

Spatial Processes

James C. Schaff

schaff@neuron.uchc.edu

Center for Cell Analysis and Modeling

University of Connecticut

Farmington, Connecticut, US

Anu Lakshminarayana

anu@uchc.edu

Center for Cell Analysis and Modeling

University of Connecticut

Farmington, Connecticut, US

Lucian P. Smith

lpsmith@u.washington.edu

Computing and Mathematical Sciences

California Institute of Technology

Seattle, Washington, US

Version 1, Release 0.85 (Draft)

13 December 2013

This is a draft specification for the SBML Level 3 package called “**spatial**”. It is not a normative document. Please send feedback to the Package Working Group mailing list at sbml-spatial@lists.sourceforge.net.

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

This release of the specification is available at

[TBD](#)



Contents

1	Introduction	5
1.1	Proposal corresponding to this package specification	5
1.2	Package dependencies	5
1.3	Document conventions	5
2	Background and context	7
2.1	Problems with current SBML approaches	7
2.2	Past work on this problem or similar topics	7
2.3	Prior work	7
3	Package syntax and semantics	8
3.1	Overview of spatial extension	8
3.1.1	Geometry	8
3.1.2	Alternative <i>GeometryDefinitions</i>	8
3.2	Namespace URI and other declarations necessary for using this package	8
3.3	Primitive data types	9
3.3.1	Type <i>SpId</i>	9
3.3.2	Type <i>SpIdRef</i>	9
3.3.3	Type <i>BoundaryConditionKind</i>	9
3.3.4	Type <i>ComponentKind</i>	9
3.3.5	Type <i>CoordinateKind</i>	10
3.3.6	Type <i>DiffusionKind</i>	10
3.3.7	Type <i>FunctionKind</i>	10
3.3.8	Type <i>GeometryKind</i>	10
3.3.9	Type <i>SetOperation</i>	10
3.3.10	Type <i>doubleArray</i>	10
3.3.11	Type <i>integerArray</i>	10
3.4	The extended <i>Model</i> object	11
3.5	The extended <i>Compartment</i> object	11
3.6	The <i>CompartmentMapping</i> class	12
3.6.1	The <i>spatialId</i> attribute	12
3.6.2	The <i>domainType</i> attribute	12
3.6.3	The <i>unitSize</i> attribute	12
3.7	The extended <i>Species</i> object	12
3.7.1	The <i>isSpatial</i> attribute	13
3.8	The extended <i>Parameter</i> object	13
3.9	The <i>SpatialSymbolReference</i> class	13
3.9.1	The <i>spatialRef</i> attribute	14
3.9.2	The <i>type</i> attribute	14
3.10	The <i>DiffusionCoefficient</i> class	14
3.10.1	The <i>variable</i> attribute	15
3.10.2	The <i>type</i> attribute	15
3.10.3	The <i>CoordinateIndex</i> class	15
3.10.4	<i>DiffusionCoefficient</i> uniqueness	15
3.11	The <i>AdvectionCoefficient</i> class	15
3.11.1	The <i>variable</i> attribute	15
3.11.2	The <i>coordinateIndex</i> attribute	16
3.12	The <i>BoundaryCondition</i> class	16
3.12.1	The <i>variable</i> attribute	16
3.12.2	The <i>type</i> attribute	17
3.12.3	The <i>coordinateBoundary</i> attribute	17
3.12.4	The <i>boundaryDomainType</i> attribute	17
3.13	The extended <i>Reaction</i> object	17
3.13.1	The <i>isLocal</i> attribute	17
3.14	The <i>Geometry</i> class	18
3.14.1	The <i>spatialId</i> attribute	18
3.14.2	The <i>coordinateSystem</i> attribute	18
3.14.3	The <i>listOf</i> container classes	19
3.15	The <i>CoordinateComponent</i> class	19
3.15.1	The <i>spatialId</i> attribute	19
3.15.2	The <i>componentType</i> attribute	19
3.15.3	The <i>coordinateType</i> attribute	20
3.15.4	The <i>sbmlUnit</i> attribute	21

3.16	The Boundary class	21
3.16.1	The spatialId attribute	21
3.16.2	The value attribute	21
3.17	The DomainType class	21
3.17.1	The spatialId attribute	22
3.17.2	The spatialDimension attribute	22
3.18	The Domain class	22
3.18.1	The spatialId attribute	22
3.18.2	The domainType attribute	23
3.19	The InteriorPoint class	23
3.19.1	The coord1 , coord2 , and coord3 attributes	23
3.20	The AdjacentDomains class	24
3.20.1	The spatialId attribute	24
3.20.2	The domain1 and domain2 attributes	24
3.21	The GeometryDefinition class	24
3.21.1	The spatialId attribute	24
3.22	The AnalyticGeometry class	24
3.23	The AnalyticVolume class	25
3.23.1	The spatialId attribute	25
3.23.2	The functionType attribute	26
3.23.3	The domainType attribute	26
3.23.4	The ordinal attribute	26
3.24	The Math class	26
3.25	The SampledFieldGeometry class	26
3.26	The SampledVolume class	26
3.26.1	The spatialId attribute	27
3.26.2	The domainType attribute	27
3.26.3	The sampledValue attribute	27
3.26.4	The minValue attribute	28
3.26.5	The maxValue attribute	28
3.27	The SampledField class	28
3.27.1	The spatialId attribute	28
3.27.2	The numSamples1 , numSamples2 , numSamples3 attributes	28
3.27.3	The dataType attribute	28
3.27.4	The interpolationType attribute	28
3.27.5	The encoding attribute	28
3.28	The ImageData class	28
3.28.1	The samples attribute	29
3.28.2	The samplesLength attribute	29
3.28.3	The dataType attribute	29
3.29	The CSGGeometry class	29
3.30	The CSGObject class	30
3.30.1	The spatialId attribute	30
3.30.2	The domain attribute	30
3.30.3	The ordinal attribute	30
3.30.4	The [csgNode] child	30
3.31	The CSGNode class	30
3.31.1	The spatialId attribute	30
3.32	The CSGPrimitive class	31
3.32.1	The primitiveType attribute	31
3.33	The CSGPseudoPrimitive class	31
3.33.1	The csgObjectRef attribute	31
3.34	The CSGSetOperator class	31
3.34.1	The operationType attribute	32
3.35	The ListOfCSGNodes class	32
3.36	The CSGTransformation class	32
3.36.1	The transformationType attribute	32
3.36.2	The csgNode child	32
3.37	The CSGTranslation class	32
3.37.1	The translateX attribute	32
3.37.2	The translateY attribute	33
3.37.3	The translateZ attribute	33
3.38	The CSGRotation class	33

3.38.1 The <code>rotateX</code> attribute	34
3.38.2 The <code>rotateY</code> attribute	34
3.38.3 The <code>rotateZ</code> attribute	34
3.38.4 The <code>rotationAngleInRadians</code> attribute	34
3.39 The <code>CSGScale</code> class	34
3.39.1 The <code>scaleX</code> attribute	34
3.39.2 The <code>scaleY</code> attribute	34
3.39.3 The <code>scaleZ</code> attribute	34
3.40 The <code>CSGHomogeneousTransformation</code> class	34
3.41 The <code>TransformationComponent</code> class	35
3.41.1 The <code>components</code> attribute	35
3.41.2 The <code>componentsLength</code> attribute	35
3.42 The <code>ParametricGeometry</code> class	35
3.43 The <code>ParametricObject</code> class	35
3.43.1 The <code>spatialId</code> attribute	35
3.43.2 The <code>polygonType</code> attribute	35
3.43.3 The <code>domain</code> attribute	35
3.44 The <code>PolygonObject</code> class	35
3.44.1 The <code>pointIndex</code> attribute	36
3.45 The <code>SpatialPoint</code> class	36
3.45.1 The <code>spatialId</code> attribute	36
3.45.2 The <code>coord1</code> , <code>coord2</code> , and <code>coord3</code> attributes	37
3.45.3 The <code>domain</code> attribute	37
4 Examples	38
5 Interaction with the Required Elements package	39
5.1 Compartments	39
5.2 Species	39
5.3 Reactions	39
5.4 Parameters	39
5.5 General	40
A Validation of SBML documents using Spatial constructs	41
Acknowledgments	43
References	44

1 Introduction

A set of biochemical process in cellular physiology may be modeled using different choices of spatial and temporal scales depending on the questions to be addressed. SBML Level 3 Core has explicit support for multi-compartmental modeling where cellular organization is approximated by a set of compartments (e.g. membrane-bound organelles) containing well-stirred populations of molecules. However, the coupling between localized biochemical reactions and diffusive molecular transport within the constraints of cellular geometry often results in important nonuniform molecular distributions. While it is often possible to approximate the influence of spatial organization and localization within a compartmental model using altered parameters and additional species, it is sometimes simpler and always more mechanistic to directly model these spatial processes.

An increasing number of modeling and simulation tools include direct support for modeling with explicitly defined cellular geometry. These models generally include heterogeneous molecular distributions, diffusive transport, and spatially localized reactions. These spatial models generally belong to two different mathematical frameworks, stochastic (where each molecule is tracked in space and time) and deterministic (where time varying species concentration fields are described by partial differential equations).

There are a sufficient number of spatial modeling tools and spatial models to justify the effort of creating a spatial modeling extension of SBML. All such models must describe the cellular geometry, map molecular species to spatial locations, map reactions to spatial locations, and specify molecular transport within geometric compartments and at boundaries of these compartments.

It is the purpose of this SBML Level 3 extension to define a common representation for cellular geometry, spatial mappings of species and reactions, and explicit species transport.

1.1 Proposal corresponding to this package specification

This specification for Spatial in SBML Level 3 Version 1 is based on the proposal located at the following URL:

<https://sbml.svn.sf.net/svnroot/sbml/trunk/specifications/sbml-level-3/version-1/spatial/proposal>

The tracking number in the SBML issue tracking system (SBML Team, 2010) for Spatial package activities is 188 (<http://sourceforge.net/p/sbml/sbml-specifications/188/>).

1.2 Package dependencies

The Spatial package has no dependencies on other SBML Level 3 packages.

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:

AbstractClass: Abstract classes are never instantiated directly, but rather serve as parents of other classes. Their names begin with a capital letter and they are printed in a slanted, bold, sans-serif typeface. In electronic document formats, the class names defined within this document are also hyperlinked to their definitions; clicking on these items will, given appropriate software, switch the view to the section in this document containing the definition of that class. (However, for classes that are unchanged from their definitions in SBML Level 3 Core, the class names are not hyperlinked because they are not defined within this 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, as in the previous case, class names are not hyperlinked if they are for classes that are unchanged from their definitions in the SBML Level 3 Core 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.

[elementName]: In some cases, an element may contain a child of any class inheriting from an abstract base class. In this case, the name of the element is indicated by giving the abstract base class name in brackets, meaning that the actual name of the element is the de-capitalized form of whichever subclass is used.

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

2.1 Problems with current SBML approaches

There is no standard way of specifying spatial models in SBML short of introducing an explicit spatial discretization in the form of a large number of compartments with duplicate species and reactions and additional reactions for coupling due to transport. This approach hard-codes the numerical methods which destroys portability and is not practical beyond a few compartments. Tools have been forced to resort to proprietary extensions (e.g. MesoRD custom annotations) to encode geometry.

2.2 Past work on this problem or similar topics

There are many standards for the exchange of geometric information of engineered parts in Computer Aided Design and Manufacturing. These formats are designed for geometric shapes which are directly specified by a designer rather than the data driven, freeform biological structures encountered in cell biology.

There also exist standards for the representation of unstructured computational meshes that can encode these freeform biological structures more readily. However, it is important to note that while a computational mesh necessarily encodes an approximation to the shapes of geometric objects, the particular form will be algorithm dependent.

To ensure model interoperability, we must encode the geometric shapes in a way that is independent of the numerical methods and even the mathematical framework. The representation of a spatial model within SBML should be largely invariant of the particular encoding of the geometry definition within that model. For example, a spatial model represented in SBML that encodes geometry as a set of geometric primitives (e.g. spheres, cylinders) should be easily portable to a tool that only supports polygonal surface tessellation. It is expected that a geometry translation library will be very useful for interoperability the same way that libSBML greatly improved model interchange by solving similar implementation problems in a standard way.


2.3 Prior work

The first version of the Spatial proposal was written [fill in history]

 **Lucian: We probably do need something here.**

3 Package syntax and semantics

This section contains a definition of the syntax and semantics of the Spatial package for SBML Level 3 Version 1 Core. The Spatial package involves several new object classes, and extends the existing **Model**, **Compartment**, **Species**, **Reaction**, and **Parameter** object class. [Section 4 on page 38](#) contains complete examples of using the constructs in SBML models.

 **Lucian:** Periodically when I have comments, I'll put them in sections that look like this—in red, with the pointy-hand icon off to the side. They tend to be design questions I had when creating this document for parts I thought were not clear, or are suggestions for changes that could be made.

