SBML Level 3 Package Specification

Required Elements

Lucian P. Smith
lpsmith@u.washington.edu
Computing and Mathematical Sciences
California Institute of Technology
Seattle, Washington, US

Michael Hucka
mhucka@caltech.edu
Computing and Mathematical Sciences
California Institute of Technology
Pasadena, California, US

Version 1 Release 0.3 (Draft)

09 May 2013

The latest release, past releases, and other materials related to this specification are available at http://sbml.org/Documents/Specifications/SBML_Level_3/Packages/req

This release of the specification is available at
TBD

Contents

1 Introduction

The SBML Level 3 Required Elements package is a small package that allows models to declare exactly which components of a model have the meaning of their mathematical semantics changed by another package. It does this simply by defining optional children of ***SBase*** which can be attached to any component of a model with associated mathematics. The rest of this specification explains the package constructs and how they can be used.

1.1 Proposal corresponding to this package specification

This specification for Required Elements in SBML Level 3 Version 1 is based on the draft located at this URL:

`https://sbml.svn.sf.net/svnroot/sbml/trunk/specifications/sbml-level-3/version-1/req/specification`

The tracking number in the SBML issue tracking system (SBML Team, 2010) for Required Elements package activities is 3594695. The version of the proposal used as the starting point for this specification is the version of August, 2011 (Smith, 2011).

1.2 Package dependencies

The Required Elements package has no dependencies on other SBML Level 3 packages, though, to be useful, some other required package must be present in the document.

1.3 Document conventions

UML 1.0 (Unified Modeling Language; Eriksson and Penker 1998; Oestereich 1999) notation is used in this document to define the constructs provided by this package. Colors in the diagrams carry the following additional information for the benefit of those viewing the document on media that can display color:

- *Black*: Items colored black in the UML diagrams are components taken unchanged from their definition in the SBML Level 3 Core specification document.
- *Green*: Items colored green are components that exist in SBML Level 3 Core, but are extended by this package. Class boxes are also drawn with dashed lines to further distinguish them.
- *Blue*: Items colored blue are new components introduced in this package specification. They have no equivalent in the SBML Level 3 Core specification.

The following typographical conventions distinguish the names of objects and data types from other entities; these conventions are identical to the conventions used in the SBML Level 3 Core specification document:

Class: Names of ordinary (concrete) classes begin with a capital letter and are printed in an upright, bold, sans-serif typeface. In electronic document formats, the class names are also hyperlinked to their definitions in this specification document. (However, class names are not hyperlinked if they are for classes that are unchanged from their definitions in their original specification.)

`SomeThing`, `otherThing`: Attributes of classes, data type names, literal XML, and tokens *other* than SBML class names, are printed in an upright typewriter typeface.

For other matters involving the use of UML and XML, this document follows the conventions used in the SBML Level 3 Core specification document.

2 Background and context

At the heart of many SBML models is mathematics. When present, the mathematics is considered to be of principle importance to understanding the model. Many SBML Level 3 packages replace or extend the mathematical concepts present in SBML Level 3 Version 1 Core, and when they do, a model that includes these concepts must add a `required`="`true`" flag on the definition of that namespace. However, for software tools that wish to manipulate and augment SBML models with extended mathematics which they do not understand, a blanket `required`="`true`" flag on the namespace is not granular enough. The tools need to know which *specific* model elements have had their mathematics changed, so that they can understand which parts of the model are safe to manipulate freely and which must be treated with care.

The Required Elements package is an exceedingly small package that allows model writers to declare specifically which components of the model have had their mathematics changed, by which package, and whether interpretating of those components in the absence of any package information results in a workable model. It accomplishes this by defining a class that can be added as an optional child of any SBML element with mathematical meaning.

3 Package syntax and semantics

This section contains a definition of the syntax and semantics of the Required Element package for SBML Level 3 Version 1 Core.

3.1 Namespace URI and other declarations necessary for using this package

Every SBML Level 3 package is identified uniquely by an XML namespace URI. For an SBML document to be able to use a given Level 3 package, it must declare the use of that package by referencing its URI. The following is the namespace URI for this version of the Required Elements package for SBML Level 3 Version 1 Core:

"`http://www.sbml.org/sbml/level3/version1/req/version1`"

In addition, SBML documents using a given package must indicate whether understanding the package is required for complete mathematical interpretation of a model. This is done using the attribute `required` on the `<sbml>` element in the SBML document. For the Required Elements package, the value of this attribute must be "`false`", because while the Required Elements package is designed to clarify how other packages change the mathematical meaning of the model, it cannot itself do so.

