdf
3 Presentation at http://sbml.org/images/7/73/Vortrag.pdf, pre-meeting proposal at http://sbml.org/images/3/3d/Joerg-sbml-proposals.pdf, and sample file at http://sbml.org/images/1/18/Sbml2.txt
4 http://sbml.org/images/9/90/Sbml-modular.pdf

definition of SBML Level 2 that was being developed at the time. This proposal clarified the need to separate model definitions from model instantiations, and, further, the need to designate one model per document as the "main" model.

In March of 2003, an independent proposal[5] by Jonathan Webb was posted to the sbml-discuss mailing list. This proposal included a unified, generic approach to making links and references to elements in submodels using XPath. Previous proposals used separate mechanisms for species, parameters, compartments, and reactions. Webb also raised the issue of how to successfully resolve conflicting attributes of linked elements, debated whether formal interfaces were necessary or even preferable to directly access model elements, discussed type checking for linkages, and discussed issues with unit incompatibilities. Around this time, Martin Ginkel formed the Model Composition Special Interest Group[6], a group that eventually reached 18 members, including Jonathan Webb.

Model composition did not make it into SBML Level 2 when that specification was released in June of 2003, because the changes between SBML Level 1 and Level 2 were already substantial enough that software developers at the time expressed a desire to delay the introduction of composition to a later revision of SBML. Andrew Finney (now the co-chair of the Model Composition SIG) presented yet another proposal[7] in May of 2003, even before SBML Level 2 Version 1 was finalized, that aimed to add model composition to SBML Level 3. With only two years having passed between SBML Level 1 and Level 2, the feeling at the time was that Level 3 was likely to be released in 2005 or 2006, and the model composition proposal would be ready when it was. However, Level 2 ended up occupying the SBML community longer than expected, with four versions of Level 2 produced to adjust features in response to user feedback and developers' experiences.

In the interim, the desire to develop model composition features for SBML continued unabated. Finney revised his 2003 proposal in October 2003[8]; this new version represented an attempt to synthesize the earlier proposals by Ginkel and Webb, supplemented with his own original submodel ideas, and was envisioned to exist in parallel with another proposal by Finney, for arrays and sets of SBML elements (including submodels)[9]. Finney attempted to resolve the differences in the two basic philosophies (essentially, black-box versus white-box encapsulation) by introducing optional "ports" as interfaces between a submodel and its containing model, as well as including an XPath-based method to allow referencing model entities. The intention was that a modeler who wanted to follow the classic modularity (black-box) approach could do so, but other modelers could still use models in ways not envisioned by the original modeler simply by accessing a model's elements directly via XPath-based references. In both schemes, elements in the submodels were replaced by corresponding elements of the containing model. Finney's proposal also provided a direct link facility that allows a containing model to refer directly to submodel elements without providing placeholder elements in the containing model. For example, a containing model could have a reaction that converts a species in one submodel to a species in a different submodel, and in the

---

5 http://sbml.org/Forums/index.php?t=msg&th=67&rid=0#msg_111
6 http://www.mpi-magdeburg.mpg.de/zlocal/martins/sbml-comp
7 http://www.mpi-magdeburg.mpg.de/zlocal/martins/sbml-comp/model-composition.pdf
8 http://sbml.org/images/7/73/Model-composition.pdf
9 Posted to sbml-discuss, http://sbml.org/Forums/index.php?t=msg&th=234&rid=0#msg_683

direct-link approach, it would only need to define the reaction, with the reactant and product being expressed as links directly to the species defined in the submodels.

After Finney's last effort, activities in the SBML community focused on updates to SBML Level 2, and since model composition was slated for Level 3, not much progress was made for several years, apart from Finney including a summary of his 2003 proposal and of some of the unresolved issues in a poster[10] at the 2004 Intelligent Systems for Molecular Biology (ISMB) conference held in Glasgow.

Finally, in June of 2007, unplanned discussions at the Fifth SBML Hackathon[11] prompted the convening of a workshop specifically to revitalize the model composition package, and in September of 2007, the SBML Composition Workshop[12] was held at the University of Connecticut Health Center, hosted by the Virtual Cell group and organized by Ion Moraru and Michael Blinov. This event produced several artifacts, still available online:

1. Martin Ginkel provided a list of goals for model composition[13], including use cases, and summarized many of the issues discussed in the present document to this point, including the notion of definition versus instantiation, linking, referencing elements that lack SBML identifiers, and the creation of optional interfaces. It also mentioned the need of allowing parameterization of instances (i.e., setting new numerical values that override the defaults), and the need to be able to "delete" or elide elements out of submodels. (He also provided a summary of ProMoT's model composition approach and a summary of other approaches[14].)
2. Andrew Finney wrote up a list of issues and comments, recorded on the meeting wiki page[15]; these included some old issues as well as some new ones:
   - There should perhaps be a flag for ports to indicate whether a given port must be overloaded.
   - There should be support for *N-to-M* links, when a set of elements in one model are replaced as a group, conceptually, with one or more elements from a different model.
   - The proposal should be generic enough to accommodate future updates and other SBML Level 3 packages.
3. Wolfram Liebermeister presented his group's experience with SBMLMerge[16], dealing with the pragmatics of merging multiple models. As far as this proposal goes, he noted that the annotations in a composed model need to be considered, particularly since they can be crucial to successfully merging models in the first place.

---

10 http://sbml.org/images/9/9c/Ismb-2004-sbml-level-3-poster.pdf
11 http://sbml.org/Events/Hackathons/The_5th_SBML_Hackathon
12 http://sbml.org/Events/Other_Events/SBML_Composition_Workshop_2007
13 http://sbml.org/Events/Other_Events/SBML_Composition_Workshop_2007/Martin_Goals
14 http://ntcnp.org/twiki/bin/view/VCell/OoModelingPromot
15 http://sbml.org/Andrew_2007_Comments_about_Model_Composition
16 http://sbml.org/images/c/c1/SemanticSBML_SBMLcomposition.pdf

4. On behalf of Ranjit Randhawa, Cliff Shaffer summarized Ranjit's work in the JigCell group on model fusion, aggregation, and composition[17]. Highlights of this presentation and work include the following:

   - A description of different methods which all need some form of model composition, along with the realization that model fusion and model composition, though philosophically different, entail exactly the same processes and require the same information.
   - A software application (the JigCell Composition Wizard) that can perform conversion between types. The application can, for example, promote a parameter to a species, a concept which had been assumed to be impossible and undesirable in previous proposals.
   - The discovery that merging of SBML models should be done in the order Compartments → Species → Function Definitions → Rules → Events → Units → Reactions → Parameters. If done in this order, potential conflicts are resolved incrementally along the way.

5. Nicolas le Novère created a proposal for SBML modularity in Core[18]. This is actually unrelated to the efforts described above; it is an attempt to modularize a "normal" SBML model in the sense of divvying up the information into modules or blocks, rather than composing a model from different chunks. It was agreed at the workshop that this is a completely separate idea, and while it has merits, should be handled separately.

6. As a collective, the group produced an "Issues to Address" document[19], with several conclusions:

   - It should be possible to "flatten" a composed model to produce a valid SBML Level 3 Core model, and all questions of validity can then be simply applied to the flattened model. If the Core-only version is valid, the composed model is valid.
   - The model composition proposal should cover both designed-ahead-of-time as well as ad-hoc composition. (The latter refers to composing models out of components that were not originally developed with the use of ports or the expectation of being incorporated into other models.)
   - The approach probably needs a mechanism for deleting SBML model elements. The deletion syntax should be explicit, instead of being implied by (e.g.) using a generic replacement construct and omitting the target of the replacement.
   - It should be possible to link any part of a model, not just (e.g.) compartments, species and parameters.
   - The approach should support "object overloading"[20] and be generally applicable to all SBML objects. However, contrary to what is provided in the JigCell Composition Wizard, changing SBML component types is not supported in object overloading.

---

[17] Presentation at http://sbml.org/documents/proposals/CCB2007DemoPresentation.pdf and publication at http://en.scientificcommons.org/53559395

[18] http://sbml.org/Events/Other_Events/SBML_Composition_Workshop_2007/Modularity_In_Core

[19] http://sbml.org/Events/Other_Events/SBML_Composition_Workshop_2007/Issues_To_Address

- A proposition made during the workshop is that elements in the outer model always override elements in the submodels, and perhaps that sibling linking be disallowed. This idea was hotly debated.
- Interfaces (ports) are indeed considered helpful, but should be optional. They do not need to be directional as in the electrical engineering "input" and "output" sense—the outer element always overrides the inner element, but apart from that, biology does not tend to work in the directional way that electrical components do.
- The ability to refer to or import external files may need a mechanism to allow an application to check whether what is being imported is the same as it was when the modeler created the model. The mechanism offered in this context was the use of MD5 hashes.
- A model composition approach should probably only allow whole-model imports, not importing of individual SBML elements such as species or reactions. The reason is that model components are invariably defined within a larger context, and attempting to pull a single piece out of a model is unlikely to be safe or desirable.
- The model composition approach must provide a means to handle the conversion of units, so that the units of entities defined in a submodel can be made congruent with the entities that refer to them in the enclosing model.

During the workshop, the attendees collectively worked on a rough draft proposal. Stefan Hoops acted as principal editor. The proposal for the SBML package (which was renamed the *Hierarchical Model Composition*[21]), was issued a mere one day after the end of the workshop. It represented an attempt to summarize the workshop as a whole, and provide a coherent whole, suitable as a Level 3 package. It provided a brief overview of the history and goals of the proposal, as well as several UML diagrams of the proposed additions. Stefan presented[22] the proposal in August of 2008 at the 13th SBML Forum, and subsequently gave the same presentation again at the 7th SBML Hackathon in March of 2009 and at the 14th SBML Forum in September of 2009, in a continuing effort to raise interest.