3.1 Overview of spatial extension

The SBML **Compartment**, **Reaction** and **Species**, and molecular transport mechanisms (**DiffusionCoefficient**, **AdvectionCoefficient**, **BoundaryCondition**) are mapped to geometric domains to describe spatial models within SBML. The primary mechanism to accomplish this mapping is to simply map **Compartments** to collections of geometric Domains called **DomainTypes**. Each **Domain** is a contiguous patch of volumetric space or a contiguous surface patch that is ultimately described by a single system of equations (whichever mathematical framework is used). In analogy with initial conditions, the mathematical system defined within a domain often needs a definition of what happens at the domain boundary (e.g. boundary conditions) to complete the specification. Because of this, the boundaries between adjacent domains need to be identified so that appropriate boundary conditions can be specified. For compactness of representation, rather than map to each individual **Domain**, **Compartments** are mapped to **DomainTypes**, along with the corresponding **Species** and **Reactions** (with the new compartment attribute).

3.1.1 Geometry

The **Geometry** object within a model is completely modular and does not reference the rest of the model, promoting reuse of the same geometry in different models. The geometry separately defines a coordinate system, a list of domain types, a list of domains and their adjacency relationships, and a list of alternate geometric representations.

3.1.2 Alternative GeometryDefinitions

Modeling and simulation tools will each natively support some subset (often just one) of the possible **GeometryDefinitions** (analytic, sampled field, constructive solid geometry, and parametric shapes). Interoperability will be enhanced if tools write as many geometry definitions as they are able. Upon reading the model, a tool will typically choose the most convenient geometry definition, i.e. the one that it natively supports. If a tool does not edit the geometry, it has the ability to preserve the alternate representations during model editing (because the mapping of the model to the geometry is not stored in the geometry).

There are two general classes of geometric representation specification: those that explicitly specify surfaces and those that implicitly specify surfaces. For example, a level set is a field where a specific isosurface of the field specifies a geometric surface. A geometry described using constructive solid geometry of geometric primitives (e.g. spheres, cylinders) specifies directly which points are "inside" an object. Alternatively, explicit surface representations explicitly declare the set of points belonging to surfaces (e.g. polygonal tessellations).

3.2 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 Spatial package for SBML Level 3 Version 1 Core:

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

In addition, SBML documents using a given package must indicate whether the package can be used to change the mathematical interpretation of a model. This is done using the attribute **required** on the `<sbml>` element in the SBML document. For the Spatial package, the value of this attribute must be “**true**”, because the use of the Spatial package can change the mathematical meaning of a model.

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

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

3.3 Primitive data types

The Spatial package uses a number of the primitive data types described in Section 3.1 of the SBML Level 3 Version 1 Core specification, and adds four additional primitive types described below.

3.3.1 Type SpId

The type **SpId** is derived from **SId** (SBML Level 3 Version 1 Core specification Section 3.1.7) and has identical syntax. The **SpId** type is used as the data type for the identifiers of various objects in the Spatial Processes package. The purpose of having a separate type for such identifiers is to enable the space of possible spatial identifier values to be separated from the space of all other identifier values in SBML. The equality of **SpId** values is determined by an exact character sequence match; i.e., comparisons of these identifiers must be performed in a case-sensitive manner.

3.3.2 Type SpIdRef

Type **SpIdRef** is used for all attributes that refer to identifiers of type **SpId**. This type is derived from **SpId**, but with the restriction that the value of an attribute having type **SpIdRef** must match the value of a **SpId** attribute in the relevant model; in other words, the value of the attribute must be an existing spatial identifier in the referenced model. As with **SpId**, the equality of **SpIdRef** values is determined by exact character sequence match; i.e., comparisons of these identifiers must be performed in a case-sensitive manner.

3.3.3 Type BoundaryConditionKind

The **BoundaryConditionKind** primitive data type is used in the definition of the **BoundaryCondition** class. The type **BoundaryConditionKind** is derived from type **string** and its values are restricted to being one of the following possibilities: “**Robin_valueCoefficient**”, “**Robin_inwardNormalGradientCoefficient**”, “**Robin_sum**”, “**inwardNormalFlux**”, and “**value**”. Attributes of type **BoundaryConditionKind** cannot take on any other values. The meaning of these values is discussed in the context of the **BoundaryCondition** class’s definition in [Section 3.12 on page 16](#).

3.3.4 Type ComponentKind

The **ComponentKind** primitive data type is used in the definition of the **CoordinateComponent** class. **ComponentKind** is derived from type **string** and its values are restricted to being one of the following possibilities: “**cartesianX**”, “**cartesianY**”, “**cartesianZ**”, “**sphericalRadius**”, “**sphericalAzimuth**”, “**sphericalElevation**”, “**cylindricalRadius**”, “**cylindricalAzimuth**”, “**cylindricalHeight**”, “**polarRadius**”, and “**polarAzimuth**”. Attributes of type **ComponentKind** cannot take on any other values. The meaning of these values is discussed in the context of the **CoordinateComponent** class’s definition in [Section 3.15 on page 19](#).

3.3.5 Type CoordinateKind

The `CoordinateKind` primitive data type is used in the definition of the `CoordinateComponent` class. The type `CoordinateKind` is derived from type `string` and its values are restricted to being one of the following possibilities: “normal”, “symmetrical”, and “fixed”. Attributes of type `CoordinateKind` cannot take on any other values. The meaning of these values is discussed in the context of the `CoordinateComponent` class’s definition in [Section 3.15 on page 19](#).

3.3.6 Type DiffusionKind

The `DiffusionKind` primitive data type is used in the definition of the `DiffusionCoefficient` class. `DiffusionKind` is derived from type `string` and its values are restricted to being one of the following possibilities: “isotropic” and “tensor”. Attributes of type `DiffusionKind` cannot take on any other values. The meaning of these two values is discussed in the context of the `DiffusionCoefficient` class’s definition in [Section 3.10 on page 14](#).

3.3.7 Type FunctionKind

The `FunctionKind` primitive data type is used in the definition of the `AnalyticVolume` class. The type `FunctionKind` is derived from type `string` and its values are restricted to being one of the following possibilities: “layered” and “R-function”. Attributes of type `FunctionKind` cannot take on any other values. The meaning of these values is discussed in the context of the `AnalyticVolume` class’s definition in [Section 3.23 on page 25](#).

3.3.8 Type GeometryKind

The `GeometryKind` primitive data type is used in the definition of the `Geometry` class. `GeometryKind` is derived from type `string` and its values are restricted to being one of the following possibilities: “cartesian”, “cylindrical”, “spherical”, and “polar”. Attributes of type `GeometryKind` cannot take on any other values. The meaning of these values is discussed in the context of the `Geometry` class’s definition in [Section 3.14 on page 18](#).

3.3.9 Type SetOperation

The `SetOperation` primitive data type is used in the definition of the `CSGSetOperator` class. The type `SetOperation` is derived from type `string` and its values are restricted to being one of the following possibilities: “union”, “intersection”, and “difference”. Attributes of type `SetOperation` cannot take on any other values. The meaning of these values is discussed in the context of the `CSGSetOperator` class’s definition in [Section 3.34 on page 31](#).

3.3.10 Type doubleArray

The `doubleArray` primitive data type is a space-delimited list of `double` values in a single string.

3.3.11 Type integerArray

The `integerArray` primitive data type is a space-delimited list of `integer` values in a single string.

3.4 The extended **Model** object

The **Model** object is extended in the spatial package to contain a new **Geometry** child, as seen in Figure 1. The **Geometry** element is contained in the Model element in the 'spatial' namespace. In order to specify a spatial geometry, some of the existing SBML elements need to be extended (**Compartment**, **Species**, **Parameter**, and **Reaction**). These extensions to the SBML elements are discussed in the sections that follow.

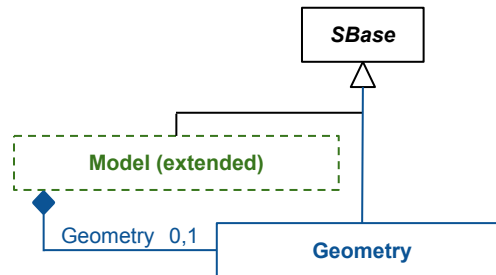


Figure 1: The definition of the extended **Model** object from the Spatial package. The **Geometry** object and its children are defined in their own sections.

3.5 The extended **Compartment** object

The **Compartment** in the SBML core is extended while defining a spatial model. An SBML model with spatial geometry defines domain types (classes of domains that are anatomically and physiologically similar). These domain types need to be mapped to a compartment in the SBML model. **Compartments** are extended to define **CompartmentMappings** that map compartments to **DomainTypes** such that each corresponding **DomainType** is assigned the same biological and mathematical function. Within SBML L3 Core, the compartment **Sid** refers to the size of that compartment and is specified by the size attribute or may be set by a rule. For spatial models, the compartment size is calculated as the product of the unit size specified in the compartment mapping and the size of the current domain. The definition for the extension of the **Compartment** element is shown in Figure 2.

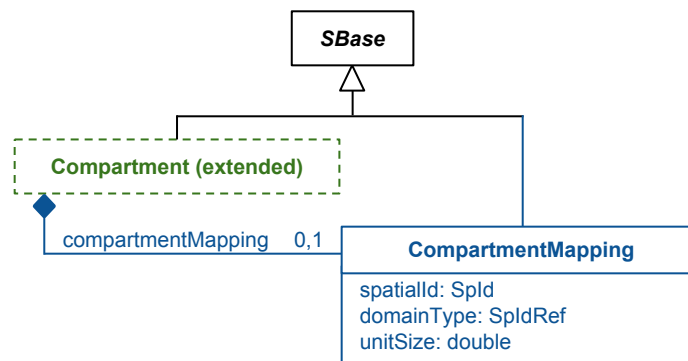


Figure 2: The definition of the extension to the **Compartment** element, and the definition of the **CompartmentMapping** class. The SBML core attributes of **Compartment** are not displayed.

The **Compartment** element has an optional **CompartmentMapping** child which indicates the **DomainType** to which the **Compartment** is mapped. If there is no **CompartmentMapping** for a **Compartment** in a spatial model, then that **Compartment** is excluded from the spatial version of the model. In the same way, if a **DomainType** is not mapped to one or more **Compartments**, then the corresponding **Domains** in the geometry have no assigned function.

3.6 The `CompartmentMapping` class

Each **Compartment** in a model that defines a spatial geometry may contain an optional **CompartmentMapping**. A **CompartmentMapping** is defined as part of the model rather than part of the geometry so that the geometry is modular and may be readily shared between models and reused. A **CompartmentMapping** maps a **Compartment** defined in the model to a **DomainType** defined in the geometry such that each corresponding **DomainType** is assigned the same biological and mathematical function described by the set of **Compartments** that are mapped to that **DomainType**.

This mapping need not be one-to-one. In fact, it is common to map er-lumen, er-membrane, and cytosol to the same cell interior volume or 3D **DomainType**. The `unitSize` attribute specifies the relative quantity of each **Compartment** that is mapped to the **DomainType**.

3.6.1 The `spatialId` attribute

The `spatialId` attribute is a mandatory attribute of type `SpId` that is used to uniquely identify a **CompartmentMapping** in the model. All identifiers of type `SpId` must be unique within the **Geometry**. The mathematical value of a **CompartmentMapping** is its `unitSize` attribute, and can be bound to a **Parameter** by using a **SpatialSymbolReference**.

3.6.2 The `domainType` attribute

The mandatory `domainType` attribute is of type `SpIdRef` that indicates a **DomainType** defined in the **Geometry** element.

3.6.3 The `unitSize` attribute

The `unitSize` attribute is of type `double` and represents the relative size of the **Compartment** with respect to the size of the **Domains** to which they are mapped. Thus for any infinitesimal subset of the **Domain** with size S , there exists an amount of Compartment_i of size $(S * \text{unitSize}_i)$ for $i=1..N$ compartments mapped to that **DomainType**. For example, a 3D **Compartment** (and **DomainType**) which is mapped to a 3D **DomainType** has a `unitSize` which is a volume fraction of dimensionless unit. The total set of all such volume fractions mapped to a particular **DomainType** will typically sum to one.

If the `spatialDimensions` attribute of the parent **Compartment** is different than the `spatialDimension` attribute of referenced **DomainType**, the `unitSize` attribute is a conversion factor between the two. The most common example of this would be a 2D **Compartment** being mapped to a 3D **DomainType**, such as an ER-membrane being mapped to a volumetric cell interior. In this case, the `unitSize` is a surface-to-volume ratio.

If connected to a **Parameter** via a **SpatialSymbolReference**, an **InitialAssignment** may override the value of the `unitSize` attribute. It is theoretically possible to have this value change in time through the use of a **Rule** or **Event**, but some (if not all) software tools may not support this setup. If the value is set to change, and the dimensionality of the parent **Compartment** and referenced **DomainType** is the same, the other **CompartmentMapping** elements for the same **DomainType** will typically change in concert, so that they continue to sum to one.

Any bound **Parameter**'s units should be equivalent to the units of the parent **Compartment** divided by the units of the referenced **DomainType**.