The following fragment illustrates the beginning of a typical SBML model using SBML Level 3 Version 1 Core and this version of the Required Elements package:

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

3.2 Primitive data types

The Required Elements package uses three of the data types described in Section 3.1 of the SBML Level 3 Version 1 Core specification, specifically `SId`, `string` and `boolean`. The package does not define any new types.

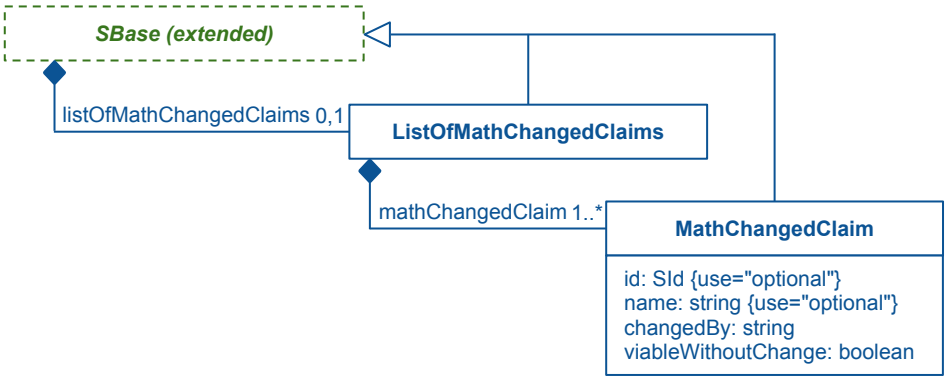

Figure 1: *The definition of the extended* ***SBase*** *class, and the new* ***ListOfMathChangedClaims*** *and* ***MathChangedClaim*** *classes.*

3.3 The extended SBase class

As mentioned above, this package's reason for being is to define an optional child of the SBML Level 3 ***SBase*** class. Figure 1 provides the definition of this ***SBase*** extension, and its optional **MathChangedClaim** children. These children are added to ***SBase*** instead of to those elements that have math in them (such as **InitialAssignment**,

KineticLaw, etc.) or that have mathematical meaning (such as **Parameter** or **Species**) because this approach allows other SBML Level 3 packages to define new elements with mathematics which may, in turn, be overridden by other packages.

3.4 The ListOfMathChangedClaims class

As is standard in SBML specifications, any time more than one child of the same class is to be added to an element, a 'ListOf' that element is provided to store them, and to collect annotations about the list itself. The child 'listOfMathChangedClaims' on SBase is optional, but if present, must contain one or more 'mathChangedClaim' children.

3.5 The MathChangedClaim class

The **MathChangedClaim** class defines an element, one or more of which may be added to any SBML element that either has mathematical meaning, or that contains a **Math** child. In SBML Level 3 Core, the former group includes the classes **Compartment**, **Parameter**, **Reaction**, **Species**, and **SpeciesReference**, and the latter group includes the classes **Constraint**, **Delay**, **EventAssignment**, **FunctionDefinition**, **InitialAssignment**, **KineticLaw**, **Priority**, **Rule**, and **Trigger**. **Event** elements do not have any direct **Math** children, but do indirectly, and may thus be given a **MathChangedClaim** child. Packages that introduce elements that have mathematical meaning and/or that contain **Math** children may also be extended through the addition of a **MathChangedClaim** child, when some other package changes that element's mathematical interpretation.

Elements with mathematical meaning may have a **MathChangedClaim** child when a package alters the value or meaning of that symbol. As an example, a **Submodel** from the Hierarchical Model Composition package may contain an **Event** or **Rule** that assigns new values to a parameter. Because an interpreter that did not understand submodels would not catch this change, the Hierarchical Model Composition package can be seen to change the math of that element, and it would be appropriate to denote this by adding a **MathChangedClaim** child to the affected parameter.

Similarly, models that use the proposed Spatial Processes package can change the meaning of a **Compartment** by turning it into a bounded object with an size implied by those boundaries (and how they change over time), instead of using the element's `size` attribute. Spatial Processes elements may also change a **Species** to be spatially defined, and therefore represent different values depending on what coordinates in space are under consideration. Affected compartments and species could be given **MathChangedClaim** children to denote this fact.