Roughly concurrently, Herbert Sauro, one of the original developers of SBML, received a grant to develop a modular human-readable model definition language, and hired Lucian Smith in November of 2007 to work on the project. Sauro and Frank Bergmann, then a graduate student with Herbert, had previously written a proposal[23] for a human-readable language that provided composition features, and this was the design document Lucian Smith initially used to create a system that was eventually called *Antimony*. Through a few iterations, the design[24] eventually settled on was very similar in concept (largely by coincidence) to that developed by the group at the 2007 Connecticut workshop: namely, with model definitions placed separately from their instantiations in other models, and with the ability to link (or "synchronize", in Antimony terminology) elements of models with each other. Because Antimony was designed to be "quick and dirty", it allowed type conversions much like the JigCell Composition Wizard,

---

[20] http://sbml.org/Events/Other_Events/SBML_Composition_Workshop_2007/Overloading_Semantics
[21] http://sbml.org/Community/Wiki/SBML_Level_3_Proposals/Hierarchical_Model_Composition_(Hoops_2007)
[22] http://sbml.org/images/e/e9/HierarchicalModelGothenburg.pdf
[23] http://www.sys-bio.org/sbwWiki/_media/sbw/standards/2006-12-17_humanreadable_mdl.pdf
[24] http://antimony.sourceforge.net/Tutorial.pdf

whereby a parameter could become a species, compartment, or even reaction. Synchronized elements could end up with aspects of both parent elements in their final definitions: if one element defined a starting condition and the other how it changed in time, the final element would have both. If both elements defined the same aspect (like a starting condition), the one designated the "default" would be used in the final version. Smith developed methods to import other Antimony files and even SBML models, which could then be used as submodels of other models and exported as flattened SBML.

At the 2010 SBML-BioModels.net Hackathon, in response to popular demand from people at the workshop, Smith put together a short presentation[25] about model composition and some of the limitations he found with the 2007 proposal. In particular, he proposed the separation of the replacement concept (where old references to replaced values are still valid) from the deletion concept (where old references to replaced values are no longer valid). Smith wrote a summary of that discussion, added some more of thoughts, and posted it to sbml-discuss[26]. In this posting, he proposed and/or reported several possible modifications to the Hoops et al. 2007 proposal, including the following:

- Separation of *replacement* from *deletion*.
- Separation of model definitions from instantiations.
- Elimination of ports, and the use of annotations instead.
- Annotation of N-to-M replacements, instead of giving them their own construct.

The message to sbml-discuss was met with limited discussion. However, it turns out that several of the issues raised by Smith were brought up at the 2007 meeting, and had simply been missed in the generation of the initial (and incomplete) proposal placed on the wiki after the workshop. The group had, for example, originally preferred to differentiate deletions from replacements more strongly than by simply having an empty list of replacements, and omitted them simply because no better method could be found. Similarly, the separation of definitions from instantiations had been a part of every proposal up until 2007, and was mentioned in the notes for that meeting. The decision to merge the two was a last-minute design decision brought about when the group noted that if the 'xinclude' construct was used, the separation was not strictly necessary from a technical standpoint.

Smith joined the SBML team part-time in September of 2010, and was tasked with going through the old proposals and synthesizing from them a new version that would work with the final incarnation of SBML Level 3. That version (the first version of this document) was presented at COMBINE in October 2010[27], and further discussed on the sbml-discuss mailing list. At HARMONY in April of 2011, consensus was reached on a way forward for resolving the remaining controversies surrounding the specification, resulting in the current version of this document.

---

[25] http://sbml.org/images/b/b6/Smith-Hierarchical_Model_Composition-2010-05-03.pdf
[26] http://www.sbml.org/Forums/index.php?t=tree&goto=6124
[27] http://precedings.nature.com/documents/5133/version/1

2.3 Genesis of the current proposal

A candidate Level 3 Version 1 Core specification was not released until the end of 2009, and it was only in October of 2010 that the final Level 3 Version 1 Core specification was released. As a consequence of the lack of a concrete, finalized SBML Level 3 Core specification, all of the model composition efforts up to this point have been theoretical: they could not define a precise syntax as long as the underlying SBML Level 3 syntax was not finalized. The few SBML-compatible software tools that *did* implement some form of composition had to do so using proprietary approaches or extensions to SBML. This has now changed, thanks to the finalization of SBML Level 3 Version 1 Core, and the present proposal is an attempt to blend features of previous efforts into a concrete, Level 3-compatible syntax.

The current proposal was written from scratch, but draws strongly on the Hoops 2007 and Finney 2003 proposals, as well as, to some degree, every one of the sources mentioned above. Some practical decisions are new to this proposal, sometimes due to additional design constraints resulting from the final incarnation of SBML Level 3, but all of them draw from a wealth of history and experimentation by many different people over the last decade. Where this proposal differs from the historical consensus, the reasoning is explained, but for the most part, the proposal follows the road most traveled, and focuses on being clear, simple, only as complex as necessary, and applicable to the largest number of situations.

2.4 Design goals

The following are the basic design goals followed in this proposal:

- *Allow modelers to build models by aggregation, composition, or modularly*. These methods are so similar to one another, and the process of creating an SBML Level 3 package is so involved, that we believe it is not advantageous to create one SBML package for aggregation and composition, and a separate package for modularity. Users of the hierarchical model composition package should be able to use and create models in the style that is best suited for their individual tasks, using any of these mechanisms, and to exchange and reuse models from other groups simply and straightforwardly.
- *Interoperate cleanly with other packages*. The rules of composition should be such that they could apply to any SBML element, even unanticipated elements not defined in SBML Level 3 Core and introduced by some future Level 3 package.
- *Allow models produced with these constructs to be valid SBML if the constructs are ignored*. As proposed by Nicolas le Novère[28] and affirmed by the SBML Editors[29], whenever possible, ignoring elements defined in a Level 3 package namespace should result in syntactically-correct SBML models that can still be interpreted to some degree, even if it cannot produce the intended simulation results of the full (i.e., interpreting the package constructs) model. For example, inspection and visualization of the Core model should still be possible.
- *Ignore verbosity of models*. We assume that software will deal with the "nuts and bolts" of reading and writing SBML. If there are two approaches to designing a mechanism for this hierarchical composition package, where one approach is clear but verbose and the other

[28] http://www.sbml.org/Forums/index.php?t=tree&goto=6104
[29] http://sbml.org/Events/SBML_Editors'_Meetings/Minutes/2010-06-22

approach is concise but complex or unobvious, we prefer the clear and verbose approach. We assume that software tools can abstract away the verbosity for the user. (However, tempering this goal is the next point.)

- *Avoid over-complicating the specification*. Apart from the base constructs defined by the proposal, any new element or attribute being introduced should have a clear use case that cannot be achieved in any other way.
- *Allow modular access to files outside the modeler's control.* In order to encourage direct model referencing (such as to models hosted online on sites such as biomodels.net), whenever possible, we will require referenced submodels only to be SBML Level 3 core, and not require that they include constructs from this proposal.
- *Incorporate most, if not all, of the desirable features of past proposals*. The names may change, but the aims of past efforts at SBML model composition should still be achievable with the present proposal.

3. Proposed Syntax and Semantics

3.1 Color conventions

Throughout this document, in all UML diagrams, classes that exist in SBML Level 3 core are pictured in black. If those elements are extended in this proposal, those extensions are displayed in green. Classes that are new to this proposal are shown in blue.

3.2 Models and Submodels

At its heart, Hierarchical Model Composition is about models that you can include in other models. When you define a model for inclusion, we call this a ModelDefinition, and when you actually include it, we call this a Submodel. A definition is like a platonic ideal: it may be a complete model in and of itself, but in this file, it exists only as a concept. A submodel is an instantiation or instance of the previously-defined model: it is the realization of that model inside another model. From the perspective of the model that contains it, it has come into being, and now exists as something that can be modified and adapted. If that containing model is the Model object of the SBML file, it has been fully instantiated. If that containing model is itself a ModelDefinition, it becomes part of that larger model, but has not been fully instantiated in the SBML file until that definition is itself instantiated in the Model.

Previous proposals tended to call ModelDefinitions 'submodels'. We avoid the term here because these models must be valid SBML Model objects in and of themselves, and may indeed have started life that way. (Instead, we are using the term 'submodel' for the instance of the model inside the containing model.) Another possible term was 'ModelTemplate', which was close, but implied that the model was not complete, and needed to be 'filled in' by the containing model. While this situation is possible, it is not required, as for example in the case of model aggregation, when several complete working models are merged to form a larger whole.

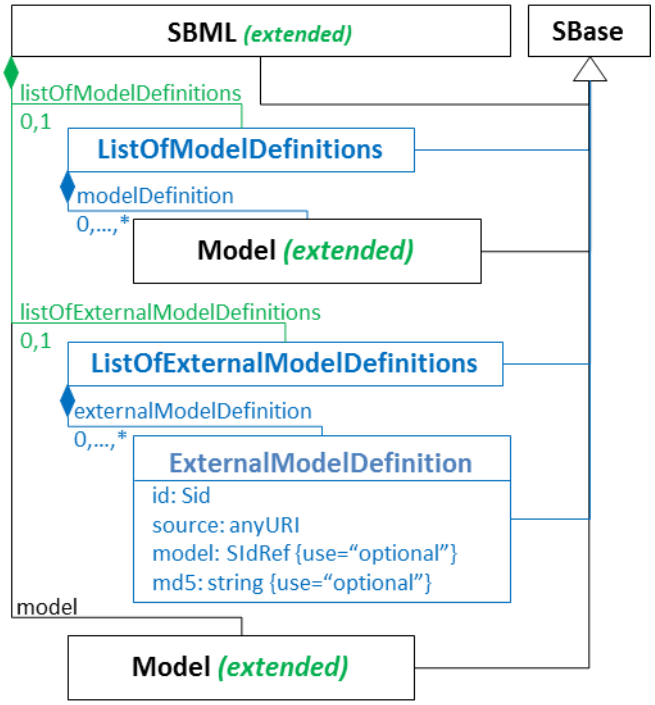

***Figure 2**: The definition of the extended SBML class, and the new ListOfModelDefinitions, ListOfExternalModelDefinitions, and ExternalModelDefinition classes. We here extend the SBML class by having including a ListOfModelDefinitions child, which may contain any number of Models (modelDefinitions), and a ListOfExternalModelDefinitions child, which may contain any number of ExternalModelDefinitions. The modelDefintions are the same as the original 'model' child of SBML, which this package extends as well (defined below). The ExternalModelDefinitions are references to model objects in other files. The 'id' is required (so it can be referred to in Submodels); the 'source' attribute describes the location of a SBML Level 3 document, and the 'model' the id of the model object in that document. If 'model' is not present, the model child of the SBML document is used. The optional md5 string attribute may be additionally used to ensure the referenced document has not changed.*

Because the model definitions are not 'owned' by any other model (as they can be instantiated anywhere), we have here pulled them out of the Model entirely, and instead make a new child of the SBML object itself: a new class, ListOfModelDefinitions, was created to hold them.