3.7 The extended **Species** object

The SBML core **Species** is extended when a spatial geometry is defined in the model with the addition of a single new required boolean “`isSpatial`” attribute. The extension to the **Species** element is shown in [Figure 3 on the following page](#).

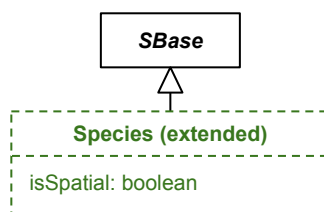


Figure 3: The extension to the **Species** element. The attributes of **Species** from SBML Level 3 Version 1 Core are not displayed.

3.7.1 The **isSpatial** attribute

The **isSpatial** attribute is of data type boolean. If it is set to true, the **Species** is spatially distributed in a possibly nonhomogeneous manner within the **Domains** of the same type as the mapped **DomainType**.

For continuous deterministic models (described by partial differential equations), a spatial **Species** will result in a concentration field described by a partial differential equation which incorporates contributions from **Reactions**, diffusion (**DiffusionCoefficient**) and advection (**AdvectionCoefficient**) and are subject to boundary conditions (**BoundaryCondition**) and initial conditions (**InitialAssignment** and **Rule**). All of these quantities can be explicit functions of the spatial coordinates as well as spatial and nonspatial **Parameters** and **Species**.

For stochastic models, the **Species** is represented as a collection of particles that are distributed throughout the **Domains** and are subject to reactions, diffusion and advection. Simulation algorithms either track individual particles (e.g. Particle-based methods) or use spatial discretization to track a large number of well stirred pools (e.g. Next-Subvolume Method).

The **compartment** of any **Species** set **isSpatial** = “true” must have a child **CompartmentMapping**: if it did not, its compartment would not actually be a part of the spatial model.

3.8 The extended **Parameter** object

When an SBML model defines a spatial geometry, the SBML core **Parameter** is used to define the diffusion coefficient, transport velocity (advection) and boundary conditions for species and the coordinate components defined in the **Geometry**. One **Parameter** is created for each quantity, by adding a child **DiffusionCoefficient**, **AdvectionCoefficient**, or **BoundaryCondition**. Conversely, some elements defined in the spatial package may need to be referenced by mathematics in core constructs, or even have their value set by core constructs such as **InitialAssignment** or **Rule**. These spatial elements can be semantically linked to a **Parameter** by giving it a child **SpatialSymbolReference** pointing to that element.

A **Parameter** that has been extended for the Spatial package can have only one of the above listed objects. For example, if a **Parameter** is extended to represent the diffusion coefficient of a species, the existing attributes of the **Parameter** (id, name, value, units, constant) are defined according to SBML core specifications, along with a **DiffusionCoefficient** child that contains the information about the species it represents. Figure 4 on the next page represents the extension to the **Parameter** element.

3.9 The **SpatialSymbolReference** class

A **Parameter** is extended with a **SpatialSymbolReference** element, when a symbol from the defined spatial geometry (**spatialId** of any element contained in **Geometry**) is required to be used in the SBML core model. Typically, the **SpatialSymbolReference** is used to represent the coordinate components defined in the **Geometry**’s **listOfCoordinateComponents**. For example, if the **Geometry** is defined in a 2-dimensional Cartesian coordinate system with X and Y defined as coordinate components, two **Parameters** (one each for **CoordinateComponents** X and Y) are created in the model. The value of the parameter is not required to be set. For each of these parameters, a **SpatialSymbolReference** object is created.

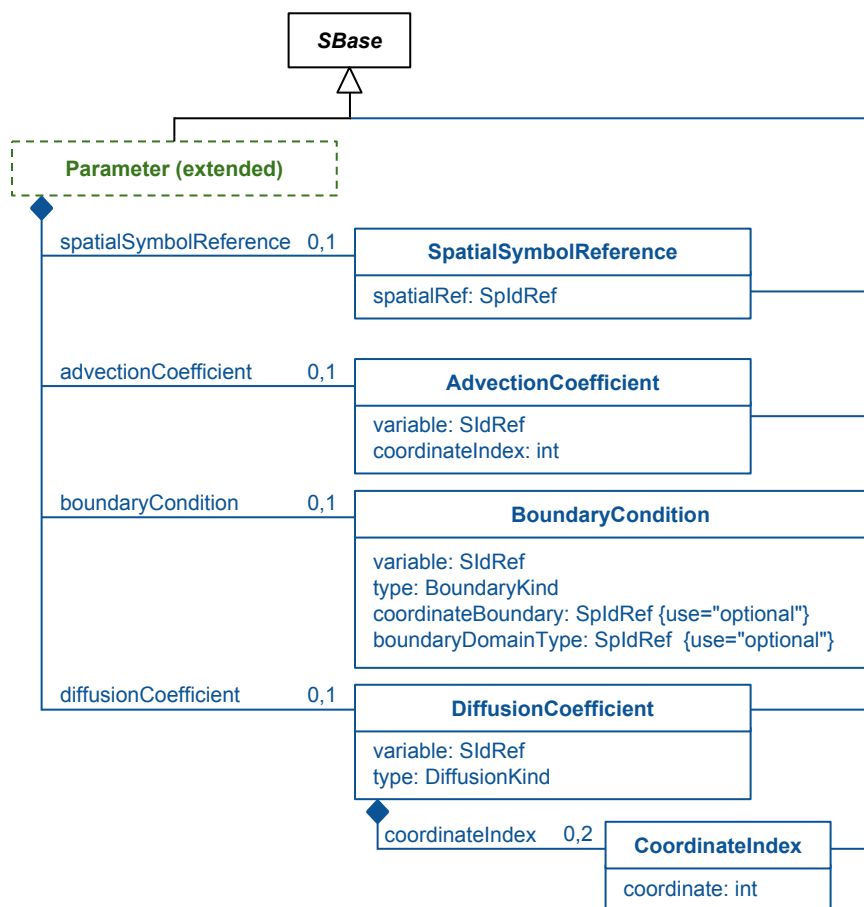


Figure 4: The **Parameter** element extension for spatial package. The SBML Level 3 Version 1 Core attributes for **Parameter** are not displayed in this figure.

3.9.1 The spatialRef attribute

The **spatialId** attribute of **SpatialSymbolReference**, is of type **SpIdRef** and refers to the **spatialId** of any element defined in the **Geometry** of the model.

3.9.2 The type attribute

 **Lucian:** From discussion on the spatial list, this attribute was dropped.

3.10 The DiffusionCoefficient class

When a species in a spatial model has a diffusion rate constant, a **Parameter** for this diffusion constant is created in the SBML model with a **DiffusionCoefficient** child, which is used to identify the **Species** whose diffusion rate the **Parameter** represents. The diffusion coefficient can then be set like any other variable: its initial value can be set using the **Parameter**'s **value** attribute or through an **InitialAssignment**, and if the diffusion coefficient changes in time, this can be defined with a **Rule** or **Event**. If set, the units of this **Parameter** should be $\text{length}^2/\text{time}$. If left unset, the **DiffusionCoefficient** will inherit the model units of $\text{length}^2/\text{time}$ (typically cm^2s^{-1} or um^2s^{-1}).

It is possible to define both diffusion and advection for the same **Species**.

3.10.1 The variable attribute

The required **variable** attribute of **DiffusionCoefficient** is of type **SIdRef** and is the SId of the **Species** or **Parameter** in the model whose diffusion coefficient is being set.

3.10.2 The type attribute

The required **type** attribute of **DiffusionCoefficient** is of type **DiffusionKind** and indicates whether the diffusion coefficient is “isotropic” (i.e. applies equally in all dimensions/directions) or “tensor” (i.e. applies only for a particular pair of coordinate indices). Coefficients of type “isotropic” may not have any child **CoordinateIndex** children, while coefficients of type “tensor” must have exactly two.

3.10.3 The **CoordinateIndex** class

The optional **CoordinateIndex** children of the **DiffusionCoefficient** object represent the index attributes of the **CoordinateComponent** (e.g. 0 for x, 1 for y, 2 for z), for specifying the diffusion coefficient for flux in the plane defined by the two **CoordinateIndex** axes. **coordinateIndex** direction due to a gradient in the **coordinateIndex** direction (diagonal term of the diffusion tensor). If the **coordinateIndex** is missing, the diffusion is considered to be Isotropic.



Lucian: I updated this some, but was also thinking: if we remove the 'index' value from the **CoordinateComponent**, it might make sense for the **coordinateIndex** attribute to be of type **SpIdRef** instead, and have it reference the actual coordinate component from the model. In other words, instead of 'coordinateIndex=0', you'd have 'coordinate="x"'. Does this seem reasonable?

Also, I'm not sure how this would work for cylendrical/spherical coordinate systems. For cylendrical coordinates, the radius and azimuth would define the plane of the base circle, but what would the difference be between the 'radius and height' plane vs. the 'azimuth and height' plane? Would they both define the same thing? And is it at all possible to define a plane by referencing two axes of a spherical coordinate system? Must all diffusion be isotropic if using that coordinate system?

3.10.4 **DiffusionCoefficient** uniqueness

Only one **DiffusionCoefficient** may be defined per **Species** per valid axis in the **Compartment** in which it resides. Since isotropic diffusion is defined for all axes at once, this means that if an isotropic **DiffusionCoefficient** is defined for a **Species**, it may have no other diffusuion coefficients.

3.11 The **AdvectionCoefficient** class

The **AdvectionCoefficient** is the extension to **Parameter** in SBML core that is used to represent transport velocity of a species, if it exists. The transport velocity for the species is defined in a manner similar to the diffusion constant with a unit of length/time (regardless of the units of the corresponding **Species**’ “**compartment**” attribute). A **Parameter** is created in SBML code for the velocity with an **AdvectionCoefficient** child to identify the **Species** whose velocity is represented by the **Parameter**; its initial value is set either through the **value** attribute or an **InitialAssignment**. If the advection coefficient changes in time or space, this can be modeled with a **Rule** or **Event**.

If defined, the units of the parent **Parameter** should be in length/time; if not defined, it inherits from the model-wide units of length divided by the model-wide units of time.


It is possible to define both diffusion and advection for the same **Species**.

3.11.1 The variable attribute

The **variable** attribute of **AdvectionCoefficient** is of type **SIdRef** and is the SId of the **Species** or **Parameter** in the model whose advection coefficient (transport velocity) is being set.


3.11.2 The **coordinateIndex** attribute

The **coordinateIndex** is of type **int** and represents the coordinate component of the velocity. For example, if the **Geometry** is defined in the Cartesian coordinate system and is 2-dimensional, the species can have velocity terms in X and Y (assuming that X and Y are the coordinates that define the 2 dimensions of the **Geometry**). If the **Parameter** represents the transport velocity of the species in the X-coordinate, the **coordinateIndex** attribute will take a value of 0, and if it represents the velocity in the Y-coordinate, the attribute will take a value of 1. Only one **AdvectionCoefficient** may be defined per **Species** per valid **coordinateIndex**.

 Lucian: Again, this might be clearer as a 'coordinate' attribute of type 'SpIdRef' instead of linking things to indices. Also, is it possible to define advection in the 'radius' or 'azimuth', etc. directions?

3.12 The **BoundaryCondition** class

A **Species** in a spatial model that has a diffusion rate or an advection velocity needs to have specified boundary conditions. A boundary condition is either the concentration of the species or the flux density of the species at a boundary. The boundary refers to either an internal membrane boundary or a face of the box defined by the minimum and maximum coordinates of the geometry (the geometries bounding box).


 Lucian: Is there any way to tell the difference between a flux boundary condition and a concentration boundary condition other than the units of the corresponding **Parameter**? I think there should be, perhaps with a new 'type' attribute—would this work?

When creating a spatial SBML model, species boundary conditions are created as parameters, one for each boundary condition, by adding a child **BoundaryCondition** that points to the corresponding **Species** and boundary, depending on the coordinate system.


For Cartesian Geometries, there are two boundaries for every axis being modeled. For example, in a 2D cartesian geometry for the external boundaries, four parameters are created for each spatial **Species** (corresponding to the boundary conditions at each of the minimum and maximums of the X and Y axes).

For polar and cylindrical Geometries, there is one boundary for the maximum of the radius, and another boundary for the minimum of the radius if that minimum is not "0". For the azimuth, there are either two boundaries corresponding to the minimum and maximum azimuth, or no boundaries, if the minimum is 0°, and the maximum is 360°. In cylindrical geometry, there will be two boundaries for the minimum and maximum height.

For spherical Geometries, there will again be one boundary for the maximum of the radius, and another boundary for the minimum of the radius if that minimum is not "0". For the spherical azimuth and elevation, there will be either two boundaries each corresponding to the minimum and maximum values, or no boundaries, for each minimum 0° and maximum 360° pair.

 Lucian: I basically made up the above, based on my understanding of the different coordinate systems. If it's wrong, let me know! Also, I imagine it might be optional to define a concentration at the boundary 'radius=0' in some cases—should we allow this?