Elements with **Math** children may also be changed by the addition of package elements. Some packages may instruct the modeler to disregard the **Math** and to use some other construct instead. For example, the proposed Distrubutions package adds a new child to **FunctionDefinition**, which replaces the old mathematics with a new set of mathematics returning a draw from a random distribution (something impossible with **Math**).

Each package may define for itself when it would be appropriate or necessary, given the package's structure, to add a **MathChangedClaim** element to something. If no such definition exists, a user may add **MathChangedClaim** elements where they understand the math to be changing.

For packages with no such rules, the addition of **MathChangedClaim** children cannot be guaranteed to be normative, that is, software that understands the Required Elements package, but which does not understand some other package flagged `required`="`true`" will not be able to assert that every element with altered mathematics has been given a **MathChangedClaim** child. However, a package may define its own validation rules in conjunction with this Required Elements package, and a user may then be able to guarantee that any affected element has been denoted as such.

Note that a **MathChangedClaim** element should be added to the model component that is directly affected only, and not on a component that is indirectly affected by the changed mathematics. For example, if a model contains a **FunctionDefinition** object whose mathematics are extended by the SBML Level 3 package for Distributions, and elsewhere in the model there is an **InitialAssignment** object that uses that function, a **MathChangedClaim** child should be added only to the **FunctionDefinition** object and not the **InitialAssignment** nor on the element that assigns

to, because the latter are only indirectly affected, through processes understood without reference to package definitions.

Multiple packages may affect the mathematics of the same object in different ways. For example, a construct from the Spatial Process package could change a **Species** to be spatially defined, and that same model may have a **Submodel** from the Hierarchical Model Composition package which contains a **Reaction** that affects that **Species**. In that case, the **Species** would have two **MathChangedClaim** children, one for each package that affects it.

Since **MathChangedClaim** itself has no mathematical meaning and no child **Math** object, it may not itself have a **MathChangedClaim** child.

3.5.1 *The* `id` *and* `name` *attributes*

The optional `id` attribute on the **MathChangedClaim** object class serves to provide a way to identify the element. The attribute takes a value of type `SId`. Note that this identifier carries no mathematical interpretation and cannot be used in mathematical formulas in a model. **MathChangedClaim** also has an optional `name` attribute, of type `string`. The `name` attribute may be used in the same manner as other `name` attributes on SBML Level 3 Version 1 Core objects; please see Section 3.3.2 of the SBML Level 3 Version 1 Core specification for more information.

3.5.2 *The* `changedBy` *attribute*

The `changedBy` attribute is required, and is of type `string`. The attribute must be set to the namespace URI of the SBML Level 3 package that redefines or alters the mathematical semantics of the parent object. In other words, if the mathematical semantics of a given component C in a model is changed by the use of a Level 3 package P, then C can be given a **MathChangedClaim** child, and its attribute `changedBy` should be P's namespace URI. The examples below will make this more clear.

Valid values for the `changedBy` token include the namespace URI for any SBML L3 package flagged `required` ="`true`". If there are no such packages present in the SBML Document, there is no valid value for the `changedBy` attribute, and therefore no **MathChangedClaim** element may appear in the document.

No two child **MathChangedClaim** elements of any single ***SBase*** object may have the same `changedBy` value.

3.5.3 *The* `viableWithoutChange` *attribute*

The `viableWithoutChange` attribute is required, and is of type `boolean`. The attribute should be used to indicate whether an interpreter that does not understand the package from the `changedBy` attribute would have an interpretable version of the mathematics for the given component in a model. This can be established using two criteria: "Complete" and "Workable". Obviously, if the math being interpreted is incomplete, the `viableWithoutChange` attribute must be "`false`". If the math is complete, then whether the solution is "workable" requires a judgement call on the part of the modeler: if the modeler feels that the alternative version makes sense in an alternative context, they may set the attribute value to "`true`"; conversely, if they feel that the resulting model component makes no sense, even if technically "complete", then they should set the attribute value to "`false`".

