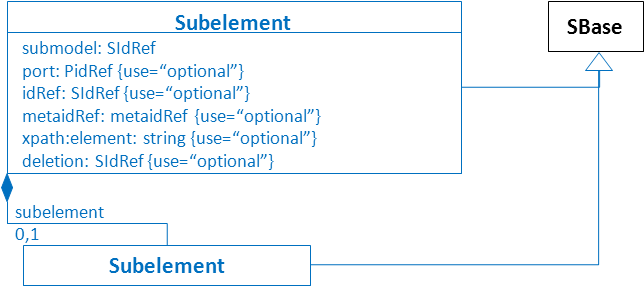
Proposed Syntax and Semantics

Part II: Replacements, Deletions, and Ports

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 this 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 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 a ‘Subelement’.



**Figure 1**: Definition of Subelement. A Subelement must contain the submodel attribute, and must contain exactly one of the attributes ‘port’, ‘symbol, ‘symbolMetaId’, ‘xpath:element’, or ‘deletion’, or else contain a subelement child.

A subelement object references an element of a submodel by first pointing to that submodel (with the ‘submodel’ attribute pointing to its SId), and then 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 [section Ports], but if the element has a port ID, it can be used directly here. A fully modular model will only use ports, which are the defined interfaces between models and submodels.
* **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.
* **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 xpath:element**. If all else fails, you can reference a subelement by the W3C standard for addressing parts of an XML document, XPath (<http://www.w3.org/TR/xpath> ). The XPath used should be relative to the ‘model’ object referred to by the ModelRef object in the corresponding Submodel object.
* **By subelement**. The above four options will all give you access to elements in the referenced submodel, but cannot give you access to elements in the submodel’s submodels. Adding a Subelement child to a Subelement with only a ‘submodel’ ID allows you to find elements ‘buried’ in the hierarchy. The namespace of the child Subelement is the namespace of the submodel referred to in the submodel ID of the parent Subelement. 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.

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

[[Optional validation rules: The order of the above list could be enforced by the validator, insisting that if a referenced object has a portid that it be used, and if not, its SId, etc. Alternatively, we could issue warnings if a metaID was used when a portid was available, etc.]]

Finally, it is sometimes useful to refer to an element of the submodel that has been deleted. This will be discussed in the [whatever] section, but for now, suffice to say that this is the construct needed to refer to the deleted element. The SidRef namespace for the ‘deletion’ attribute is the namespace of the \*parent\* model, not the submodel, as the (optional) SId in question is the one on the Deletion element. It is illegal to reference an already-deleted element of a submodel other than with this construct.

We have already seen the Deletion element in the Submodel discussion, which are a subclass of Subelement, extended to have an optional SId. As we now know, these ids are there so that they can be referenced by other Subelements, elsewhere in the model. Any reference to a deleted element, whether in the submodel or in the parent model, and whether to the element’s SId or to its MetaID, will produce a validation error. The model must be additionally modified to remove any such references.

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.

Replacements and Ports

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. 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 links. 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, and another list of all ports that this model would provide for those models that imported it.

Here, the concepts of replacements and port-ness are distributed to the individual elements that are replacing others, and/or which are intended to be ports for containing models. This is accomplished by extending the SBase class itself:

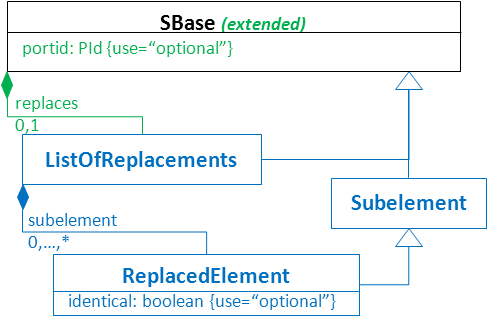


Figure : Definition of the SBase extension and the new ListOfReplacements, and ReplacedElement classes. SBase is here extended to have a single new optional attribute (‘portid’), and an optional child, ListOfReplacements, which may contain any number of ReplacedElement children. The ReplacedElement class inherits from Subelement, with the additional optional Boolean attribute of ‘identical’.

The portid referenced type, PId, has the same string restrictions as an SId (see section 3.1.7 of the Level 3 Version 1 SBML core specification), but has its own namespace within its parent model, and thus may be the same as the element’s SId, should one be present. It makes creating ports extremely simple (simply define the ‘portid’ of the element), and even desirable in those cases where one must link an element with no SId, encouraging port use. By extending SBase, it allows even elements defined in other SBML packages to immediately develop hierarchical models using their new constructs, as nearly everything [? Or absolutely everything?] in SBML inherits from SBase.

As written, this scheme does not have the capability to tag a port with a Boolean flag saying whether it must be overloaded or not (as Andrew Finney wanted in 2007 (<http://sbml.org/Andrew_2007_Comments_about_Model_Composition>)), but if the community decided such a flag was important, it, too, could be added as a flag to HierarchicalSBase. Since a design goal of Level 3 is to have no defaults, we would then have to decide what was meant if that flag was ‘true’, ‘false’, or undeclared. (As a possibility: issue a validation error if the flag was set ‘true’ but the element was not replaced; issue a validation warning if the flag was undeclared and the element not replaced; and issue none of these warnings or errors if the flag was set ‘false’.)

In the same way that ports look outward to parent models that contain them, replacements look inward to find submodel elements that are being replaced. The ListOfReplacements child of HierarchicalSBase lists everything in all the submodels that this element is replacing. This, again, 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; 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.

We note here that the conversion attributes on the Submodel, if present, will be crucial when aggregating and synchronizing the math in the combined model. Care should be taken that individual replacements like this are appropriately converted according to the Submodel conversion attributes. In the Hoops 2007 proposal, conversion factors were included on individual Replacement elements like this—we believe that this was redundant with the conversion factors present on the submodel itself, but it will take implementation of hierarchical mixed-unit model simulators before the issue can be considered completely resolved. It will be particularly tricky to include a model that was written in units of amount within a model written in units of concentration, and vice versa. [Further bulletins as events warrant.]

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. The sole exceptions are the ‘id’, ‘metaid’, and ‘portid’ attributes, which each exist in different namespaces (and may be required to be different, in the metaid case). 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. The most likely use case for this is in an ‘N to M’ replacement discussed by Andrew Finney in the [History] section: 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 nor 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.