The **Parameter**'s value is set either through the **value** attribute or an **InitialAssignment**. If the boundary condition changes in time, it can be set with a **Rule** or **Event**. The **Parameter** unit is set to be the unit of the boundary condition, namely, the concentration of the **Species** at the boundary. Only one **BoundaryCondition** may be defined per **Species** per boundary (regardless of type).

 Lucian: Make sure I got the unit description correct here. Also, it might need more explanation, particularly to account for non-3d compartments.


3.12.1 The **variable** attribute

The **variable** attribute of **BoundaryCondition** is of type **SIdRef** and is the **SId** of the **Species** or **Parameter** in the model whose boundary condition is being set.

3.12.2 The type attribute

The **type** attribute is of type **BoundaryConditionKind** and indicates the type of boundary condition. The boundary condition types come in three groups: for Neumann boundaries, “**inwardNormalFlux**” is used. For Dirichlet boundaries, “**value**” is used. For Robin boundaries, three different Parameters must be defined, with **BoundaryCondition** elements of type “**Robin_valueCoefficient**”, “**Robin_inwardNormalGradientCoefficient**”, and “**Robin_sum**”.

The unit of the boundary condition is determined by the type, and the unit for density and velocity. For “**value**”, the unit would be the unit of concentration. For “**inwardNormalFlux**”, the unit would be concentration*length/time. For Robin boundaries, for a variable “**u**” with an inward pointing normal vector “**n**”, Robin value coefficient “**a**”, Robin inward normal gradient coefficient “**b**”, and Robin sum “**g**”, the condition is defined by the equation “**a*u + b*inner_product(grad(u),n) = g**”, with appropriate units.

 **Lucian:** Does this explanation work?

3.12.3 The coordinateBoundary attribute

The **coordinateBoundary** attribute is of type **SpIdRef** and refers to the **spatialId** of either the **boundaryMin** or **boundaryMax** object of the **CoordinateComponent** defined in **Geometry**. This **spatialId** indicates the boundary condition (minimum or maximum) in the **CoordinateComponent**. A **Parameter** that is extended with a **BoundaryCondition** object can only define the **coordinateBoundary** attribute or the **boundaryDomainType** attribute, but not both.

 **Lucian:** Can this reference the new ‘boundaryFixed’ objects, too?

3.12.4 The boundaryDomainType attribute

The **boundaryDomainType** attribute is of type **SpIdRef** and refers to the **spatialId** of the **DomainType** of the location of the species whose boundary condition is being defined. A **Parameter** that is extended with a **BoundaryCondition** object can only define the **coordinateBoundary** attribute or the **boundaryDomainType** attribute, but not both.

3.13 The extended Reaction object

The SBML core **Reaction** is extended when a spatial geometry is defined in the model with the addition of a single new required boolean **isLocal** attribute. Figure 5 displays the definition of the extension of the **Reaction** element.

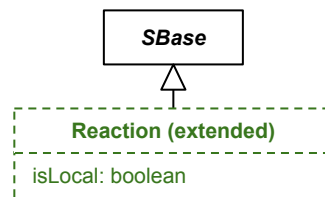


Figure 5: The extension to the **Reaction** element. The SBML Level 3 Version 1 Core attributes and children for **Reaction** are not displayed in the figure.

3.13.1 The isLocal attribute

The **isLocal** attribute for a **Reaction** is of type **Boolean**. The attribute is set to true if the reaction is to be considered a local description of the reaction in terms of concentration/time defined at each point in space rather than substance/time over an entire **Compartment** or “pool”. Note that this means that the units of the **KineticLaw** are different depending on whether the **Reaction** is local or not.

3.14 The Geometry class

A single geometry must be defined within the model if the spatial extension is to be used. Figure 6 shows the definition of the **Geometry** element.

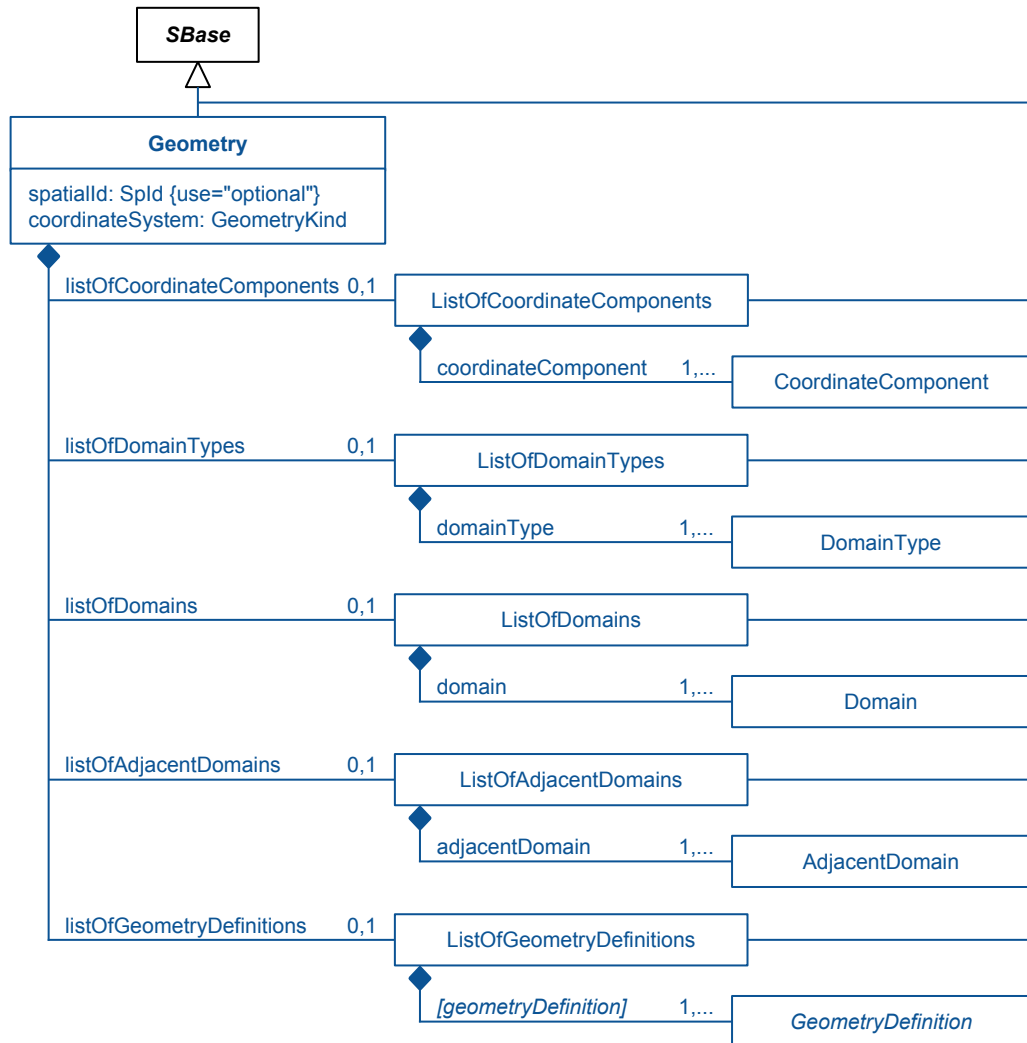


Figure 6: The definition of the **Geometry**, **ListOfCoordinateComponents**, **ListOfDomainTypes**, **ListOfDomains**, **ListOfAdjacentDomains**, and **ListOfGeometryDefinitions** classes from the *Spatial* package. The various children of the **ListOf**-classes are defined in their own sections.


3.14.1 The spatialId attribute

The **spatialId** attribute is of type **SpId**, uniquely identifies the **Geometry** element, and is optional. It has no mathematical meaning.


3.14.2 The coordinateSystem attribute

The **coordinateSystem** attribute is a required attribute and is of type **GeometryKind**. It represents the coordinate system used by the **Geometry**. A value of “**cartesian**” indicates that the geometry is a cartesian coordinate system, with the coordinate components corresponding to the x, y, and z components of that system (which could be 1-, 2-, or

3-dimensional). A value of “**cylindrical**” indicates that the geometry is a three-dimensional cylindrical coordinate system, with coordinate components corresponding to radial distance, azimuth angle, and axial position. A value of “**spherical**” indicates that the geometry is a three-dimensional spherical coordinate system, with coordinate components corresponding to radial distance, polar angle, and azimuth angle. A value of “**polar**” indicates that the geometry is a two-dimensional polar coordinate system, with coordinate components corresponding to radial distance and angle (the two-dimensional equivalent of the cylindrical coordinate system).

 **Lucian:** For cartesian coordinates, does anyone use higher-dimensional modeling for anything? Would it be worth trying to expand the spec to cover n-dimensional cartesian coordinates? I suppose the safest bet would be to say something like ‘future versions of this specification might be expanded to cover n-dimensional modeling if there is an expression of interest from the modeling community to exchange these types of models’.

Other coordinate systems in use in biological models are possible, such as cylindrical symmetry where only the “rho” coordinate component is required and the geometry is one dimensional. However, these are not directly supported by the Spatial package.

 **Lucian:** That paragraph was old text—are those systems actually supported now by the latest updates? It seems to me like they may well be.

3.14.3 The *listOf container classes*

The **Geometry** has **listOfCoordinateComponents**, **listOfDomainTypes**, **listOfDomains**, and **listOfAdjacentDomains**, and **listOfGeometryDefinitions** that help define the geometry. The **ListOfCoordinateComponents** is a list of **CoordinateComponent** objects, the **ListOfDomainTypes** is a list of **DomainType** objects, the **ListOfDomains** is a list of **Domain** objects, **ListOfAdjacentDomains** is a list of **AdjacentDomains** objects, and the **ListOfGeometryDefinitions** is a list of alternative **GeometryDefinitions** (**ParametricGeometry**, **CSGeometry**, **SampledFieldGeometry**, **AnalyticGeometry**). None of these lists are technically required, but, if present, none of them may be empty.

Note that the children of the **ListOfGeometryDefinitions** object are not called **geometryDefinition** but rather take the name of the derived class, decapitalized. Thus, they may be called **parametricGeometry**, **sampledFieldGeometry**, **csGeometry**, or **analyticGeometry**.

3.15 The **CoordinateComponent** class

A **CoordinateComponent** object explicitly defines a coordinate component of the coordinate axes and gives them names, units, and formally associates them with a coordinate system. The **CoordinateComponent** also defines the minimum and maximum values of the coordinate axis it represents. The definition of **CoordinateComponent** is shown in [Figure 7 on the following page](#).

3.15.1 The *spatialId attribute*

A **CoordinateComponent** is identified with the **spatialId** attribute which is of type **SpId**. When referenced (for example, via a **SpatialSymbolReference** extension of the SBML core **Parameter** object) and used within a mathematical expression, it represents the coordinate value for this **CoordinateComponent**.

Because a **CoordinateComponent** represents an entire axis, it is not appropriate, should it be connected to a **Parameter** via a **SpatialSymbolReference**, for that **Parameter** to be set via an **InitialAssignment** or **Rule**. Rather, it is treated like the SBML core **csymbol** “time”, and can be used as an independent variable in other calculations.

3.15.2 The *componentType attribute*

The **componentType** attribute of type **ComponentKind** represents the type of the coordinate component, and may take one of a subset of all possible **ComponentKind** values depending on its parent **Geometry**, as defined in [Table 1 on the next page](#). For all **GeometryKind** values except “cartesian”, exactly one **CoordinateComponent** must exist with each of the allowed **ComponentKind** values: in other words, a “cylindrical” **Geometry** must have exactly three **CoordinateComponent** children, each one having a different allowed value for its **componentType** attribute.

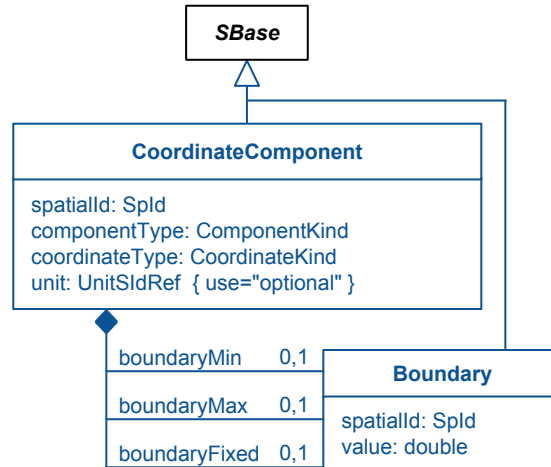


Figure 7: The `CoordinateComponent` object definition. One or more instances of `CoordinateComponent` objects in a `ListOfCoordinateComponents` can be present in `Geometry`.

For Cartesian geometries, one-dimensional geometries may be defined by having a single “`cartesianX`” `CoordinateComponent`; two-dimensional geometries may be defined by having two `CoordinateComponent` children with `componentType` values of “`cartesianX`” and “`cartesianY`”, and three-dimensional geometries may be defined by having three `CoordinateComponent` children with `componentType` values of “`cartesianX`”, “`cartesianY`”, and “`cartesianZ`”.