When two different packages alter the mathematical interpretation of an ***SBase*** object in different ways, and two corresponding **MathChangedClaim** elements are added, each should consider whether the aspect of the change that package is responsible for has an alternative, irrespective of how the other package affects the element. In our above example where both the Spatial Processes and the Hierarchical Model Composition package affected the same **Species**, the **MathChangedClaim** element for the Spatial Processes package may have its `viableWithoutChange` attribute set to "`true`" if an alternate framework without Spatial Processes constructs is present (even if spread across multiple **Submodel** elements from the Hierarchical Model Composition package). Conversely, if the **Reaction** that affects the **Species** is in a **Submodel**, the corresponding `viableWithoutChange` for that package should be set to "`false`" if that **Reaction** is considered vital to the understanding and interpertation of the model.

3.6 Additional considerations

The Required Elements package is not, itself, required (that is, models that use it must set `required`="`false`" for its namespace when using it) because it, in and of itself, does not change any mathematical semantics. However, it is intended to be used by other required packages, which may in turn make the use of this package in certain contexts defined in those package specifications.

4 Examples

This section contains examples employing the Required Elements package for SBML Level 3.

4.1 Function whose interpretation is reasonable in the absence of a package

In this example, there is a **FunctionDefinition** object that has had its mathematics changed by an (as-yet hypothetical) Distributions package.

```
<listOfFunctionDefinitions>
  <functionDefinition id="unitGaussian">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <lambda>
        <bvar>
          <ci> mean </ci>
        </bvar>
        <bvar>
          <ci> variance </ci>
        </bvar>
        <ci> mean </ci>
      </lambda>
    </math>
    <distrib:drawFromDistribution>
      <distrib:input> mean </distrib:input>
      <distrib:input> var </distrib:input>
      <distrib:predefinedPDF xmlns:un="http://uncertml.org/3.0">
        <un:NormalDistribution>
           <un:mean>mean</un:mean>
           <un:variance>var</un:variance>
         </un:NormalDistribution>
      </distrib:predefinedPDF>
    </distrib:drawFromDistribution>
    <req:listOfMathChangedClaims>
      <req:mathChangedClaim changedBy="http://www.sbml.org/sbml/level3/version1/distrib/version1"
                            viableWithoutChange="true" />
    </req:listOfMathChangedClaims>
  </functionDefinition>
</listOfFunctionDefinitions>
```

The function `unitGaussian` above has both a straight SBML Level 3 Core definition (whose MathML formula has the effect of returning "`mean`", unchanged), and a package-defined version (in effect, "select a random number from a Gaussian distribution with mean "`mean`" and variance "`var`""). The modeler has elected to claim that the SBML Core definition's version of the function is adequate for the model in which it is being used, although different results are to be expected.

4.2 Function whose interpretation is *not* reasonable in the absence of a package

In this example, the modeler has decided that the plain SBML Level 3 Core version of the mathematics will not make an adequate substitution for using the mathematics defined by the Distributions package, as it returns '0' instead of something more reasonable:

```
<listOfFunctionDefinitions>
  <functionDefinition id="unitGaussian">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <lambda>
        <bvar>
          <ci> mean </ci>
```

```
        </bvar>
        <bvar>
          <ci> var </ci>
        </bvar>
        <cn> 0 </cn>
      </lambda>
    </math>
    <distrib:drawFromDistribution>
      <distrib:input> mean </distrib:input>
      <distrib:input> var </distrib:input>
      <distrib:predefinedPDF xmlns:un="http://uncertml.org/3.0">
         <un:NormalDistribution>
           <un:mean>mean</un:mean>
           <un:variance>var</un:variance>
         </un:NormalDistribution>
      </distrib:predefinedPDF>
    </distrib:drawFromDistribution>
    <req:listOfMathChangedClaims>
      <req:mathChangedClaim changedBy="http://www.sbml.org/sbml/level3/version1/distrib/version1"
                            viableWithoutChange="false" />
    </req:listOfMathChangedClaims>
  </functionDefinition>
</listOfFunctionDefinitions>
```