If a model from a different document is needed, it can be referenced here with the ExternalModelDefinition class. The external document is found with the 'source' attribute as a URI, as described by the W3C document RFC3986[30]. URIs may therefore either be URLs (such as web addresses), URNs (such as those defined by the 'urn:' protocol), or relative or absolute file locations. This must point to an SBML Level 3 Version 1 document, though this means the entire file at that location, including any xml ('<?xml>') and comment ('<!-- >') elements, and not just the SBML document itself ('<sbml>').

Because the 'id' of a Model object of the target document is optional, the model in the referenced file might not be named—if no 'model' attribute is used, the main model of the referenced file is used (the 'model' object that is the child of the 'SBML' object). If the 'model'

[30] http://tools.ietf.org/html/rfc3986

attribute is used, the Model or ModelDefintion in the referenced file with the given ID is imported, whether this is the main model or a model listed in that file's ListOfDefinedModules.

The SId namespace for SIds defined within ModelDefintions (as Species, Parameters, etc.) follows the same rules as defined in the core specification for the model-wide level, but need not be unique on a document-wide level: two different ModelDefinitions may each define a Parameter with the same SId, for example, without either violating the core specification's uniqueness requirement, or implying that the two elements are 'the same' within this document.

Similarly, the SIds for Models, ModelDefinitions, and ExternalModelDefinitions within a single SBMLDocument must be unique within that document, but need not be unique across documents (referenced or otherwise), nor need they be different from other named elements of other Models. In other words, a model with the id 'mod1' may not contain a parameter with the id 'mod1' (according to the core specification), nor may there be any other models with the id 'mod1' in the same file. But if one model has the id 'm1', this id would not conflict with a parameter with the id 'm1' in a different model in the same file. This is referred to as the 'model namespace of the document'.

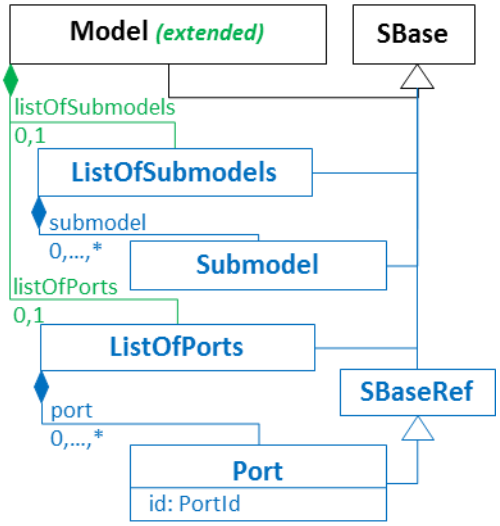

***Figure 3**: Definition of the extension of the Model class and of the new ListOfSubmodels, ListOfPorts, and Port classes. The Model class now may contain a single ListOfSubmodels, which in turn may contain any number of Submodel objects, defined below. It may also contain a single ListOfPorts, which in turn may contain any number of Port objects, which are SBaseRef objects (defined below) with the addition of a required 'id' attribute.*

The Model class is extended to potentially contain a single ListOfSubmodels and a single ListOfPorts. The ListOfSubmodels contains Submodel objects which will instantiate model definitions inside the containing model. The ListOfPorts contains Port objects which point the other way: they guide this model's interactions with models that contain it. The concept of ports are discussed more below in the 'Ports' section.

A Model that contains Submodels should be considered to itself contain those elements (such as species, reactions, compartments, rules, and events) contained in the submodel. This is the heart of this proposal: combining models together to make new models.

A Submodel object must say which model it is an instantiation of, and may additionally contain information about how one must modify it to make sense in its new context. The model it instantiates may either be another model in this file, or it may be a model defined in a separate file. The only restriction is that it may not contain loops: it may not refer to its parent model, nor may it refer to a model which in turn instantiates its parent model, etc.

The direct modifications are of two types: conversion factors, and deletions. If the math of the referenced model must be changed in this context, this is handled through conversion factors. Deletions are for when an element in the referenced model no longer makes sense in its new context: a duplicated degradation rate of a species, for example, or a no-longer-relevant InitialAssignment, or even a particular event assignment.

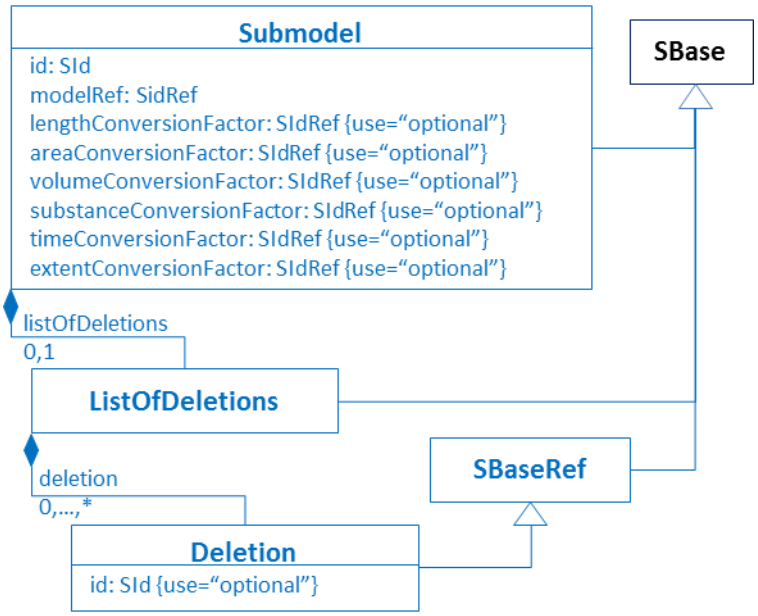

Figure 4: Definition of Submodel, ListOfDeletions, and Deletion. The Submodel object must contain an id and a modelRef attribute with an SidRef of a model in this document. It may additionally contain up to six different conversion factor attributes and a ListOfDeletions object. The conversion factors, if present, must reference parameter objects from the parent model. The ListOfDeletions may contain any number of Deletions, which are of the class SBaseRef (defined below) with the addition of an optional 'id', so that they can be referenced if needed in the parent model.

The Submodel's ID attribute is required so that other references may always have a method through which a parent model may refer to this submodel's elements (to link and replace them). This ID must follow the normal restrictions on SBML SIds for uniqueness within models, though they may not be used within core elements that do not know about this package.

The 'model' SidRef must be the 'id' of any other Model, ModelDefinition, or ExternalModelDefinition defined in this document (the 'model namespace of this document'). It is even legal, though unlikely, to refer to the <model> child of the SBMLDocument in this way, meaning that the file contains a ModelDefinition that itself contains (and presumably modifies) the model it presents to the world as the main model associated with this file. Perhaps the main model defines a common scenario, and alternate initial conditions are defined in the ListOfModelDefinitions, for example. Because the model namespace is defined per

document, this means that it is possible to define and include a new model namespace by creating a new document, then importing one or more of those models using the ExternalModelDefinition class.

The six possible conversion factors must be references to Parameter objects in the parent model which describe how to convert subelements whose units are inherited from the units of the base submodel. The intricacies of how this works out in practice are complex; for more detail, see the 'Conversion Factors' section.

The imported model may in turn contain submodels imported from the same or from other external files. This chain should, of course, be followed, with the same caveat that no loops are allowed, whether internally or across files.

The list of deletions is for removing elements from the submodel that no longer belong in the parent model, and do not need to be referenced by the submodel. We will discuss deletions and replacements in the next section, along with ports.

3.3 SBaseRefs

Thus far, we have described how to aggregate models together, and synchronize them so that their math is compatible. A simulation of the resulting model would be equivalent to simulating the parent model and the submodels separately, scaling them appropriately, and then overlaying the results from each on the same graph. This is usually not sufficient. What is needed is a way to tell the simulator that *this* element of one submodel is the same as *this other* element of a second submodel. In the case of a species, one submodel may control its creation and destruction, and the second may define how its presence modulates the rate of a related reaction. It may have even been modeled as a constant parameter in the second submodel, if its concentration never changed under the conditions it was designed to imitate. The new parent model may be an attempt to relax the assumptions in the submodels, and create a more complicated and robust model of the system being studied.

To do this, we must be able to refer to elements of the submodels. The construct to do this is called an 'SBaseRef'.

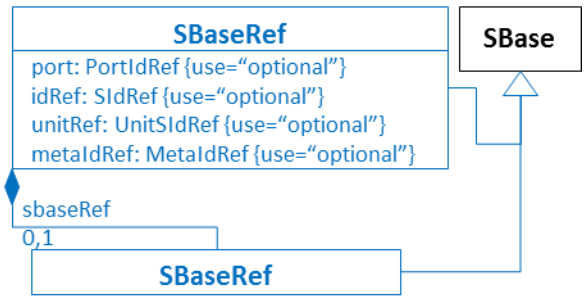

***Figure 5**: Definition of SBaseRef. A SBaseRef must contain exactly one of the attributes 'port', 'idRef', 'unitRef' or 'metaIdRef'. If this refers to a submodel itself, the element may additionally contain a SBaseRef child, to refer to a particular element of that submodel, instead of referring to the submodel itself.*

An SBaseRef object references an element of a submodel by referencing a particular element in one of several ways. The multiple options available are because the referenced submodel may

belong to an external file beyond the control of the modeler, and the preferred methods of referencing its subobjects may not be available.