GeometryKind	Dimensions	ComponentKinds
cartesian	1	“ <code>cartesianX</code> ” (<code>coord1</code>)
cartesian	2	“ <code>cartesianX</code> ” (<code>coord1</code>) and “ <code>cartesianY</code> ” (<code>coord2</code>)
cartesian	3	“ <code>cartesianX</code> ” (<code>coord1</code>), “ <code>cartesianY</code> ” (<code>coord2</code>), and “ <code>cartesianZ</code> ” (<code>coord3</code>)
cylindrical	3	“ <code>cylindricalRadius</code> ” (<code>coord1</code>), “ <code>cylindricalAzimuth</code> ” (<code>coord2</code>), and “ <code>cylindricalHeight</code> ” (<code>coord3</code>)
polar	2	“ <code>polarRadius</code> ” (<code>coord1</code>) and “ <code>polarAzimuth</code> ” (<code>coord2</code>)
spherical	3	“ <code>sphericalRadius</code> ” (<code>coord1</code>), “ <code>sphericalAzimuth</code> ” (<code>coord2</code>), and “ <code>sphericalElevation</code> ” (<code>coord3</code>)

Table 1: Correspondance between the type of a `Geometry` and the possible `componentTypes` of it child `CoordinateComponent` elements. Also noted is the corresponding attribute (`coord1`, `coord2`, or `coord3`) corresponding to each axis when defining `InteriorPoint` elements (see Section 3.19 on page 23).

3.15.3 The `coordinateType` attribute

The `coordinateType` attribute, of type `CoordinateKind`, represents whether the axis being defined is normal (“`normal`”), or whether it represents a dimension that does not need to be modeled explicitly in any performed calculations, whether due to symmetry, where the modeled element exists in the given dimension, but is assumed to be the same for all values along that axis (“`symmetrical`”), or due to being constrained to a single value along that axis (“`fixed`”).

`CoordinateComponent` elements of `coordinateType` “`normal`” must have a `boundaryMin` and `boundaryMax` child. `CoordinateComponent` elements of `coordinateType` “`symmetrical`” may have a `boundaryMin` and `boundaryMax` children for visualization purposes, but they are optional. `CoordinateComponent` elements of `coordinateType` “`fixed`” must have a single `boundaryFixed` child.



Lucian: New text in place–check to make sure this makes sense. Also, I named the two types ‘`coordinateType`’ and

'componentType'. Not sure if that works or not, but it's cute ;-)

3.15.4 The **sbmlUnit** attribute

The unit of a **CoordinateComponent** is represented by the **sbmlUnit** attribute, of type **UnitSidRef**. If not specified, the unit of a **CoordinateComponent** inherits from the **lengthUnits** attribute of the **Model** object, and if that in turn is not specified, the **CoordinateComponent** units cannot be determined. Many coordinates may have different units from the **lengthUnits** of the **Model**, since this is a distance metric and not always the same as the unit of each individual coordinate. For example, in Polar coordinates, the coordinate component 'theta' may be described in radians while component 'rho' may be in microns.

3.16 The **Boundary** class

The minimum and the maximum for a **CoordinateComponent** represent the bounds in each coordinate. For example, for three dimensional Cartesian coordinate system with x, y, and z coordinates, the minimum and maximum limits for each coordinates define planes orthogonal to each coordinate axis and passing through the minimum or maximum. If max-min is the same for each x,y,z then the bounds on the geometry is a cube. The **Boundary** class interacts with the **BoundaryCondition** class, allowing modelers to define For species defined within volumes adjacent to these surfaces, **BoundaryCondition** elements must be introduced. Independent of the mathematical framework, the boundary conditions can be described as independently specified molecular sources/sinks (if molecular flux is given) or as infinite pools of molecules (if concentration is given).

 Lucian: Just a note to coordinate the text here to sync with the text in the **BoundaryCondition** class, if that changes.

The minimum limit of a **CoordinateComponent** is represented by the **boundaryMin** object and the maximum limit is represented by the **boundaryMax** object, and apply to **CoordinateComponent** elements of coordinateType "normal" and "symmetrical" (and are optional in the latter case only). For **CoordinateComponent** elements of coordinateType "fixed", a single **boundaryFixed** object defines the value at which the coordinate component is fixed. All three are **Boundary** objects, and have the following attributes.

3.16.1 The **spatialId** attribute

The **spatialId** attribute of the **Boundary** object identifies the object. The attribute is required and is of type **SpId**. This attribute is used when specifying the **BoundaryCondition** for a species as an extension of an SBML core **Parameter**. When referenced (for example, via a **SpatialSymbolReference** extension of the SBML core **Parameter** object) and used within a mathematical expression, it represents the value of the **Boundary**. The units are the same as its parent **CoordinateComponent**, and are not set separately.

3.16.2 The **value** attribute

The **value** attribute is of type **double**. In a **boundaryMin** object, it represents the minimum limit of the **CoordinateComponent**. In a **boundaryMax** object, it represents the maximum limit of the **CoordinateComponent**. In a **boundaryFixed** object, it represents the fixed level of the **CoordinateComponent**.

If connected to a **Parameter** via a **SpatialSymbolReference**, this **value** may be overridden by an **InitialAssignment**. It is theoretically possible to have this value change in time through the use of a **Rule** or **Event**, but some (if not all) software tools may not support this setup.

 Lucian: Again, new text in place.

3.17 The **DomainType** class

A **DomainType** is a class of domains that are identified as being anatomically and physiologically similar. For example, a **DomainType** "cytosol" may be defined in a **Geometry** as identifying the structure and function of the cell interior. If there is one cell, then there is one domain, if there are multiple cells, then there are multiple disjoint domains ("cytosol1", "cytosol2", etc.) identified with the **DomainType** "cytosol". **CompartmentMappings**, defined as

an extension to an SBML core **Compartment**, map compartments to domain types such that each corresponding domain is assigned the same biological and mathematical function. Figure 8 shows the **DomainType** object.

Each SBML **Compartment** maps to a single **DomainType**, meaning that the initial condition of each **Species** in each **Compartment** will be the same across all **Domains** that map to a given **DomainType**. If those **Species** are spatially distributed, they will subsequently evolve independently from each other. However, if modeling two **Domains** that are similar but whose **Species** have different initial conditions, those **Domains** should be modeled as separate **DomainTypes**.

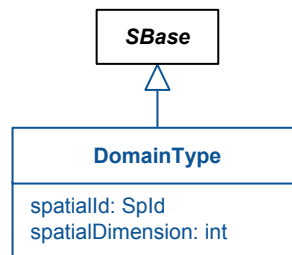


Figure 8: The **DomainType** object. One or more instances of **DomainType** in a **ListOfDomainTypes** instance can be present in a **Geometry** object.

3.17.1 The **spatialId** attribute

Each **DomainType** is identified with a **spatialId** of type **SpId**. When referenced and used within a mathematical expression, it represents the sum of the sizes of all domains associated with this **DomainType**.

As a derived quantity, if connected to a **Parameter** via a **SpatialSymbolReference**, this value may *not* be overridden by an **InitialAssignment**, nor by the use of a **Rule** or **Event**. Its value is always connected to the size of its component **Domains** instead. The units of a **DomainType** are the units of the corresponding base units of the SBML **Model** for length (for one-dimensional domains), area (for two-dimensional domains), or volume (for three-dimensional domains). It is required to define the corresponding base units for every **DomainType** in the **Model**.

3.17.2 The **spatialDimension** attribute

The **spatialDimension** attribute of the **DomainType** is of type **int** and can take on a value of 0, 1, 2, or 3. The spatial dimension is specified for a **DomainType**, rather than being repeated for each **Domain** that is represented by the **DomainType**.



Lucian: The equivalent attribute of a **Compartment** is **spatialDimensions**, i.e. it has an 's' on the end. Can we change this to match?

3.18 The **Domain** class

Domains represent contiguous regions identified by the same **DomainType**. One, two and three dimensional domains are contiguous linear regions, surface regions, and volume regions (respectively), bounded by the limits of the coordinate system (e.g. min/max of x,y,z) and adjacent domains corresponding to different domain types. **Domain** is shown in Figure 9 on the next page.

3.18.1 The **spatialId** attribute

A **Domain** is identified with a **spatialID** attribute of type **SpId**. This **spatialId** may be used within a **SpatialSymbolReference** object that is extended from an SBML core **Parameter** and can be used in an expression. When referenced, the domain evaluates to the absolute size of that domain as used by the simulator (the meshed size).

As a derived quantity, if connected to a **Parameter** via a **SpatialSymbolReference**, this value may *not* be overridden by

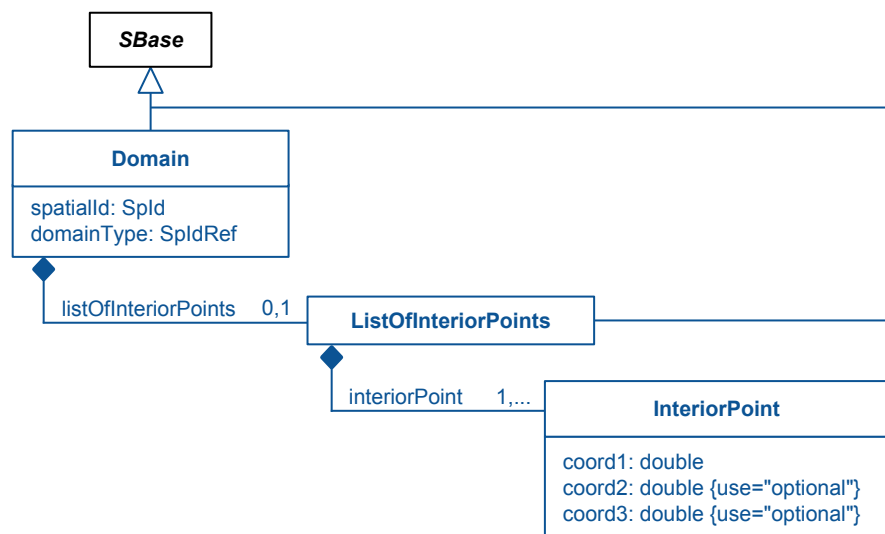


Figure 9: The definition of the **Domain**, **ListOfInteriorPoints**, and **InteriorPoint** classes. A **ListOfDomains** instance in **Geometry** can contain one or more **Domain** object instances.

an **InitialAssignment**, nor by the use of a **Rule** or **Event**. Its value is always connected to the size of the corresponding **Geometry** instead. The units of the **Domain** are the same as the units of the corresponding **DomainType**.

3.18.2 The domainType attribute

The **domainType** attribute refers to the **spatialId** of the **DomainType** that describes the anatomy and physiology of this domain. The attribute is of type **SpIdRef**. It is through this association that compartments, and hence the whole SBML model, gets mapped to the individual domains.

3.19 The InteriorPoint class

Each **Domain** can contain a **ListOfInteriorPoints**. The list of spatial points for a domain is interior to that domain. This list is optional for a **Domain** if it is the only **Domain** defined for its **DomainType**, but is required otherwise.

For those geometric descriptions that can describe multiple disjoint domains belonging to the same **domainType**, these interior points allow unambiguous identification of each domain. Formally, a single point would suffice, but in practice some tools (e.g. Smoldyn) require multiple points to handle non-convex volumes bounded by explicit surfaces. For discontinuous surfaces with the same **domainType**, the interior point identifies which domain is associated with which surface patch defined in the geometry definition.

Each **InteriorPoint** has three attributes: **coord1**, **coord2**, and **coord3**.

3.19.1 The coord1, coord2, and coord3 attributes

An **InteriorPoint** element represents a single point within the defined coordinate system and should be in the interior of the domain that contains it. It has three attributes, **coord1**, **coord2**, and **coord3**, of type **double**, representing the position along each of the up to three coordinate axes defined by the **CoordinateComponents** (with **index** 0, 1, and 2 respectively for a three dimensional geometry; see Table 1 on page 20).

Each **InteriorPoint** must define the same number of attributes as there are dimensions of the corresponding **Geometry** to which it belongs.

In the case of surfaces, interior points are sometimes required to make unambiguous identification of multiple surfaces (e.g multiple plasma membranes for multiple cells present in a geometry). Due to roundoff error and finite word lengths, it is difficult to find a three dimensional point that lies on a surface. In this case, the distance from the

surface will be used to provide unambiguous identification.

3.20 The **AdjacentDomains** class

AdjacentDomains (or domain adjacencies) captures the topological relationships within the **Geometry**. Consider that the **Domains** are nodes in a graph. The **AdjacentDomains** objects are the edges that specify the spatial connectivity of these nodes. Armed with the topology and the domain sizes, one can readily perform a compartmental approximation. Figure 10 shows the definition of the **AdjacentDomains** object.

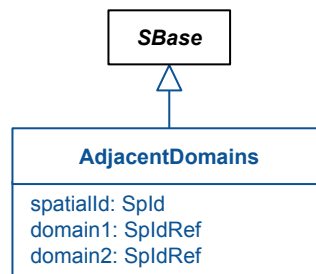


Figure 10: The definition of the **AdjacentDomains** class. **Geometry** can contain one instance of **ListOfAdjacentDomains** that can have one or more instances of **AdjacentDomains** objects.