4.3 Function whose definition is incomplete

In the following example, the function definition is marked as having `viableWithoutChange`="`false`" because the lambda function is actually incomplete, and returns nothing at all.

```
<listOfFunctionDefinitions>
  <functionDefinition id="unitGaussian">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <lambda>
        <bvar>
          <ci> mean </ci>
        </bvar>
        <bvar>
          <ci> variance </ci>
        </bvar>
      </lambda>
    </math>
    <distrib:drawFromDistribution>
      <distrib:input> mean </distrib:input>
      <distrib:input> var </distrib:input>
      <distrib:predefinedPDF xmlns:un="http://uncertml.org/3.0">
         <un:NormalDistribution>
           <un:mean>mean</un:mean>
           <un:variance>var</un:variance>
         </un:NormalDistribution>
      </distrib:predefinedPDF>
    </distrib:drawFromDistribution>
    <req:listOfMathChangedClaims>
      <req:mathChangedClaim changedBy="http://www.sbml.org/sbml/level3/version1/distrib/version1"
                            viableWithoutChange="false" />
    </req:listOfMathChangedClaims>
  </functionDefinition>
</listOfFunctionDefinitions>
```

5 Relationship with other packages

Each package that can be set `required`="`true`" has the potential to interact with the Required Elements package. For packages that have already been finalized, we present validation rules that, if followed, would allow a software package that did not understand the package in question to know what math was changed from what would have been expected, had those package constructs not been present. For packages that have not yet been finalized, hypothetical validation rules are proposed based on the current version of the specification, but to fully ensure that all the appropriate **MathChangedClaim** elements were added to a document with that package, a full set of validation rules that matched the final package specification would have to be produced, either as a part of that other package's specification, or as part of a future version of this specification.

It is important to note that a package can be set `required`="`true`" for two reasons: One, if the existing mathematical interpretation of the model has been changed, and two, if *new* mathematics are introduced. The Required Elements package is only concerned with the former. For that reason, a document may have a required package present that nevertheless will not have any **MathChangedClaim** elements.

Determining whether a model follows the given validation rules will require understanding of that package, which presents a sort of catch-22 for any software tool: the only reason to use Required Elements constructs in the first place is to mitigate situations where a package is *not* understood. We propose that any software in this situation use an external validator that understands all packages in a technical sense, but has no need to actually interpret the models mathematically, to ensure that a model is valid, and then proceed from there. If the model is valid, all mathematically-modified elements will be guaranteed to have **MathChangedClaim** children.

It is important to note that all the validation rules presented only take effect if the Required Elements package namespace is present in an SBML document. In effect, by adding the package to your model, you agree to abide by the validation rules herein. However, it is perfectly possible to use the packages without 'Required Elements'; you then simply revert to the standard global required/not required package information, which may be perfectly acceptable.

5.1 The Hierarchical Model Composition package

The URI "`http://www.sbml.org/sbml/level3/version1/comp/version1`" is used to refer to the Hierarchical Model Composition package. This must be the string used for any `changedBy` attribute that refers to this package.

The Hierarchical Model Composition package allows modelers to compose models hierarchically, by adding a list of **Model** objects to the document, allowing them to be imported into other models via the **Submodel** construct, and then connecting elements of those models together with **ReplacedElement** and **ReplacedBy** constructs.

It is only the last of those three abilities that can directly change the mathematical interpretation of an element of the main **Model** object in an SBML Document. Defining a new **Model** will not change the interpretation of anything in the original **Model**, nor will incorporating submodels into the original **Model**.

All elements with mathematical meaning which have been given a **ReplacedBy** child have had their meaning changed, by definition. Therefore, any **Compartment**, **Parameter**, **Reaction**, **Species**, **Constraint**, **Event**, **FunctionDefinition**, **InitialAssignment**, or **Rule** with a **ReplacedBy** child must be given a **MathChangedClaim** child. (Other elements with mathematical meaning exist in SBML Level 3 Core, but are not on this list because they may not have **ReplacedBy** children.) The value of the `viableWithoutChange` may be set to be "`true`" if the original element is completely defined (or if it is *at least as* defined as its replacement), and must be set "`false`" otherwise. The attribute may also be set "`false`" if the original element differs from its replacement, and/or if their mathematical interpretation is changed by some other element in the replacement (such as by a **Rule** or **Reaction**; see below).

Compartment, **Parameter**, **Reaction**, **Species**, and **SpeciesReference** elements all have the ability to have their mathematical interpretation changed by **InitialAssignment**, **Rule**, and **Event** constructs. If any of these three elements are present in a **Submodel**, and their target is an element in the main model (because their 'local' target has been replaced by the main model element), that element in the main model must be given a **MathChangedClaim** child

referencing this package. The value of the `viableWithoutChange` may be set to be "`true`" if the modeler determines that the absence of the **InitialAssignment**, **Rule**, or **Event** creates an alternate model scenario that is nonetheless workable.