There are five different ways of referencing a subelement, in order of preference:

- **By port.** We will talk about how to set up ports below, but if the ModelDefinition has set up a system of ports, they are referenced here. A fully modular model will only use ports, which are the defined interfaces between models and submodels. The namespace of the PortSIdRef is the port namespace of the submodel, not the parent model (see section 3.7; 'Identifier Scoping').
- **By SId**. Most elements one would want to replace (such as species and reactions) will have SIds. If they do not have ports, you can reference them by this. The SIdRef namespace is the namespace of the submodel, not the parent model, and refers to that model's element namespace (see section 3.7; 'Identifier Scoping').
- **By Unit SId**. The SId of a UnitDefinition is defined in the core specification to exist in its own namespace. Therefore, this attribute is provided to be able to search that unit namespace of the submodel (see section 3.7; 'Identifier Scoping'). It should be noted that even though this attribute is of type UnitSIdRef, the reserved identifier names that are usually valid for other constructs (see section 3.1.10 of the core specification) are not valid here, as these may not be replaced or deleted.
- **By MetaId**. Because some elements never have SIds and for some they are only optional, the modeler may wish to replace or delete an element that has no SId but does have a MetaID. (Since MetaIDs are optional attributes of SBase, all SBML element have the potential to have a MetaID.) If the element has no port or SId, you may use the metaID, if present.
- **By an element of a submodel**. The above four options will all give you access to elements in a submodel, but cannot give you access to elements in the submodel's submodels. If the object referred to by one of the above methods is itself a submodel, adding an SBaseRef child to the SBaseRef allows you to find elements 'buried' in the hierarchy. This can, in turn, refer to a deeper submodel, allowing access to any element of any arbitrary depth using this construct. This is considered inelegant design: it is better to 'promote' any element in a submodel to a local element if it needs to be referenced by a containing model, but if the submodel is fixed, no other option is available.

This does not actually allow access to any possible element in a model definition that you do not control: if the element in question has no port, id, or metaId, you must create a local copy of that model and add (for instance) a metaId to the element you wish to reference. In a future version of this specification, this restriction may be relaxed, allowing one to reference submodel elements by xpath:element. For now, this was considered to add too much of a burden to implementers of this specification, and was therefore delayed to a future version.

Any element that has been replaced or deleted may not be referenced by an SBaseRef, including anything replaced or deleted within the submodel.

If you replace or delete an element that itself has children, those children are considered to be deleted unless replaced. If you replace a KineticLaw, for example, any annotations that referred to the metaIDs of its LocalParameters will be invalid. To correct this situation, those annotations must themselves be deleted or replaced by valid elements, or the referenced

LocalParameter must be explicitly replaced (by its equivalent in the new KineticLaw, presumably).

It is legal to explicitly delete an element which was deleted by implication in this way if you need to refer to it elsewhere; the resulting model is exactly the same.

3.4 Replacements

Replacements are at the heart of hierarchical modeling, as they are the glue that connects submodels together with each other and with the containing model. Some previous proposals have called this concept 'links', as they link similar elements from different sources. All previous model composition proposals have lumped these things together in lists that were children of the Model class: one list of all replacements (or even all replacements and deletions) between this model and its submodels.

Here, the concept of replacements is distributed to the individual elements that are replacing others. This is accomplished by extending the SBase class itself:

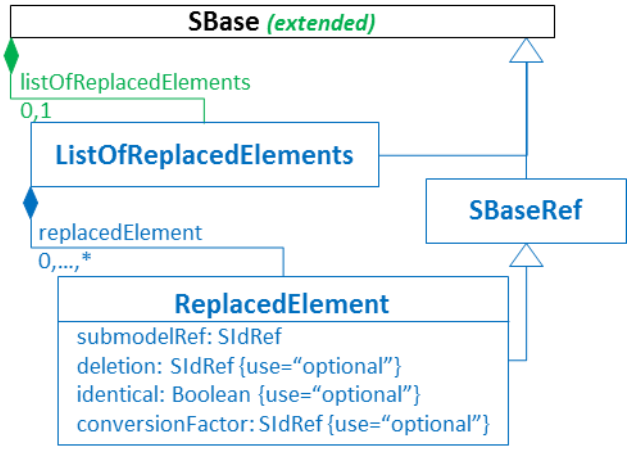

Figure 6: Definition of the SBase extension and the new ListOfReplacements and ReplacedElement classes. SBase is here extended to have a single new optional child, ListOfReplacements, which may contain any number of ReplacedElement children. The ReplacedElement class inherits from SBaseRef, with the additional required attribute 'submodelRef' SIdRef, and the optional attributes 'deletion' (an SIdRef), 'identical' (Boolean), and 'conversionFactor' (an SIdRef).

ReplacedElements are pointers to submodel elements that are being replaced. The ListOfReplacements child of the extended SBase lists everything in all the submodels that this element is replacing. This is a distributed model of defining replacements, and allows a simple way of describing the idea of a replacement that should be fully compatible with all objects in Level 3 core and all potential Level 3 packages. It also avoids the necessity of having to refer to elements that may not have SIds.

When a subelement is replaced, all references to that element are now considered to point to the replacement element. This means that any math in the submodel that contains that subelement's SId will now refer to the replacement SId (as modified by either this element's conversionFactor or the relevant submodel conversion factors—see the next section); any Species ID in a Reaction element will now refer to the replacement element; any annotations

that refer to the replaced subelement's metaid will now refer to the replacement element; and anything else that referred either to the SId or MetaID will now point to the replacement element. This means that when anything refers to a replaced element's SId or MetaID, the replacing element must itself define its own SId or MetaID. It also implies that any subelement may appear in exactly one 'ListOfReplacements': otherwise, old references would not know which of the multiple replacements to choose.

The ReplacedElement's conversionFactor may be used to define how to convert this element's old values in its new context. If present, it overrides any automatic conversion that may have been performed based on the submodel's relevant conversion factors. As this issue quickly gets complicated, it is discussed in detail in the next section.

Under this model for replacement, the element in the parent model always takes precedence over elements from the submodels, and no 'horizontal replacements' are possible that involve only subelements. This decision means that some of the design goals of previous proposals cannot be met, though it is still true that all possible models can still be created. One proposal, for example, wanted to be able to define one species in one submodel, a second species in a second submodel, and then define a reaction that converted one to the other in the containing model. In this scheme, one would have to replace both species with local objects in the containing model, and then create a reaction that converted one to the other. Unfortunately, no other scheme is possible with the current constraint that 'stripped' models remain valid SBML, because Reactions cannot involve anything but Species objects, so we cannot create a 'SpeciesRef' class and use it instead without invalidating Level 3 core. But a scheme would be possible, similar to those proposed before, that would link elements 'sideways' across sibling submodels, and we do not allow this here. Mostly, this is because of our design goal of not worrying about the verbosity of models, but rather ensuring clarity and simplicity. We believe that it is not worth introducing a new construct solely for the purpose of linking parallel elements, when the same task can be accomplished, though more verbosely, in the current scheme.

However, we do want to acknowledge that in a given model, the 'canonical' form of a particular element may lie in a submodel and not in the containing model, and that the only reason a parent model may contain a replacement element is to be able to refer to it. This is the purpose of the optional 'identical' attribute of the ReplacedElement class. Setting this Boolean value to 'true' indicates that the two linked elements are intended to be the same in every respect, so if the two differ in any attribute or subobject, a validation error is produced. This means that even if the subelement sets the optional 'name' property to one value and the replacing element sets it to a different value, the resulting SBML would not be considered valid. Semantic equivalence is required here, not literal equivalence. Numerical values of any attributes such as 'initialValue' must be equivalent to each other as dictated by any relevant conversion factors (see 'Conversion Factors', section 3.5). Similarly, SIds (such as the 'compartment' attribute of a Species) should point to the same element, but should that element have been replaced in the containing model, the literal SId will be different. Finally, the 'id', and 'metaid' attributes, which each exist in different namespaces (and may be required to be different, in the metaid case), may change without breaking identity. If the 'identical' attribute is not set, the validator will warn if an attribute of the replaced element was defined that is not defined at all on the replacement element. If the 'identical' attribute is set 'false', no validation errors or warnings will be produced from any comparison of the two elements.

There is no restriction here that replaced elements must be of the same type as the replacing element. The only restriction is that all old references to the replaced element will now point at the replacing element, so they must continue to make sense and produce valid SBML: a Species that appeared in a Reaction would produce invalid SBML if replaced by a Parameter; if, however, that same Species never appeared in any reactions (or if those reactions were all Deleted), this would result in valid SBML and would be perfectly acceptable. Similarly, a Parameter replaced by a Species will always produce valid SBML, as there are no places where a reference to a Parameter SId could not equally well accept a Species SId. However, don't go overboard with this capability: it is legal in this scheme to replace an Event with a Species, but it is probably never wise. We expect that tools written to produce hierarchical SBML will have their own restrictions that make sense in context. This relaxation of the official validation allows freer intercompatibility with other package extensions—it may be that a Species could be validly replaced by a multi-component species, or it may not, but we will rely here on the normal validation rules that package supplies to dictate the results.

Finally, it is possible to claim that an element replaces a deletion from a submodel. In this case, the 'deletion' attribute is used instead of the normally-required 'port', 'idRef', or 'metaIdRef' attributes from the SBaseRef parent class. The most likely use case for this is in an 'N to M' replacement proposed by Andrew Finney[31]; perhaps an entire pathway is being replaced by a more detailed pathway with more reaction steps. In this case, no one reaction step is replacing any one original reaction step, but the path as a whole is being conceptually replaced. The way to implement this is to delete the original reaction steps from the submodel, and include the new reaction steps in the parent model. If you wish to annotate those deletions, you may list the deletions as being replaced by elements of the new pathway. This has no material effect on the model composition or on the math: it is merely a way to conceptually annotate the modeler's decision-making process. As such, a deletion is the only type of Subelement that may be listed in more than one ListOfReplacements. It is recommended that in the above N to M scenario, all N deleted elements be listed under all M replacement elements, to make things easier on visualization software that may try to display the results.

3.5 Ports

The 'port' concept allows a modeler to design a submodel such that it can be used in a particular way by a containing model: ports are those elements that are designed to be used in replacements or deletions. As seen in Figure 3, Ports are SBaseRef objects with a required 'id' attribute of type PortSId. The PortSId and PortSIdRef types have the same restrictions as the SId and SIdRef types (see section 3.1.7 of the Level 3 Version 1 SBML core specification), but have their own namespace within the parent model, and thus may be the same as the referenced element's SId, should one be present. The model to which the Port refers with its SBaseRef constructs is the parent model of the Port itself. In turn, the port namespace of a model is created from the 'id' attributes of the Ports of a Model, and is searched using the 'port' attribute of an SBaseRef.

[31] http://sbml.org/Andrew_2007_Comments_about_Model_Composition

Each port in a model must refer to a unique element of that model: two ports may not both refer to the same model element. This carries the additional implication that ports may also not refer to other ports of the same model. They may also not (of course) refer to themselves.

As written, this scheme does not have the capability to place restrictions on ports, as may be desired by 'black box' modelers. For example, it might be desirable to tag a port with a Boolean flag saying whether it must be overloaded or not (as Andrew Finney proposed in 2007[32]). In future versions of this specification we plan to add such options. For now, we encourage users to develop their own annotation scheme for ports and add them to Port objects.