3.20.1 The **spatialId** attribute

This attribute identifies an **AdjacentDomains** object. The attribute is of type **SpId**. It has no mathematical meaning.

3.20.2 The **domain1** and **domain2** attributes

The **domain1** and **domain2** attributes, of type **SpIdRef**, are required attributes. They are the **spatialIds** of two domains that touch each other (spatially adjacent). These are typically surface-volume contacts.

3.21 The **GeometryDefinition** class

A **Geometry** can specify a list of **GeometryDefinitions**. The **GeometryDefinition** is an abstract class that is the general term for the container which defines the concrete geometric constructs represented by the **Geometry**. Four types of **GeometryDefinitions** have been identified - **AnalyticGeometry**, **SampledFieldGeometry**, **ParametricGeometry**, **CSGeometry** (Constructed Solid Geometry) - and are elaborated in the following sections. The definition of the **GeometryDefinition** element is displayed in Figure 11 on the following page. The spatial dimension of the **GeometryDefinition** must match the **spatialDimension** of the **DomainType** defined for the associated **Domain**.

3.21.1 The **spatialId** attribute

The **spatialId** attribute that is common to all the **GeometryDefinition** types is used to uniquely identify the **GeometryDefinition**. The attribute is of type **SpId**. A **GeometryDefinition** has no mathematical meaning.

3.22 The **AnalyticGeometry** class

The **AnalyticGeometry** is a class of **GeometryDefinition** where the geometry of each domain is defined by an analytic expression. An **AnalyticGeometry** is defined as a collection of **AnalyticVolumes**, one **AnalyticVolume** for each volumetric domain in the geometry. In this representation, the surfaces are treated as the boundaries between dissimilar **AnalyticVolumes**. The **AnalyticGeometry** object contains a **ListOfAnalyticVolumes**. Figure 12 on the next page shows the definition of the **AnalyticGeometry** object.

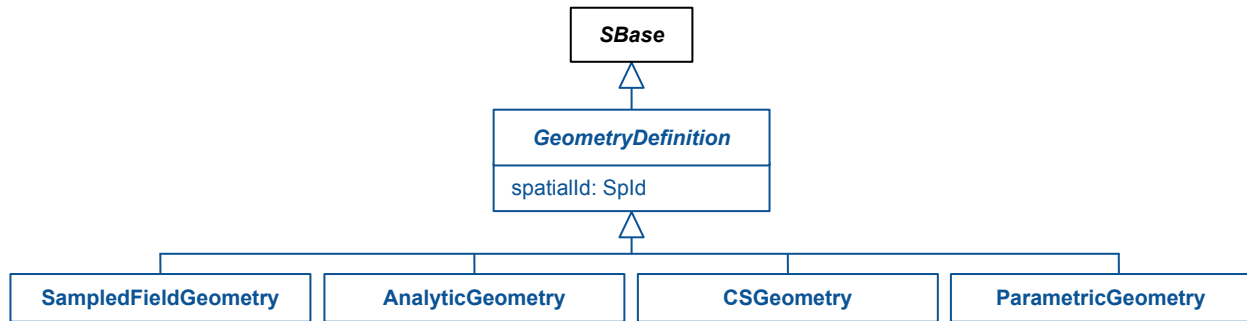


Figure 11: The **GeometryDefinition** element. **Geometry** contains one instance of **listOfGeometryDefinitions** that can contain one or more instances of **GeometryDefinition** (one of **AnalyticGeometry**, **SampledFieldGeometry**, **CSGeometry**, **ParametricGeometry**, defined below).

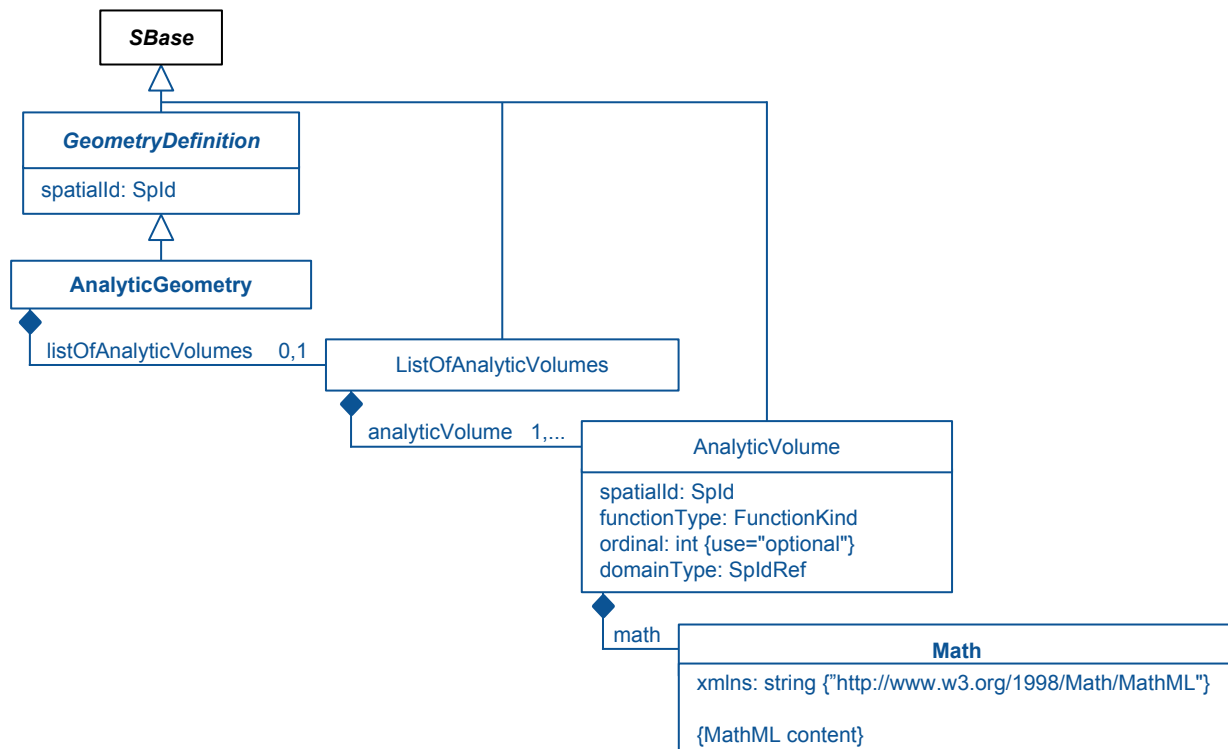


Figure 12: The definition of the **AnalyticGeometry**, **ListOfAnalyticVolumes**, and **AnalyticVolume** classes.

3.23 The **AnalyticVolume** class

The **AnalyticVolume** is used to specify the analytic expression of a volumetric (3-dimensional) domain. The analytic expression for the **AnalyticVolume** is defined in the **Math** element.

3.23.1 The **spatialId** attribute

The **spatialId** attribute uniquely identifies the **AnalyticVolume**. The attribute is required and is of type **SpId**. It has no mathematical meaning.

3.23.2 The `functionType` attribute

The `functionType` attribute is of type `FunctionKind` and is currently limited to “layered” (to be renamed), or “R-function”. A “layered” function type implies that the Math child element contains an inequality in the spatial dimensions (e.g. x,y,z) such that evaluation to “true” indicates that the point (x,y,z) is within that shape, and “false” indicates that it is not covered by that shape. The “R-function” `functionType` indicates that the shape is represented by a real-valued function whose sign indicates coverage by the shape.

 **Lucian:** If you want to rename 'layered', now is the time to do it ;-)

3.23.3 The `domainType` attribute

The `domainType` attribute of type `SpIdRef` is a required attribute. It represents the `spatialId` of the `DomainType` of the `Domain` that is represented by this `AnalyticVolume`.

3.23.4 The `ordinal` attribute

The `ordinal` attribute of type `int` is an optional attribute. It is used to represent the order of the `AnalyticVolume`. The `ordinal` is useful while reconstructing the geometry in the specific software tool - it represents the order in which the `AnalyticVolumes` representing geometric domains have to be evaluated.

Rather than struggle with the task of preventing overlapping regions of space from different `AnalyticVolumes`, the `AnalyticVolumes` are to be considered to be evaluated in the reverse order of their ordinals. In this way, any `AnalyticVolumes` that have already been processed will cover those with a smaller ordinal, thus resolving any ambiguities and removing the constraint that all `AnalyticVolumes` be disjoint and cover the entire geometric domain. The `AnalyticVolume` with `ordinal` 0 can be the “background” layer (typically the extracellular space).

No two `AnalyticVolumes` may have the same `ordinal` value, even if they should not overlap, because some tools may not calculate the geometries to the same level of precision as other tools, and may end up with overlap due to rounding errors, and will still need to resolve the ambiguity for their own purposes. Any `AnalyticVolume` without an `ordinal` value should also not overlap with any other `AnalyticVolume`, but if it does, the software tool may resolve the situation however it sees fit.

3.24 The Math class

The Math element is a required element for an `AnalyticVolume`. The Math element contains a MathML expression that defines the analytic expression for the `AnalyticVolume` referencing the coordinate components that are specified in the `ListOfCoordinateComponents` in the `Geometry`, according to the `functionType`.

3.25 The `SampledFieldGeometry` class

`SampledFieldGeometry` is a type of `GeometryDefinition` that defines a sampled image-based geometry or a geometry based on samples from a level set. `SampledFieldGeometry` is defined using a `SampledField` that specifies the sampled image and a list of `SampledVolumes` that represent the volumetric domains as sampled image regions. Figure 13 on the following page shows the definition of the `SampledFieldGeometry` object. It may be used for geometries of any dimension.

3.26 The `SampledVolume` class

A `SampledVolume` represents an interval of the sampled field that constitutes one or more contiguous regions. A `SampledVolume` is defined for each volumetric (3-dimensional) `Domain` in the `Geometry`. It has the following attributes.

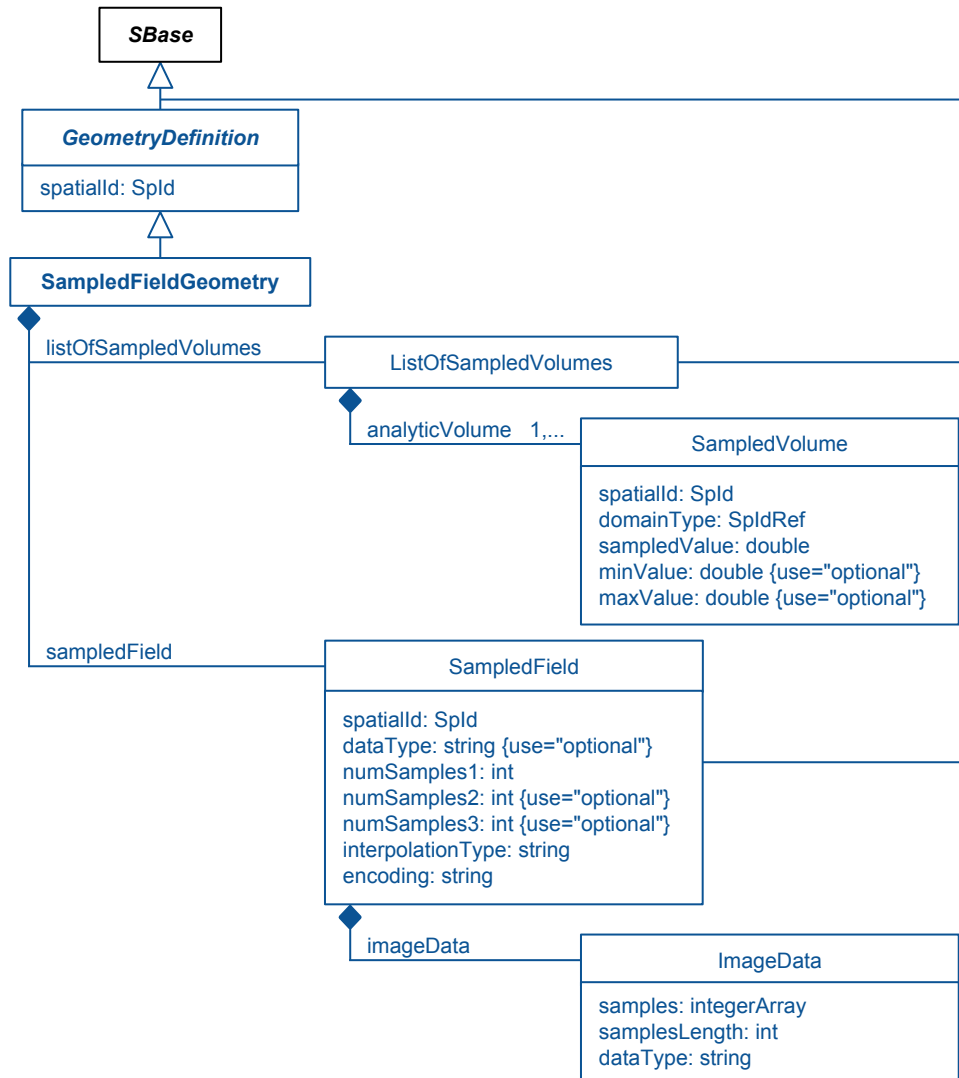


Figure 13: The definition of the **SampledFieldGeometry**, **ListOfSampledVolumes**, **SampledVolume**, **SampledField**, and **ImageData** classes.