Any element that does not fall under the above two situations will not have its math changed, and must therefore not have a **MathChangedClaim** child targeting this package.

Modelers that wish to use **MathChangedClaim** elements in **Model** elements that are *not* the direct child of the `sbml` element may do so, but this only facilitates the translation of those tertiary models to their own SBML Documents: any software that could read and understand a tertiary model must, by definition, be able to understand the Hierarchical Model Composition package, and therefore has no need of any **MathChangedClaim** element that targets this package.

5.2 The Flux Balance Constraints package

Currently, documents that use the Flux Balance Constraints package must set its `required` attribute to "`false`", making it an illegal target of the Required Elements Package. However, the SBML editors recently revised their interpretation of the `required` attribute such that future versions of the package may allow (or even require) that it be set to "`true`". However, even if this occurs, the function of this package is solely to define a new set of mathematical constructs, along with their interpretation: no element not in the package is affected at all. Therefore, there is no need to create any **MathChangedClaim** elements that target this package: any such elements are illegal.

5.3 The Qualitative Models package

In Version 1 of the Qualitative Models package, all of the added constructs create a new mathematical system of **QualitativeSpecies** and **Transitions**, which can be affected by core constructs, but which cannot affect core constructs. Therefore, there is no need to create any **MathChangedClaim** elements that target this package: any such elements are illegal.

5.4 The Distributions package

The Distrubtions package has not yet been finalized, but in its current (draft) form, it is used to annotate SBML elements with information about the error or uncertainty in their values, which does not affect the mathematics of the model, and to modify **FunctionDefinition** elements to produce draws from distributions, instead of producing deterministic outcomes. Any **FunctionDefinition** so modified must be given a **MathChangedClaim** referring to this package; the `viableWithoutChange` element must be set "`false`" if the original **Math** child is incomplete, and may be set "`true`" if the deterministic value it produces results in a workable, though non-random, model. No other element may be given a **MathChangedClaim** child that targets this package.

5.5 The Multistate and Multicomponent Species package

The Multistate and Multicomponent Species package has not yet been finalized, but in its current (draft) form, it allows the redefinition of **Species** and **Compartment** elements as types which might be instantiated in a number of ways. Any such 'prototype'-style elements would need **MathChangedClaim** children. Furthermore, as currently written, no such **Species** or **Compartment** may be instantiated or changed, whether by attribute, **InitialAssignment**, or **Rule**, and the value of the `viableWithoutChange` attribute would always be required to be set "`false`".

5.6 The Spatial Processes package

The Spatial Processes package has not yet been finalized, but in its current (draft) form, it modifies **Species**, **Compartment**, and **Reaction** elements such that they can be modeled spatially, with partial differential equations. All such modified elements must therefore be given **MathChangedClaim** children–the specification itself details how this is to be done, and what values for `viableWithoutChange` are approprite.

5.7 The Arrays package

The Arrays package has not yet been finalized, but in its current (draft) form, it has the ability to modify any element with mathematical meaning to represent an array instead of a single value. All such modified elements must therefore be given **MathChangedClaim** children, with more details to follow as the package is developed.

5.8 The Dynamic Structures package

The Dynamic Structurs package has not yet been finalized, and is not, as of this writing, undergoing active development. Any structures it sets in place to create or destroy elements with mathematical meaning would clearly affect the mathematics of the model, and any elements so affected would be appropriate targets for **MathChangedClaim** children.

5.9 Non-required packages

The Annotations, Layout, Render, Groups, and this Required Elements package all explicitly do not affect the mathematics of a model, and must be set `required`="`false`". No **MathChangedClaim** element may therefore reference these packages.

5.10 Future packages

It is hoped that, should this Required Elements package prove useful, packages defined in the future will explicitly mention what elements it defines that must be given **MathChangedClaim** children, and what values to give its attributes.

A Validation of SBML documents using Required Elements constructs

This section summarizes all the conditions that must (or in some cases, at least *should*) be true of an SBML Level 3 Version 1 model that uses the Required Elements package. We use the same conventions that are used in the SBML Level 3 Version 1 Core specification document. In particular, there are different degrees of rule strictness. Formally, the differences are expressed in the statement of a rule: either a rule states that a condition *must* be true, or a rule states that it *should* be true. Rules of the former kind are strict SBML validation rules—a model encoded in SBML must conform to all of them in order to be considered valid. Rules of the latter kind are consistency rules. To help highlight these differences, we use the following three symbols next to the rule numbers:

☑ A checked box indicates a *requirement* for SBML conformance. If a model does not follow this rule, it does not conform to the Required Elements package specification. (Mnemonic intention behind the choice of symbol: "This must be checked.")