3.6 Conversion Factors

In SBML core models, units are optional. Modelers are required to write their models in such a way that all conversions between units are explicit, so that nowhere do units need to be understood and values implicitly converted before use.

Given this package's design goal of compatibility with models that are fixed and unable to be changed, it is not an option to declare the same thing here: that all included models must be written such that they are numerically compatible with each other. If one submodel defines how a species *amount* changes in time, and a second submodel defines an InitialAssignment for that same species in terms of *concentration*, something must be done to make the model as a whole coherent without editing the submodels directly. However, we likewise cannot rely on implicit conversion based on declared units, but must make everything explicit instead.

This is the use of the six different optional conversion factors present on the Submodel class, and the single conversion factor present in the ReplacedElement class. The conversion factor in the ReplacedElement class is relatively straightforward (and overrides the Submodel conversion factors), so we will tackle that one first.

If the submodels of the merged model retain any math (that is, if there are any reactions or assignments or anything with a math element on it that has not been replaced or deleted from the submodel), that math may be enmeshed in a different scale than the math of the containing model. If so, all these math elements should (theoretically) be converted to the new scale. If a replaced element has a defined conversion factor, any time a calculation is performed within the math of the Submodel where the replaced element's SId is found on the left-hand side of an equation, the right-hand side is multiplied by that conversion factor before assignment to that variable. So, if a species has an InitialAssignment of 4x+3, and has a conversion factor of CF, the InitialAssignment become CF*(4x+3). The same is true for AssignmentRules, and RateRules, KineticLaws, EventAssignments, and the implied rates of change of species as calculated from kinetic laws, as described in section 4.11.7 of the core specification.

Conversely, any time the SId of a replaced element appears on the right-hand side of an equation in its original submodel, its appearance in that equation should be divided by the conversion factor. In our previous example of an InitialAssignment rule of '4x+3', if the 'x' had been replaced and given a conversion factor of CFx, that InitialAssignment would become

[32] http://sbml.org/Andrew_2007_Comments_about_Model_Composition

'4(x/CFx)+3'. This holds true for any mathematical equation in the model, including AlgebraicRules.

This also means that if a value appears on the right and left-hand sides of an equation, you must apply the conversion factor twice: if the RateRule of x is "4x+3", it becomes "CFx*(4(x/CFx) + 3)". (Note that this simplifies to "4x + 3*CFx", as you would expect—the 'x' part of the equation is already in the correct scale; it is only the 3 that must be converted.)

The situation becomes trickier when talking about implied units of model elements. SBML Level 2 defined in the specification what the 'base units' of the model were, even if none were defined in the model itself. SBML Level 3 no longer does this, but it retains the concept of unit **types** for certain elements, even if those units are left undefined. A compartment with 'spatialDimensions=3' is of the unit type 'volume', for example, even if exactly what units those are (liters; millicubits^3; etc.) is left undefined. Similarly, all species are either of the unit type 'substance' or 'concentration', depending on the value of the required Boolean attribute 'hasOnlySubstanceUnits'. ('Concentration' is, in turn, defined as 'substance divided by the units of the containing compartment'.) Rate rules of elements are defined as the unit of that element divided by time. The implied equations derived from reactions (see section 4.11.7 of the core specification) are defined as being substance over time. Regardless of what those units are defined to be, or even if those units are left undefined, the unit types are set, and must be consistent throughout the model, so nothing is implicitly converted.

The six Submodel attributes 'lengthConversionFactor', 'volumeConversionFactor', 'areaConversionFactor', 'substanceConversionFactor', 'timeConversionFactor', and 'extentConversionFactor' dictate how any submodel math whose unit types are defined by the Level 3 core specification are to be converted **whether or not that element was replaced**, in the absence of an explicit conversion factor for that element. Thus, all Compartments set 'spatialDimension=1' in the submodel must be converted according to the lengthConversionFactor, with all assignments to that compartment multiplied by the conversion factor, and that compartment's SId divided by it wherever it appears inside a math element. All rates of change of species amounts (defined in section 4.11.7 of the Level 3 specification) are converted by the substanceConversionFactor divided by the timeConversionFactor, after being converted (if necessary) by any internal conversionFactors, as described. All species concentrations from compartments of dimension 2 are converted by the substanceConversionFactor divided by the areaConversionFactor. Non-replaced elements with defined unit types are still converted, so that the output of any simulation will be on the same scale as elements from the containing model.

In the core specification for SBML Level 3, if the conversionFactor attributes for Model and on Species are undefined, the rate of change of species amounts over time is defined to be equal to the rate of extent of the reaction over time, arguably creating a default conversion of extent to amount of 1. Similarly, all conversion factors here effectively default to '1' as well, so that if (for example) 'substanceConversionFactor' is defined but 'areaConversionFactor' is not, species concentrations from compartments of dimension 2 are still converted according to the substanceConversionFactor, 'divided by 1'.

Critically, if an element's unit type cannot be determined, it has no default conversion factor, and one must be set explicitly for the element in question. All Parameters fall in this category,

as parameters may have any unit at all, and have no defined unit type as a class. Similarly, compartments with no spatialDimension set, or set to a fractal spatialDimension such as 2.6 should not be converted automatically. This means that if a Parameter is internal to a submodel and not replaced, there is no way to convert it, and it will remain in its original scale. This will not affect the math of the converted elements, as the rules above first convert all math to the original scale, and only convert it to the new scale when assigning it to a variable. However, if it is displayed as output, these values may be in a different scale from other displayed output. The only way to correct this situation is to replace the parameter in question, and give it an explicit conversion factor.

Some math may use a combination of conversion factors defined on the Submodel with the conversion factors defined explicitly on an element's replacement construct. The simplest example is that of a RateRule that defines how a Parameter changes with time. If the Parameter has been replaced and given a conversion factor, the Parameter's explicit conversion factor is divided by the Submodel's timeConversionFactor to act as the overall conversion factor for the RateRule's math. As a slightly more complicated example, a species concentration that has no explicit conversion factor set for its replacement, and which is contained in a compartment that does have an explicit conversion factor, will be converted according to the substanceConversionFactor from the Submodel divided by the conversion factor defined by the compartment replacement construct.

Species concentrations of species from compartments with undefined unit types will be converted according to the substanceConversionFactor alone, if no conversion factor is defined for its compartment. An odd potential situation arises here in the case where the species' compartment has been actually deleted instead of replaced, the replacement species being put into a new compartment in the containing model. In this case, no automatic conversionFactor is possible, and if one is needed, it must be set explicitly on the species' replacement itself.

Another complication is the situation where a species is set 'hasOnlySubstanceUnits=true' in the submodel, but is set 'hasOnlySubstanceUnits=false' in its replacement, or vice versa. In this case, the species must be converted according to the actual value of its compartment. If an explicit conversion factor is set, it is assumed that the modeler took this into consideration, and created an assignment rule (or similar) such that the conversion parameter would function appropriately. If not, the automatic conversion must use the value for the compartment of the replacement species to convert the species values to amounts from concentrations, and back again. Unreplaced species are still converted, but if they were in amounts before, they remain in amounts afterwards and likewise when in concentrations.

Any math not directly associated with a replaced element and that does not have a defined unit type is assumed to exist in the same scale as all other similar elements across all submodels. The only example of this in the Level 3 core is the math associated with the Priority subelement of Events. A Priority element may be replaced directly by a Replacement construct or by replacing its parent Event, but when comparing Priority expressions from submodels with Priority expressions from the containing model or from other submodels, they are all assumed to be on the same scale relative to each other. (If one model had priorities set on a scale of 0 to 10 and another had priorities set on a scale of -100 to 100, that is just the way it is, and to fix it, all incompatible Priorities would have to be replaced.) The same would be true of math defined in any other Level 3 package without a defined unit type, or with a newly-defined unit type:

none of it would be converted automatically, and all such elements would have to be converted explicitly by being replaced, or that package would have to extend this Hierarchical Model Composition package to define a new attribute on Submodel ('newpack:newPackageUnitTypeConversionFactor') that could be used to automatically convert all such elements in the submodel with that unit type. (If anyone really cared, they could do this for Priority objects, too, and define a priorityConversionFactor. Here, we assume that this capability is too obscure to be desirable, particularly given that Priority objects have just been introduced to SBML, and thus have no divergent traditions to convert between.)

For convenience, here is a table of SBML core elements and concepts, and what conversion factors they use when included as a submodel:

Table 1: A list of all core SBML elements containing math elements, and which conversion factors (if any) are used for them. When different conversion factors are used in different situations, the attributes that define those differences are listed. Listed conversion factors of other elements ('conversion factor of compartment'; 'conversion factor of symbol') use either the automatic conversion factor for the referenced compartment/symbol, or, if defined, the conversion factor used in its replacement.

SBML class	Defined attributes	Automatic conversion factor
AlgebraicRule		1
AssignmentRule		(conversion factor of symbol)
Compartment	spatialDimensions=1	lengthConversionFactor
Compartment	spatialDimensions=2	areaConversionFactor
Compartment	spatialDimensions=3	volumeConversionFactor
Compartment	spatialDimensions undefined or not equal to 1, 2, or 3	1
Constraint		[none--boolean]
Delay		timeConversionFactor
EventAssignment		(conversion factor of symbol)
FunctionDefinition		1
InitialAssignment		(conversion factor of symbol)
KineticLaw		extentConversionFactor/ timeConversionFactor
Implied math for rates of change of species		substanceConversionFactor/ timeConversionFactor
Parameter		1
Priority		1
RateRule		(conversion factor of symbol)/

		timeConversionFactor
Species	hasOnlySubstanceUnits=true	substanceConversionFactor
Species	hasOnlySubstanceUnits=false	substanceConversionFactor/ (conversion factor of compartment)
Species	hasOnlySubstanceUnits=true, replaced by a Species with hasOnlySubstanceUnits=false	substanceConversionFactor/ compartment
Species	hasOnlySubstanceUnits=false, replaced by a Species with hasOnlySubstanceUnits=true	substanceConversionFactor * compartment/ (conversion factor of compartment)
SpeciesReference		1
Trigger		[none--boolean]
<unknown>		1

3.7 Identifier scoping

In section 3.3.1 of the SBML Level 3 version 1 specification, three scopes for element ids are defined: the scope of IDs over the Model, over a Reaction, and for Units within a Model. The introduction of hierarchy to SBML creates a profusion of namespaces, each of which is explained above, but which is summarized here for convenience.