3.26.1 The spatialId attribute

The **spatialId** attribute identifies a **SampledVolume** object. The attribute is of type **SpId** and is required when specifying a **SampledVolume**. It has no mathematical meaning.

3.26.2 The domainType attribute

The required **domainType** attribute is of type **SpIdRef**. It is the **spatialId** of the **DomainType** that represents this class of anatomical features. If there are more than one contiguous regions, then more than one domain will be defined corresponding to each **SampledVolume**.

3.26.3 The sampledValue attribute

The required **sampledValue** attribute is of type **double**. It represents the pixel value of a **SampledVolume**.

3.26.4 The **minValue** attribute

The optional **minValue** attribute is of type **double**. It represents the minimum of the pixel value (**sampledValue**) range.

3.26.5 The **maxValue** attribute

The optional **maxValue** attribute is of type **double**. It represents the maximum of the pixel value (**sampledValue**) range.

3.27 The **SampledField** class

A **SampledField** defined in a **SampledFieldGeometry** is a sampled scalar field such as an image or samples from a level set. Currently, the attributes of **SampledField** represent the specification of an image dataset (the number of samples in x, y, z coordinates, data type of the image representation, etc.) and the **ImageData** element of the **SampledField** specifies the actual image as integer sampled data.

3.27.1 The **spatialId** attribute


The **spatialId** attribute identifies a **SampledField**. It is of type **SpId** and is a required attribute. It has no mathematical meaning.

3.27.2 The **numSamples1**, **numSamples2**, **numSamples3** attributes

The **numSamples1**, **numSamples2**, and **numSamples3** attributes represent the number of samples in each of the coordinate components. (e.g. **numX**, **numY**, **numZ**) in an image dataset. These attributes are of type **int** and are required to specify the **SampledField**. The samples are assumed to be uniformly sampled.


3.27.3 The **dataType** attribute

This attribute represents the data type of each sample (e.g. **int32**). The attribute is of type **string** and is optional.

 **Lucian:** This attribute seems a bit arbitrary, and unhelpful unless there's a standard list of possible values here. Is there one? If not, I say we drop the attribute entirely.


3.27.4 The **interpolationType** attribute

The **interpolationType** attribute is an optional attribute of type **string** and represents the interpolation type of the sampled data and is defined as "constant" for zeroth order, "linear" for first order, etc.

 **Lucian:** Again, the list of possible values here needs to be explicitly stated. If it really can be any arbitrary order, switching this to a non-negative integer would make more sense.

3.27.5 The **encoding** attribute

The **encoding** attribute is an optional attribute of type **string**. It is used to specify text encoding and compression.

 **Lucian:** Again, the list of possible values here needs to be explicitly stated, if it makes any difference to the model. Unless you really want people to be able to put in 'standard zip' or 'rot13' as their encoding type.

3.28 The **ImageData** class

The **ImageData** element represents the actual image data of the image-based geometry as encoded samples defined by the encoding and the data type. The **ImageData** element has the following attributes.

3.28.1 The **samples** attribute

The **samples** attribute is of type **integerArray**, whose values represent the image pixel values. This attribute is required.

3.28.2 The **samplesLength** attribute

The **samplesLength** attribute is of type **int** and is required. It represents the array length of the **samples** attribute (number of values in the image data array).

3.28.3 The **dataType** attribute

The **dataType** attribute is of type **string** and is required. Presently, this attribute indicates if the image data is 'compressed' or 'uncompressed'. **Anu:** (This attribute and the encoding attribute of **SampledField** have to be revisited.)

Lucian: I agree!

3.29 The **CSGeometry** class

CSGeometry (Constructed Solid Geometry) is a type of **GeometryDefinition** that defines a combined, solid, volumetric object from a number of primitive solid volumes by the application of set operations such as union, intersection and difference and affine transformations such as rotation, scaling, translation, etc. The **CSGeometry** element is defined by a **listOfCSGObjects** element that contains a collection of **CSGObjects**. Figure 14 shows the definition of the **CSGeometry** object.

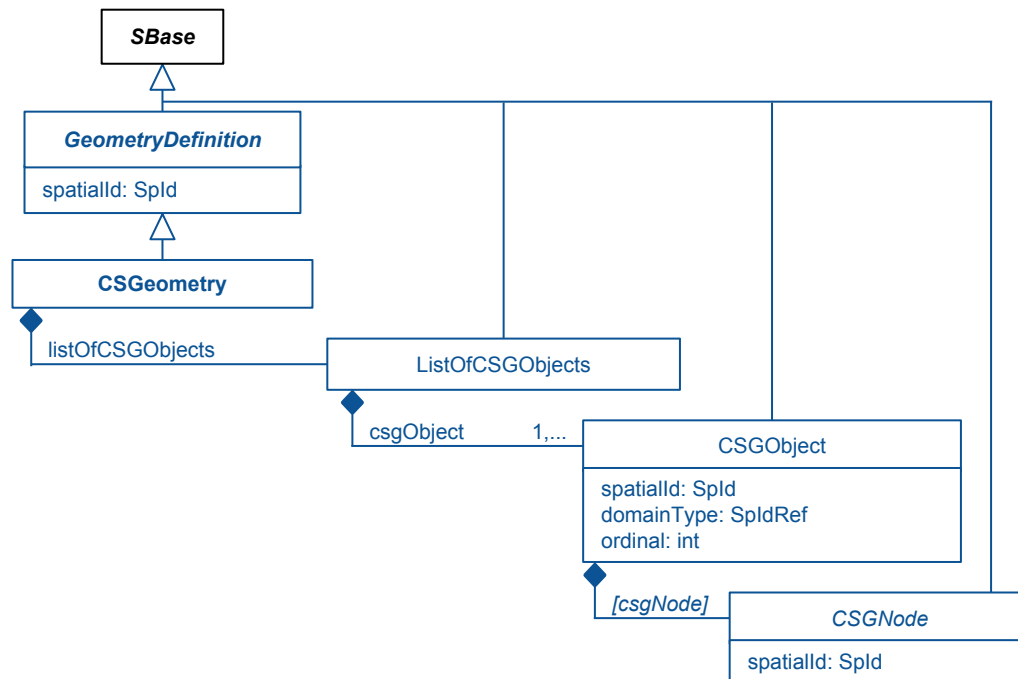


Figure 14: The definition of the **CSGeometry**, **ListOfCSGObjects**, and **CSGObject** classes.

3.30 The `CSGObject` class

Each `CSGObject` is a scene graph representing a particular geometric object using constructed solid geometry. A node in a tree (scene graph) is made up of `CSGPrimitives`, `CSGSetOperators`, and `CSGTransformations`. Note that the `CSGPrimitives` are always leaves in this tree. The `CSGObject` is analogous to an `AnalyticVolume` element in the sense that it is a constructed geometry (from primitives) used to specify a volumetric (3-dimensional) domain. The `CSGObject` element has three attributes: `spatialId`, `domain` and `ordinal`. The definition of the `CSGObject` is completed by defining a `CSGNode` which is the root of the `CSGObject` scene graph.


3.30.1 The `spatialId` attribute

The `spatialId` attribute uniquely identifies the `CSGObject` element. The attribute is required and is of type `SpId`.

 Lucian: Does this element have mathematical meaning?

3.30.2 The `domain` attribute


The `domain` attribute is of type `SpIdRef` and is a required attribute. It is a reference to the `spatialId` of the `Domain` that this `CSGObject` represents.

 Lucian: Am I correct in assuming that this information is redundant with the corresponding `Domain`'s interior points list? Is this information necessarily redundant? Is that OK? I'm going to assume that the following is a reasonable restriction:

All `InteriorPoints` of the corresponding `Domain` must be points inside the geometry this `CSGObject` describes.

3.30.3 The `ordinal` attribute

The `ordinal` attribute of type `int` is an optional attribute. It is used to represent the order of the `CSGObject`. The `ordinal` is useful while reconstructing the geometry in the specific software tool - it represents the order in which the `CSGObjects` representing geometric domains have to be placed.

 Lucian: We should probably be as explicit here as we were with the similar “`ordinal`” attribute of the `AnalyticVolume` class. Can I assume that the same rules apply and copy the text here? One oddness is that the “`ordinal`” attribute of the `AnalyticVolume` class is optional, while here it is required. If they do the same thing, we should probably either make them both optional, or both required.

3.30.4 The `[csgNode]` child

The child `[csgNode]` element represents the geometry that is to be linked to the `domainType` of the `CSGObject`. Note that the child of the `CSGObject` element is not called “`csgNode`” but rather takes the name of the derived class, decapitalized. Thus, a `CSGObject` may have a `csgPrimitive`, `csgPseudoPrimitive`, `csgSetOperator`, `csgTranslation`, `csgRotation`, `csgScale`, or `csgHomogeneousTransformation` child.

3.31 The `CSGNode` class

The operators and operands used to construct a constructed solid geometry are generalized as a `CSGNode`, defined in Figure 15 on the following page as an abstract base class. The classes that inherit from `CSGNode` can be one of the following: `CSGSetOperator`, `CSGTransformation` (operators; itself another abstract base class), `CSGPrimitive`, or `CSGPseudoPrimitive` (operands). The `CSGNode` has one attribute: `spatialId`. The `CSGObject` contains a `CSGNode` object which is the root of the `CSGObject` scene graph (representing one constructed solid geometry domain).

3.31.1 The `spatialId` attribute

The `spatialId` attribute uniquely identifies the `CSGNode` element. The attribute is required and is of type `SpId`.

 Lucian: Do some of these classes have mathematical meaning?

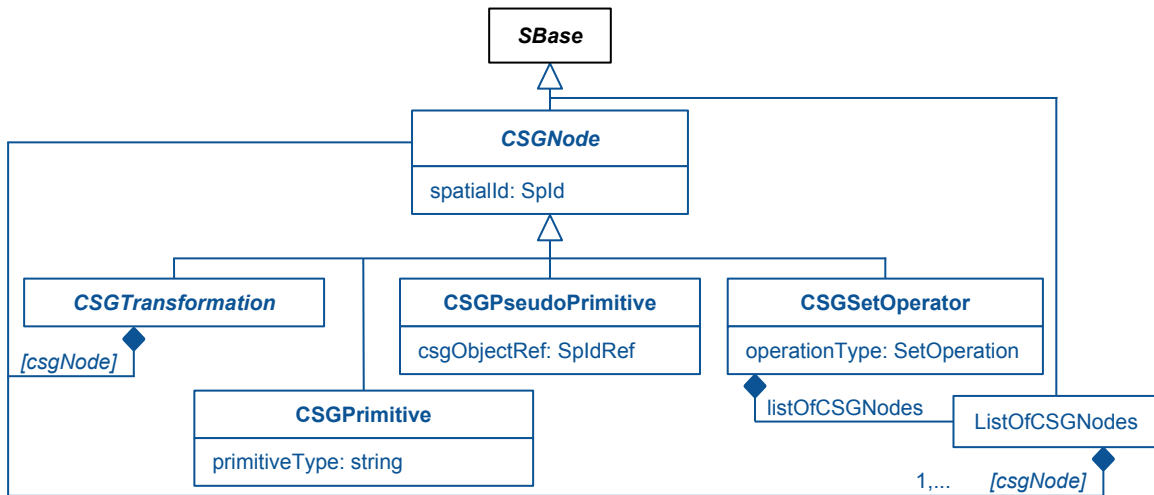


Figure 15: The definition of the abstract base class **CSGNode**, and its subclasses **CSGPrimitive**, **CSGPseudoPrimitive**, and **CSGSetOperator**. The abstract base class **CSGTransformation** (also a subclass of **CSGNode**) is defined in Section 3.36 on the following page.

3.32 The **CSGPrimitive** class

CSGPrimitive element represents the primitive geometric shapes that can be represented by the **CSGeometry**. Some of the primitive shapes that can be used are "sphere", "cylinder", "cube", "cone", etc. These shapes are defined with a predefined orientation and fitting within the unit cube (+/- 1 in x, y, and z). This element has one required attribute: **primitiveType** of type **string**.

3.32.1 The **primitiveType** attribute

The **primitiveType** attribute is a required attribute that is of type **string**. It represents one of the predefined primitive shapes.

 **Lucian:** This list absolutely needs to be included in the spec!

3.33 The **CSGPseudoPrimitive** class

CSGPseudoPrimitive element is used to reference a pre-defined **CSGObject** object while defining a **CSGObject** (geometric domain). This allows the re-use of one constructed **CSGObject** in another. It has one attribute of type **SpIdRef**.