▲ A triangle indicates a *recommendation* for model consistency. If a model does not follow this rule, it is not considered strictly invalid as far as the Required Elements package specification is concerned; however, it indicates that the model contains a physical or conceptual inconsistency. (Mnemonic intention behind the choice of symbol: "This is a cause for warning.")

★ A star indicates a strong recommendation for good modeling practice. This rule is not strictly a matter of SBML encoding, but the recommendation comes from logical reasoning. As in the previous case, if a model does not follow this rule, it is not considered an invalid SBML encoding. (Mnemonic intention behind the choice of symbol: "You're a star if you heed this.")

The validation rules listed in the following subsections are all stated or implied in the rest of this specification document. They are enumerated here for convenience. Unless explicitly stated, all validation rules concern objects and attributes specifically defined in the Required Elements package.

☞ For convenience and brievity, we use the shorthand "`req:x`" to stand for an attribute or element name `x` in the namespace for the Required Elements package, using the namespace prefix `req`. In reality, the prefix string may be different from the literal "`req`" used here (and indeed, it can be any valid XML namespace prefix that the modeler or software chooses). We use "`req:x`" because it is shorter than to write a full explanation everywhere we refer to an attribute or element in the Required Elements package namespace.

General rules about the Required Elements package

req-10101 ☑ To conform to Version 1 of the Required Elements package specification for SBML Level 3, an SBML document must declare the use of the following XML Namespace: "`http://www.sbml.org/sbml/level3/version1/req/version1`". (References: SBML Level 3 Package Specification for Required Elements, Version 1, Section 3.1 on page 5.)

req-10102 ☑ Wherever they appear in an SBML document, elements and attributes from the Required Elements package must be declared either implicitly or explicitly to be in the XML namespace "`http://www.sbml.org/sbml/level3/version1/req/version1`". (References: SBML Level 3 Package Specification for Required Elements, Version 1, Section 3.1 on page 5.)

req-10103 ☑ The value of attribute `req:required` on the **SBML** object must be set to "`false`". (References: SBML Level 3 Package Specification for Required Elements, Version 1, Section 3.1 on page 5.)

Rules for Required Elements attributes

req-20101 ☑ The attribute `mathOverridden`, if present, must have a value of type `string`. (References: SBML Level 3 Package Specification for Required Elements, Version 1, Section 3.3 on page 5.)

req-20102 ☑ The attribute `mathOverridden`, if present, must be the label of an SBML Level 3 package present in the same document. (References: SBML Level 3 Package Specification for Required Elements, Version 1, Section 3.3 on page 5.)

req-20103 ☑ The attribute `coreHasAlternativeMath`, if present, must have a value of type `boolean`. (References: SBML Level 3 Package Specification for Required Elements, Version 1, Section 3.3 on page 5.)

req-20104 ☑ If the value of the attribute `coreHasAlternativeMath` is set `true`, any `math` child of the parent element must be present and complete. (References: SBML Level 3 Package Specification for Required Elements, Version 1, Section 3.3 on page 5.)

Acknowledgments

Work such as this does not take place in a vacuum; many people contributed ideas and discussions that shaped the Required Elements proposal that you see before you. We particularly thank Sarah Keating, Frank Bergmann, James Schaff, and the members of the *sbml-discuss* mailing list for suggestions and comments.

This work was funded by the National Institutes of Health (USA) under grant R01 GM070923.

References

Eriksson, H.-E. and Penker, M. (1998). *UML Toolkit*. John Wiley & Sons, New York.

Oestereich, B. (1999). *Developing Software with UML: Object-Oriented Analysis and Design in Practice*. Addison-Wesley.

SBML Team (2010). The SBML issue tracker. Available via the World Wide Web at http://sbml.org/issue-tracker.

Smith, L. P. (2011). Required Elements Proposal. Available via the World Wide Web at http://sbml.org/Community/Wiki/SBML_Level_3_Proposals/Required_Elements_Proposal.