- **SBMLDocument**: The identifier of every Model, ModelDefinition, and ExternalModelDefinition in the document must be unique across the set of all such identifiers in the document. This is referred to as the 'model namespace of the document'.
- **Model and ModelDefinitions**: As in core SBML Level3, the identifiers of every FunctionDefinition, Compartment, Species, Parameter, Reaction, SpeciesReference, ModifierSpeciesReference, Event, and Model, plus the newly-defined classes Submodel and Deletion must be unique across the set of all such identifiers in the Model or ModelDefinition to which they belong. This is referred to as the 'element namespace of the model'. When there are multiple Models or ModelDefintions, this means there are multiple such namespaces, so that when an SIdRef points to this namespace, one must also know to which Model one is referring.
- **Units**: As in Core SBML Level 3, the identifiers of every Unit must be unique across the set of all such identifiers in the Model or ModelDefinition to which they belong. This is referred to as the 'unit namespace of the model'. When there are multiple Models or ModelDefintions, this means there are multiple such namespaces, so that when a UnitSIdRef points to this namespace, one must also know to which Model one is referring.
- **Ports**: A new construct in this package: the identifiers of every Port must be unique across the set of all such identifiers in the Model or ModelDefinition to which they belong. This is referred to as the 'port namespace of the model'. When there are multiple Models or ModelDefintions, this means there are multiple such namespaces, so that when a UnitSIdRef points to this namespace, one must also know to which Model one is referring.

- **Reactions**: As Reactions augment and override the element namespace of a model with the identifiers of LocalParameters, these LocalParameters are not referenceable outside of that Reaction. This is referred to as the 'element namespace of the reaction'. In particular, this means that SBaseRef elements may not reference LocalParameters by id. However, these LocalParameters may still be deleted or replaced by giving them metaIds and referring to those. If replaced, it must be by an element in the normal element namespace of a model (such as a global Parameter). Old references to that replaced LocalParameter will tShen point to the new replacement element. If a LocalParameter is deleted from a Reaction whose KineticLaw used it in its math, that KineticLaw may still be valid if there was an element in the element namespace of the model with that same id to which the math in the KineticLaw can now refer (for example, if the LocalParameter shadowed a global Parameter or Species).

4. Package Dependencies

This package has no dependencies on any other package. It is also designed to work seamlessly with other packages, so one could seamlessly create a set of hierarchical models using Groups or Layout or Spatial (for example). If any incompatibilities are found, please contact the authors of this package.

5. Use cases

The following is a simple aggregate model, with one defined model being instantiated twice:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core"
xmlns:comp="http://www.sbml.org/sbml/level3/version1/comp/version1" level="3" version="1"
comp:required="true">
  <model id="aggregate">
    <comp:listOfSubmodels>
      <comp:submodel comp:id="submod1" comp:modelRef="enzyme"/>
      <comp:submodel comp:id="submod2" comp:modelRef="enzyme"/>
    </comp:listOfSubmodels>
  </model>
  <comp:listOfModelDefinitions>
    <comp:modelDefinition id="enzyme" name="enzyme">
      <listOfCompartments>
        <compartment id="comp" spatialDimensions="3" size="1" constant="true"/>
      </listOfCompartments>
      <listOfSpecies>
        <species id="S" compartment="comp" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false"/>
        <species id="E" compartment="comp" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false"/>
        <species id="D" compartment="comp" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false"/>
        <species id="ES" compartment="comp" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false"/>
      </listOfSpecies>
      <listOfReactions>
        <reaction id="J0" reversible="true" fast="false">
          <listOfReactants>
            <speciesReference species="S" stoichiometry="1" constant="true"/>
            <speciesReference species="E" stoichiometry="1" constant="true"/>
          </listOfReactants>
          <listOfProducts>
            <speciesReference species="ES" stoichiometry="1" constant="true"/>
```

```
          </listOfProducts>
        </reaction>
        <reaction id="J1" reversible="true" fast="false">
          <listOfReactants>
            <speciesReference species="ES" stoichiometry="1" constant="true"/>
          </listOfReactants>
          <listOfProducts>
            <speciesReference species="E" stoichiometry="1" constant="true"/>
            <speciesReference species="D" stoichiometry="1" constant="true"/>
          </listOfProducts>
        </reaction>
      </listOfReactions>
    </comp:modelDefinition>
  </comp:listOfModelDefinitions>
</sbml>
```

Here we have two-step enzymatic process, with S and E forming a complex, then dissociating to E and D. The aggregate model instantiates it twice, so the resulting model "aggregate" has two parallel processes in two parallel compartments performing the same reaction.

Now we will claim that we have saved the above SBML file to the file "enzyme_model.xml", and import the "enzyme" model from it into a new model:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core"
xmlns:comp="http://www.sbml.org/sbml/level3/version1/comp/version1" level="3" version="1"
comp:required="true">
  <model>
    <listOfCompartments>
      <compartment id="comp" spatialDimensions="3" size="1" constant="true">
        <comp:listOfReplacedElements>
          <comp:replacedElement comp:idRef="comp" comp:submodelRef="A" comp:identical="true"/>
          <comp:replacedElement comp:idRef="comp" comp:submodelRef="B" comp:identical="true"/>
        </comp:listOfReplacedElements>
      </compartment>
    </listOfCompartments>
    <listOfSpecies>
      <species id="S" compartment="comp" hasOnlySubstanceUnits="false" boundaryCondition="false"
constant="false">
        <comp:listOfReplacedElements>
          <comp:replacedElement comp:idRef="S" comp:submodelRef="A" comp:identical="true"/>
          <comp:replacedElement comp:idRef="S" comp:submodelRef="B" comp:identical="true"/>
        </comp:listOfReplacedElements>
      </species>
    </listOfSpecies>
    <comp:listOfSubmodels>
      <comp:submodel comp:id="A" comp:modelRef="ExtMod1"/>
      <comp:submodel comp:id="B" comp:modelRef="ExtMod1"/>
    </comp:listOfSubmodels>
  </model>
  <comp:listOfExternalModelDefinitions>
    <comp:externalModelDefinition comp:id="ExtMod1" comp:source="enzyme_model.xml"
comp:model="enzyme"/>
  </comp:listOfExternalModelDefinitions>
</sbml>
```

Here, we again import "enzyme" twice, create a compartment and species local to the parent model, and replace the compartment and species "S" from the two instantiations with the new elements. Now we have a model with a single compartment in which a species S can either bind with enzyme A.E to form A.D, or bind with enzyme B.E to form B.D.

Next, we define one model ("simple") with a single reaction "S -> D" that has ports, and again import "enzyme":

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
      xmlns:comp="http://www.sbml.org/sbml/level3/version1/comp/version1"
      comp:required="true">

  <listOfModelDefinitions xmlns="http://www.sbml.org/sbml/level3/version1/comp/version1">
    <modelDefinition id="simple" name="simple">
      <listOfCompartments>
        <compartment id="comp" size="1" spatialDimensions="3" constant="true"
PortSId="comp_port"/>
      </listOfCompartments>
      <listOfSpecies>
        <species id="S" initialConcentration="5" compartment="comp" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false" PortSId="S_port"/>
        <species id="D" initialConcentration="10" compartment="comp"
hasOnlySubstanceUnits="false" boundaryCondition="false" constant="false" PortSId="D_port"/>
      </listOfSpecies>
      <listOfReactions>
        <reaction id="J0" reversible="true" fast="false" PortSId="J0_port">
          <listOfReactants>
            <speciesReference species="S" stoichiometry="1" constant="false"/>
          </listOfReactants>
          <listOfProducts>
            <speciesReference species="D" stoichiometry="1" constant="false"/>
          </listOfProducts>
        </reaction>
      </listOfReactions>
    </modelDefinition>
  </listOfModelDefinitions>

  <model id="complexified" xmlns="http://www.sbml.org/sbml/level3/version1/comp/version1">
    <listOfSubmodels>
       <submodel id="simple">
         <modelRef model="simple" />
         <listOfDeletions>
           <deletion id="oldrxn" port="J0_port" />
         </listOfDeletions>
       </submodel>
       <submodel id="enzyme">
         <modelRef model="enzyme" />
       </submodel>
    </listOfSubmodels>
    <listOfCompartments>
      <compartment id="comp" size="1" constant="true" spatialDimensions="3">
        <listOfReplacements xmlns="http://www.sbml.org/sbml/level3/version1/comp/version1">
          <replace submodel="simple" port="comp_port" identical="true" />
          <replace submodel="enzyme" symbol="comp" identical="true" />
        </listOfReplacements>
      </compartment>
    </listOfCompartments>
    <listOfSpecies>
      <species id="S" compartment="comp" initialConcentration="5" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false">
        <listOfReplacements xmlns="http://www.sbml.org/sbml/level3/version1/comp/version1">
          <replace submodel="simple" port="S_port" identical="true" />
          <replace submodel="enzyme" symbol="S" identical="false" />
        </listOfReplacements>
      </species>
      <species id="D" compartment="comp" initialConcentration="10" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false">
        <listOfReplacements xmlns="http://www.sbml.org/sbml/level3/version1/comp/version1">
          <replace submodel="simple" port="D_port" identical="true" />
          <replace submodel="enzyme" symbol="D" identical="false" />
        </listOfReplacements>
      </species>
    </listOfSpecies>
  </model>

</sbml>
```

In “simple”, we give ports to the compartment, the two species, and the reaction. Then, in “complexified”, we import both this and the model “enzyme” from “enzyme_model.xml”, and replace the simple reaction with the more complex two-step reaction by deleting the J0 reaction from “simple” and replacing “S” and “D” from both models with local replacements. Note that it is “simple” that defined the initial concentrations of S and D, so our modeler set the ‘identical’ flag to ‘true’ for those elements , faithfully reproducing the 5 and 10 in the local copy, and set the ‘identical’ flag to ‘false’ for the replacement of those elements from “enzyme”. Even had our modeler not done so, no warnings would have been produced, since nothing was defined for “S” or “D” in “enzyme” that was not defined for their replacement elements in “complexified”. Also note that since “simple” defined ports, the ‘port’ attribute was used for the subelements that referenced “simple” model elements, but “symbol” still had to be used for subelements referencing “enzyme”.

In the resulting model, S is converted to D by a two-step enzymatic reaction defined wholly in “enzyme”, with S and D’s initial conditions set, in effect, in “simple” (through the ‘identical’ flag). If “simple” had other reactions that created S and destroyed D, S would be created, would bind with E to form D, and D would then be destroyed, even though those reaction steps were defined in separate models.

But what if we had wanted to annotate that the deleted reaction had been ‘replaced’ by the two-step enzymatic process? To do this, we must move those reactions to the parent model, and, since those reactions involve E and ES, we must also move those as well:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
      xmlns:comp="http://www.sbml.org/sbml/level3/version1/comp/version1"
      comp:required="true">

<listOfModelDefinitions xmlns="http://www.sbml.org/sbml/level3/version1/comp/version1">
  <modelDefinition id="simple" name="simple">
    <listOfCompartments>
      <compartment id="comp" size="1" constant="true" spatialDimensions="3" PortSId="comp"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="S" compartment="comp" initialConcentration="5" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false" PortSId="S_port"/>
      <species id="D" compartment="comp" initialConcentration="10" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false" PortSId="D_port"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="J0" reversible="true" fast="false" PortSId="J0_port">
        <listOfReactants>
          <speciesReference species="S" stoichiometry="1" constant="false"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="D" stoichiometry="1" constant="false"/>
        </listOfProducts>
      </reaction>
    </listOfReactions>
  </modelDefinition>
  <modelDefinition id="enzyme" name="enzyme">
    <listOfCompartments>
      <compartment id="comp" size="1" constant="true" spatialDimensions="3" PortSId="comp_port"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="S" compartment="comp" hasOnlySubstanceUnits="false" boundaryCondition="false"
constant="false" PortSId="S_port"/>
      <species id="E" compartment="comp" hasOnlySubstanceUnits="false" boundaryCondition="false"
constant="false" PortSId="E_port"/>
```

```
      <species id="D" compartment="comp" hasOnlySubstanceUnits="false" boundaryCondition="false"
constant="false" PortSId="D_port"/>
      <species id="ES" compartment="comp" hasOnlySubstanceUnits="false" boundaryCondition="false"
constant="false" PortSId="ES_port"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="J0" reversible="true" fast="false" PortSId="J0_port">
        <listOfReactants>
          <speciesReference species="S" stoichiometry="1" constant="false"/>
          <speciesReference species="E" stoichiometry="1" constant="false"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="ES" stoichiometry="1" constant="false"/>
        </listOfProducts>
      </reaction>
      <reaction id="J1" reversible="true" fast="false" PortSId="J1_port">
        <listOfReactants>
          <speciesReference species="ES" stoichiometry="1" constant="false"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="D" stoichiometry="1" constant="false"/>
          <speciesReference species="E" stoichiometry="1" constant="false"/>
        </listOfProducts>
      </reaction>
    </listOfReactions>
  </modelDefinition>
  </listOfModelDefinitions>

  <model id="complexified" xmlns="http://www.sbml.org/sbml/level3/version1/comp/version1">
    <listOfSubmodels>
       <submodel id="simple">
         <modelRef model="simple" />
         <listOfDeletions>
           <deletion id="oldrxn" port="J0_port" />
         </listOfDeletions>
       </submodel>
       <submodel id="enzyme">
         <modelRef model="enzyme" />
       </submodel>
    </listOfSubmodels>
    <listOfReactions>
      <reaction id="J0" reversible="true" fast="false">
        <replaces>
          <subelement submodel="simple" deletion="oldrxn" />
          <subelement submodel="enzyme" port="J0_port" identical="true" />
        </replaces>
        <listOfReactants>
          <speciesReference species="S" stoichiometry="1" constant="false"/>
          <speciesReference species="E" stoichiometry="1" constant="false"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="ES" stoichiometry="1" constant="false"/>
        </listOfProducts>
      </reaction>
      <reaction id="J1" reversible="true" fast="false">
        <replaces>
          <subelement submodel="simple" deletion="oldrxn" />
          <subelement submodel="enzyme" port="J1_port" identical="true" />
        </replaces>
        <listOfReactants>
          <speciesReference species="ES" stoichiometry="1" constant="false"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="D" stoichiometry="1" constant="false"/>
        </listOfProducts>
      </reaction>
    </listOfReactions>
    <listOfCompartments>
      <compartment id="comp" size="1" constant="true" spatialDimensions="3">
        <replaces>
```

```
          <subelement submodel="simple" port="comp_port" identical="true" />
          <subelement submodel="enzyme" port="comp_port" identical="true" />
        </replaces>
      </compartment>
    </listOfCompartments>
    <listOfSpecies>
      <species id="S" compartment="comp" initialConcentration="5" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false">
        <replaces>
          <subelement submodel="simple" port="S_port" identical="true" />
          <subelement submodel="enzyme" port="S_port" identical="false" />
        </replaces>
      </species>
      <species id="D" compartment="comp" initialConcentration="10" hasOnlySubstanceUnits="false"
boundaryCondition="false" constant="false">
        <replaces>
          <subelement submodel="simple" port="D_port" identical="true" />
          <subelement submodel="enzyme" port="D_port" identical="false" />
        </replaces>
      </species>
      <species id="E" compartment="comp" hasOnlySubstanceUnits="false" boundaryCondition="false"
constant="false">
        <replaces>
          <subelement submodel="simple" deletion="oldrxn" />
          <subelement submodel="enzyme" port="E_port" identical="true" />
        </replaces>
      </species>
      <species id="ES" compartment="comp" hasOnlySubstanceUnits="false" boundaryCondition="false"
constant="false">
        <replaces>
          <subelement submodel="simple" deletion="oldrxn" />
          <subelement submodel="enzyme" port="ES_port" identical="true" />
        </replaces>
      </species>
    </listOfSpecies>
  </model>

</sbml>
```

Here, we have recreated “enzyme” so as to give it ports, then recreated basically the entire model in the parent “complexified” so we can reference the deleted “oldrxn” in the replacements lists. Note that we list that reaction deletion both for the two new reactions and for the two new species “E” and “ES”, since those species were themselves elided in the simple form of the S to D reaction in “simple”. The “identical” flag is used throughout, so that any visualization or manipulation software knows that the only reason those elements exist in the parent model is to create a reference, not to actually change the element itself.

6. Validation

There are two steps to validating a hierarchical model. First, the aggregation and synchronization rules are checked to see if they are well-formed, and if the elements they reference exist. Referenced files and models must exist; ‘ports’ must reference PortSIds in the relevant namespace of the referenced model; xpaths must point to real elements. These sorts of validation rules are discussed in the main document under the relevant constructs.

Second, if one follows the rules of aggregation and synchronization for each model in the SBML file to create a ‘flattened’ model with no hierarchical structure, the resulting model and modelDefintions must all be valid SBML. The one caveat here is namespace issues: if two submodels containing elements with the same SId or metaID (if they came from different files)

are aggregated into the same containing model, they may exist peacefully together in the flattened model.

These validation rules only apply to models defined in the SBML file being validated. Models defined in external files and instantiated as submodels may or may not be valid; the only rule is that the model containing that instantiation must itself be valid. Obviously, malformed XML will be completely unparseable, and the model will be impossible to instantiate at all. But models that leave required flags undefined in its elements may become valid if those elements are replaced by valid elements, or if they are deleted entirely. Similarly, references to nonexistent elements may themselves be deleted, and illegal combinations of objects may be rectified.

The intent behind this rule is that, again, the modeler may not be able to change the model they wish to incorporate, and this gives them a way to create valid SBML while still properly referencing their source.

For translation purposes and to make validation simpler, the following algorithm is proposed to flatten hierarchical models. For every Model object in the file (modelDefinition and model):

1. Examine the submodels of the model you are translating or validating. If any of them themselves contain submodels, repeat this algorithm from the beginning with those submodels before continuing. Afterwards, repeat steps 2-5 for each submodel:
2. Give all submodels that do not yet have one an ID unique to the containing model's namespace. For example, if the model has three unnamed submodels, one might assign them the names 'submod1', 'submod2', and 'submod3', assuming no other element in the model had those names. Also ensure that no other element ID or metaID *begins* with "[modelID]_". If it does, add an underscore to the modelID and test again.
3. Remove any submodel elements that have been replaced or deleted.
4. Rename the remaining subelements. Every SId will become '[modelID]_[originalID]', and every metaID will become '[modelID]_[originalMetaID]'.
5. Rename all SIdRefs and metaIdRefs remaining in the submodel. Every IdRef that pointed to an element that was not removed, replace with '[modelID]_[originalID]'. Every IdRef that pointed to a replaced element, replace with the SId of the replacement element. This is the purpose of ensuring no element in the containing model began with the modelID and an underscore—if they had, we might overlap here, or at least be confusing. If any IdRefs are found that point to a deleted element, the model is invalid, as it contains a reference to a nonexistent element. Note that because containing models may reference sub-subelements directly by using the subelement-child-of-a-subelement construct, the SId or MetaId thus referenced may have been mangled by this algorithm. Be sure to be able to find these new sub-subelement IDs in some way, perhaps by keeping a map of old ID to new IDs. Rename in the order: Compartments → Species → Function Definitions → Rules → Events → Units → Reactions → Parameters.
6. After performing the above tasks for all submodels, assemble all remaining elements of the submodels and of the containing model into a single Model. Merge the various lists (ListOfSpecies, ListOfReactions, etc.), and be sure to include all remaining annotations. If

any other packages have defined their own ListOfX elements, ensure that these are merged properly as well.

7. Test the Model for validity.

At this point, the Model is ready to be exported as Level 3 core (plus any attendant packages used by the submodels), or to be declared valid or invalid by the validator. Again, when instantiating a model, one does not have to first test the validity of that model. If it is in the same file as the containing model, it will be tested anyway by this algorithm. If it is in a different file, that file's validity (or lack thereof) should not affect the validity of the file being tested (though a validator may warn the user of this situation if it desires).

7. Prototype Implementations

No implementations of this specific proposal yet exist. It is our intent to produce a working libSBML implementation next, and to incorporate that into the next version of Antimony within the year. However, Antimony and JigCell provide ways to modularize and combine existing SBML models in ways very similar to those proposed here, and the curious and anxious modeler is encouraged to experiment with those programs in anticipation of a libSBML implementation.

8. Translation to SBML Level 2

Once a hierarchical model is flattened (see the 'Validation' section), it becomes valid SBML Level 3, and, from there, it can be translated as well or as poorly as any other Level 3 model. Any other packages may be used in conjunction with this package, and they too should survive the flattening procedure to a non-hierarchical version of the model. The only issue is that since a file using this package may define multiple models, deciding which model to translate may be an issue. If a translation request is ambiguous, it is recommended that the translator produce a flattened version of the main 'model' child of the SBML object, instead of trying to produce flattened versions of every 'modelDefinition' object in the file. A robust translator should provide a mechanism to produce these models if requested, however.

9. Unresolved issues

We have addressed all known issues in the current proposal, but are still a bit concerned that the proposed conversion factors will be insufficient in some unforeseen way. Also, as currently structured, there is also no way to convert non-multiplicative conversions, such as the conversion of Celsius to Fahrenheit. We anticipate that this case will be rare enough that modelers will be able to find workarounds, such as replacing the odd parameter directly in the containing model, and then using an assignment rule for a new parameter that incorporates the appropriate conversion. It may be that this workaround will suffice for any unforeseen problems as well.

10. References

All references in this document (in particular, those in the 'History' section) are included either as part of the text itself, or as a footnote on the page on which it is found. Rules about SBML

classes that have child subclasses are assumed to apply to that subclass as well, unless explicitly stated (thus, rules about Models apply to ModelDefinitions; rules about SBaseRefs apply to Deletions, ReplacedElements, and Ports, etc.)

Appendix A: Validation rules

The following validation rules should all be stated or implied in the above specification, but are enumerated here for convenience. Unless explicitly stated, assume that all validation rules here are talking about elements and attributes defined in the 'comp' namespace.

General rules about this package

comp10101. An SBML file with elements conforming to this specification must use the namespace `"http://www.sbml.org/sbml/level3/version1/comp/version1"`

comp10102. All new classes and attributes described in this document must be defined using the `comp` namespace, either explicitly or via a prefix.

comp10103. The prefix used for this package's namespace must be `comp`.

General rules about identifiers

comp10201. Within an SBMLDocument, the value of the attribute `id` and `comp:id` on every instance of all Model, ModelDefinition, and ExternalModelDefinition objects must be unique across the set of all `id` and `comp:id` attribute values of such identifiers in the SBMLDocument to which they belong.

comp10202. (extending core 10301): Within a Model or ModelDefinition, the value of the attribute `id` and `comp:id` on every instance of the following classes of objects must be unique across the set of all id and `comp:id` attribute values of all such objects in a model: the Model itself, plus all contained FunctionDefinition, Compartment, Species, Reaction, SpeciesReference, ModifierSpeciesReference, Event, and Parameter objects, plus the newly-defined objects Submodel and Deletion.

comp10203. Within a Model or ModelDefinition, the value of the attribute `comp:id` on every instance of all Port objects must be unique across the set of all `comp:id` attribute values of all such objects in the model.

comp10204. The value of a `comp:id` attribute must always conform to the syntax of the SBML data type SId.

comp10205. The value of `model` attributes on ExternalModelDefinition objects, `submodelRef`, `deletion`, and `conversionFactor` attributes on ReplacedElement objects, `modelRef`, `lengthConversionFactor`, `areaConversionFactor`, `volumeConversionFactor`, `substanceConversionFactor`, `timeConversionFactor`, and `extentConversionFactor` attributes on Submodel objects, and `port` and `idRef` attributes on SBaseRef objects must always conform to the syntax of the SBML data type SId.

comp10206. The value of `unitRef` attributes on SBaseRef objects must always conform to the syntax of the SBML data type UnitSId.

comp10207. The value of `metaIdRef` attributes on SBaseRef objects must always conform to the syntax of the XML data type ID.

comp10208. The value of `source` attributes on ExternalModelDefinition objects must always conform to the syntax of the XML data type `anyURI`. (Reference: http://www.w3.org/TR/xmlschema-2/#anyURI)

comp10209. The value of `md5` attributes on ExternalModelDefinition objects must always conform to the syntax of type `string`.

General rules about SBaseRef objects and subclasses

comp10301. No Port object may use the optional `port` attribute, as this would cause either a circular reference, or would cause two port objects in the same model to point to the same element.

comp10302. No two Port objects in the same Model may reference the same XML element. That is, the element pointed to through the use of the `idRef`, `unitRef`, or `metaIdRef` attributes, in conjunction with any child SBaseRef element, may not be the same element pointed to by a Port object with the same parent ListOfPorts, whether it uses the same attribute to point to that object or not.

comp10303. No two ReplacedElement objects in the same Model may reference the same XML element unless that element is a Deletion. That is, the element pointed to through the use of the `port`, `idRef`, `unitRef`, or `metaIdRef` attributes, in conjunction with any child SBaseRef element, may not be the same element pointed to by any other ReplacedElement in the same Model, whether it uses the same attribute to point to that object or not.

General rules about circular references in models

comp10401. No ExternalModelDefinition may reference an ExternalModelDefinition in a different SBML document that in turn refers to the original ExternalModelDefinition object, whether directly or indirectly through a chain of ExternalModelDefinitions.

comp10402. No Model may contain a Submodel which references itself. That is, the `id` attribute of a Model may not match the `modelRef` attribute on any of its Submodels.

comp10403. No Model may contain a Submodel which references itself indirectly. That is, the `modelRef` attribute of a Submodel may not point to a Model, any of whose Submodels point to the original Model, whether directly or indirectly through a chain of Model/Submodel pairs.

Rules for the <sbml> container element

comp20101. There may be at most one instance of each of the following kind of object in an SBMLDocument: ListOfModelDefinitions, and ListOfExternalModelDefinitions.

comp20102. The `comp:required` attribute must be set `true` if its Model child contains any Submodels with Species, Parameters, Reactions, or Events (directly or indirectly) that have not been replaced. [Note: This may be too hard to implement—maybe go for a warning instead?]

Rules for the SBaseRef, Deletion, Port, and ReplacedElement components

comp20201. All SBaseRef objects must point to an object. That is, SBaseRef, Deletion, and Port objects must define one of the attributes `port`, `idRef`, `unitRef`, or `metaIdRef`, and ReplacedElement objects must define one of the attributes `port`, `idRef`, `unitRef`, `metaIdRef`, or `deletion`.

comp20202. No SBaseRef object may point to an object using more than one method. That is, SBaseRef, Deletion, and Port objects must not define more than one of the attributes `port`, `idRef`, `unitRef`, or `metaIdRef`, and ReplacedElement objects must not define more than one of the attributes `port`, `idRef`, `unitRef`, `metaIdRef`, or `deletion`.

comp20203. The value of a `port` attribute on an SBaseRef object must be the identifier of a Port object from the referenced Model.

comp20204. The value of an `idRef` attribute on an SBaseRef object must be the identifier of an object from the referenced Model within the SId namespace for that model. This includes elements with `id` attributes which are defined in packages other than Level 3 core or this `comp` package.

comp20205. The value of a `unitRef` attribute on an SBaseRef object must be the identifier of a UnitDefinition object from the referenced Model.

comp20206. The value of a `metaIdRef` attribute on an SBaseRef object must be the value of a `metaId` attribute on any element contained in the referenced Model. This includes elements with `metaId` attributes which are defined in packages other than Level 3 core or this `comp` package.

comp20207. If an SBaseRef object contains an SBaseRef child, it must point to a Submodel element.

comp20208. The value of a `submodelRef` attribute on a ReplacedElement object must be the identifier of a Submodel object from the parent Model of the ReplacedElement.

comp20209. The value of a `deletion` attribute on a ReplacedElement object must be the identifier of a Deletion object from the parent Model of the ReplacedElement.

comp20210. The value of a `submodelRef` attribute on a ReplacedElement object which also defines a `deletion` attribute must be the identifier of the Submodel object to which the referenced Deletion belongs.

comp20211. The value of a `conversionFactor` attribute on a ReplacedElement object must be the identifier of a Parameter object from the parent Model of the ReplacedElement.

comp20212. The value of an `identical` attribute on a ReplacedElement object must, if present, have a value of type `Boolean`.

comp20213. If the value of the `identical` attribute on a ReplacedElement object is `true`, the parent element of the ListOfReplacedElements to which the ReplacedElement belongs must be the same class as the referenced element.

comp20214. If the value of the `identical` attribute on a ReplacedElement object is `true`, the parent element of the ListOfReplacedElements to which the ReplacedElement belongs must define all of the attributes present on the referenced element. This

includes attributes from other namespaces, such as from packages other than Level 3 core and this 'comp' package.

comp20215. If the value of the `identical` attribute on a ReplacedElement object is `true`, the parent element of the ListOfReplacedElements to which the ReplacedElement belongs must only define attributes present on the referenced element, with the exception of the `id` and `metaId` attributes, which may be added even if not present on the referenced element.

comp20216. If the value of the `identical` attribute on a ReplacedElement object is `true`, all attributes of the parent element of the ListOfReplacedElements to which the ReplacedElement belongs (including attributes from other namespaces) must be identical to the corresponding attributes of the referenced element, with the exception of the `id` and `metaId` attributes, which may be anything, and with the exception of attributes of type SIdRef, UnitSIdRef, PortSIdRef, and IDREF, which must now reference elements of the parent model which themselves are replacements for the original target of the reference attribute. Those referenced replacements need not be flagged with '`identical=true`', and need not be identical to the elements they replace. If any attributes define a numerical value in the submodel that is converted to a new value in the parent model, that attribute must be set to be equal to the new numerical value.

comp20217. If the value of the `identical` attribute on a ReplacedElement object is `true`, the children of the parent element of the ListOfReplacedElements to which the ReplacedElement belongs must be identical to the corresponding children of the referenced element, with the exception of any child ListOfReplacedElements objects (which have no restrictions). 'Identical' means these child objects themselves must follow validation rules comp20213, comp20214, comp20215, comp20216, and comp20217.

comp20218. (warning) If the identical attribute on a ReplacedElement object is not set, all attributes with defined values on the referenced element should be defined on the parent element of the ListOfReplacedElements to which the ReplacedElement belongs.

comp20218. (warning) If the identical attribute on a ReplacedElement object is not set, the parent element of the ListOfReplacedElements to which the ReplacedElement belongs should contain the same number and type of children as the referenced element, with the exception of ListOfReplacedElements children.