3.33.1 The **csgObjectRef** attribute


The **csgObjectRef** attribute identifies a pre-defined **CSGObject** in the **CSGeometry**. The attribute is required and is of type **SpId**. A **CSGObject** may not reference itself, nor its parent, nor its parent's parent, etc.

3.34 The **CSGSetOperator** class

The **CSGSetOperator** element represents the set operations (union, intersection, difference) that can be performed on a set of primitive geometric shapes (**CSGPrimitives**) or on a set of **CSGNodes** (a transformation or set operation on one or a set of **CSGPrimitives**). This element has one attribute of type **string**. It also contains a required child **ListOfCSGNodes** that represents the set of nodes on which the set operation is performed.

3.34.1 The `operationType` attribute

The `operationType` attribute is of type `SetOperation` and represents an operation that can be performed on a set of `CSGNodes`. The possible values that the `operationType` attribute can take are “union”, “intersection” or “difference”.

 Lucian: 'union' and 'intersection' are unordered, but 'difference' is order-dependent, which is tricky to accomplish in the order-agnostic XML of SBML. Either you have to say that order is important in this one case (something we've tried to avoid everywhere else), or we add an 'order' attribute somewhere, or we drop 'difference' as a possible set operator. Also, according to wikipedia, the more common term is 'complement', with 'relative complement' and 'absolute complement' both meaning different things.

3.35 The `ListOfCSGNodes` class


The `ListOfCSGNodes` must contain one or more `csgNode` children that are to be combined according to the set operation of the parent `CSGSetOperator`. While having a single child is legal, this is semantically equivalent to simply putting that child in the model instead of the `CSGSetOperator`, and therefore has limited modeling benefit. Note that the children of the `ListOfCSGNodes` object are not called `csgNode` but rather take the name of the derived class, decapitalized. Thus, they may be called `csgPrimitive`, `csgPseudoPrimitive`, `csgSetOperator`, `csgTranslation`, `csgRotation`, `csgScale`, or `csgHomogeneousTransformation`.

3.36 The `CSGTransformation` class

The `CSGTransformation` represents a generalization for the type of transformation that can be performed on a primitive geometric shape (`CSGPrimitive`) or on a `CSGNode` (a transformation or set operation on one or a set of `CSGPrimitives`). The types of possible transformations are 'rotation', 'translation', 'scaling', and 'homogeneous transformation', defined below. The `CSGTransformation` element contains a `CSGNode` element upon which the transformation is performed. It also has one attribute : `transformationType` of type `string`.

3.36.1 The `transformationType` attribute

The `transformationType` attribute is of type `string` and is a required attribute. It represents the type of transformation ('rotation', 'translation', 'scaling', and 'homogeneous transformation').

 Lucian: This attribute didn't show up in the UML diagrams, and is redundant if the names of the elements are 'csgRotation', 'csgTranslation', etc. I would strongly recommend the naming approach, as you've done this already, and I in fact already assumed it would be done this way. If not... can we change it? I dunno.

3.36.2 The `csgNode` child

The child `csgNode` element represents the geometry that is to be transformed by the `CSGTransformation` element. Note that this child is not called `csgNode` but rather takes the name of the derived class, decapitalized. Thus, it may be called `csgPrimitive`, `csgPseudoPrimitive`, `csgSetOperator`, `csgTranslation`, `csgRotation`, `csgScale`, or `csgHomogeneousTransformation`.

3.37 The `CSGTranslation` class

The `CSGTranslation` element represents a translation transformation on a `CSGNode` (a transformation or set operation on one or a set of `CSGPrimitives`) or a `CSGPrimitive` along the axes defined in the `Geometry`. This element has 3 attributes:

3.37.1 The `translateX` attribute

The `translateX` attribute is of type `double`. It represents the translation of the `CSGNode` along the x-axis (or first `CoordinateComponent` defined in `Geometry`).

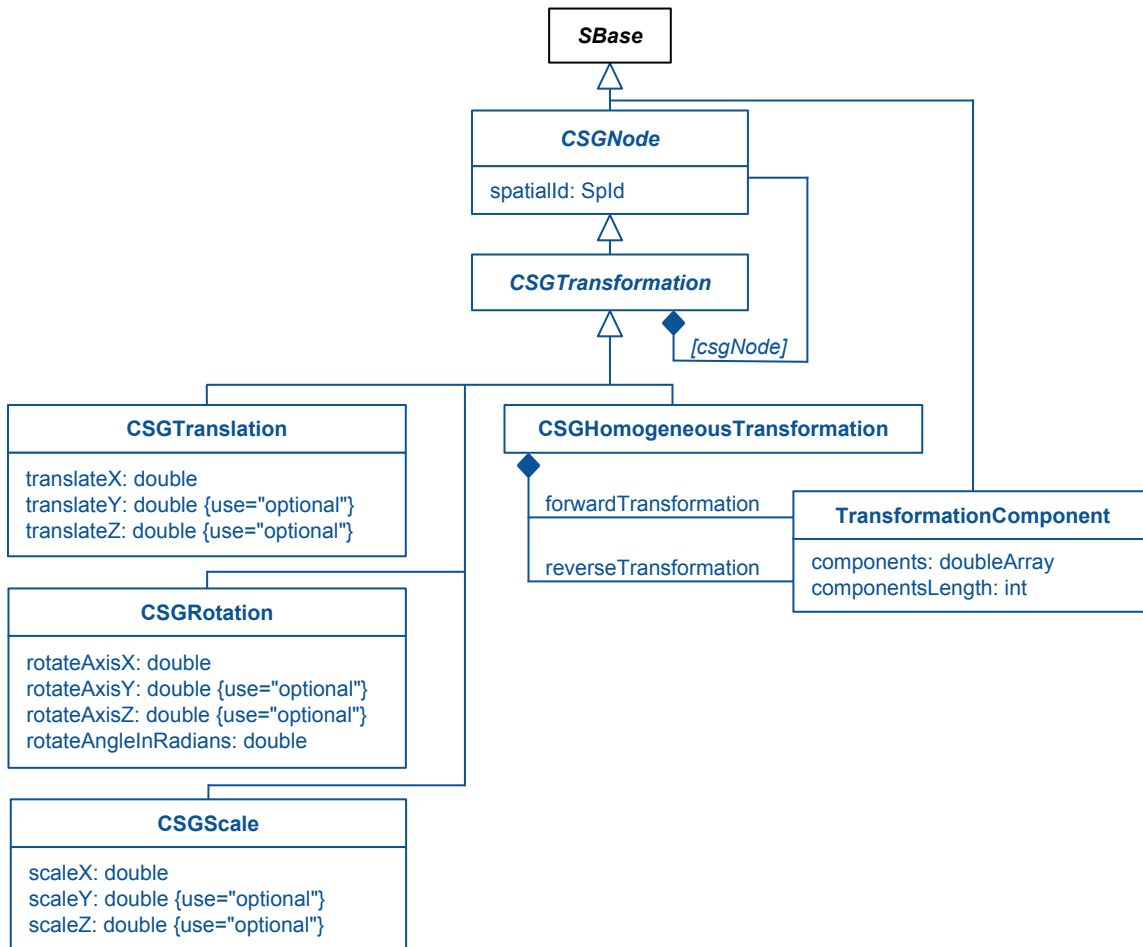


Figure 16: The definition of the abstract base class **CSGTransformation**, its subclasses, **CSGRotation**, **CSGScale**, and **CSGHomogeneousTransformation**, and the **TransformationComponent** class.

3.37.2 The `translateY` attribute

The `translateY` attribute is of type `double`. It represents the translation of the **CSGNode** along the y-axis (or second **CoordinateComponent** defined in **Geometry**).

3.37.3 The `translateZ` attribute

The `translateZ` attribute is of type `double`. It represents the translation of the **CSGNode** along the z-axis (or third **CoordinateComponent** defined in **Geometry**).




Lucian: There might not be any 'x', 'y', or 'z' axes at all, if the modeler is using some other coordinate system. I would recommend a different approach where instead of having 'translate*' attributes, you instead have 'translate' children with two attributes, one an `SpIdRef` to the axis in question, and one the double that's the amount to be translated.

3.38 The **CSGRotation** class

The **CSGRotation** element represents a rotation transformation on a **CSGNode** (a transformation or set operation on one or a set of **CSGPrimitives**) or a **CSGPrimitive** about the axes defined in the **Geometry**. This element has 4 attributes:

3.38.1 The rotateX attribute

The **rotateX** attribute is of type **double**. It represents the rotation of the **CSGNode** along the x-axis (or first **CoordinateComponent** defined in **Geometry**).

 Lucian: I don't know what 'the rotation' means, since there's already an angle attribute. Are we talking relative rotation of that angle on this axis? I think we need more semantics here.

3.38.2 The rotateY attribute

The **rotateY** attribute is of type **double**. It represents the rotation of the **CSGNode** along the y-axis (or second **CoordinateComponent** defined in **Geometry**).

3.38.3 The rotateZ attribute

The **rotateZ** attribute is of type **double**. It represents the rotation of the **CSGNode** along the z-axis (or third **CoordinateComponent** defined in **Geometry**).


 Lucian: Again, I think something should be written here for when the geometries are defined with different axes.

3.38.4 The rotationAngleInRadians attribute

The **rotationAngleInRadians** attribute is of type **double**. It represents the rotation angle of the **CSGNode**, in radians, along the defined axis.

3.39 The CSGScale class

The **CSGScale** element represents a scale transformation on a **CSGNode** (a transformation or set operation on one or a set of **CSGPrimitives**) or a **CSGPrimitive** along the axes defined in the **Geometry**. This element has 3 attributes:

 Lucian: Does it make a difference if the object is scaled and remains centered where it was, or leaves its corner at the origin? If so, the version used should be explicitly stated here, I think.

3.39.1 The scaleX attribute

The **scaleX** attribute is of type **double**. It represents the amount of scaling of the **CSGNode** along the x-axis (or first **CoordinateComponent** defined in **Geometry**).

3.39.2 The scaleY attribute

The **scaleY** attribute is of type **double**. It represents the amount of scaling of the **CSGNode** along the y-axis (or second **CoordinateComponent** defined in **Geometry**).


3.39.3 The scaleZ attribute

The **scaleZ** attribute is of type **double**. It represents the amount of scaling of the **CSGNode** along the z-axis (or third **CoordinateComponent** defined in **Geometry**).

 Lucian: And again: something should be written here about how to do this when you have other axes than cartesian.

3.40 The CSGHomogeneousTransformation class

The **CSGHomogeneousTransformation** element represents a homogeneous transformation on a **CSGNode**: a transformation or set operation on one or more **CSGPrimitives**. This element contains two **TransformationComponent** elements: a **forwardTransformation** and a **reverseTransformation**, both of type **TransformationComponent**.

 Lucian: The difference between the forward and reverse transformations needs to be explained here, as well as why you need both.

3.41 The `TransformationComponent` class

The `TransformationComponent` element represents an affine transformation that can be applied to a `CSGNode`. This element has the following two attributes:

3.41.1 The `components` attribute

The `components` attribute is of type `doubleArray`, whose values represent the affine transformation. This attribute is required.

3.41.2 The `componentsLength` attribute

The `componentsLength` attribute is of type `int` and is required. It represents the array length of the `components` attribute (number of values in the `components` array).

3.42 The `ParametricGeometry` class

`ParametricGeometry` is a type of `GeometryDefinition` that parametrically defines geometric structures/domains. The `ParametricGeometry` element is defined with a `listOfObjects` that is a collection of `ParametricObjects` and a `listOfSpatialPoints` that is a collection of `SpatialPoints`. Figure 17 on the next page shows the definition of the `ParametricGeometry` object.

3.43 The `ParametricObject` class

The `ParametricObject` element represents a parametric geometry object.

3.43.1 The `spatialId` attribute

The `spatialId` attribute is a required attribute of type `SpId`. It uniquely identifies the `ParametricObject` element.

 **Lucian:** Does this element have mathematical meaning?


3.43.2 The `polygonType` attribute

The `polygonType` attribute is of type `string` and is a required attribute. It represents the type of polygon that describes the `ParametricObject`.

 **Lucian:** What options are there for this attribute?

3.43.3 The `domain` attribute


The `domain` attribute is of type `SpIdRef` and is a required attribute. It is a reference to the `spatialId` of the domain that this `ParametricObject` represents.

 **Lucian:** As above: Am I correct in assuming that this information is redundant with the corresponding `Domain`'s interior points list? Is this information necessarily redundant? Is that OK? I'm going to assume that the following is a reasonable restriction:

All `InteriorPoints` of the corresponding `Domain` must be points inside the geometry this `ParametricObject` describes.

3.44 The `PolygonObject` class

The `polygonObject` represents an ordered list of indices that refer to elements in the `SpatialPoints` array and are interpreted by considering the `polygonType` attribute of the `ParametricObject`. For reference, we could consider using the VTK polydata conversions for shapes and vertex orderings.

 **Lucian:** Wait, 'consider'? Are there other options? If so, how would the modeler know that a different conversion was being followed? Or does it not matter?

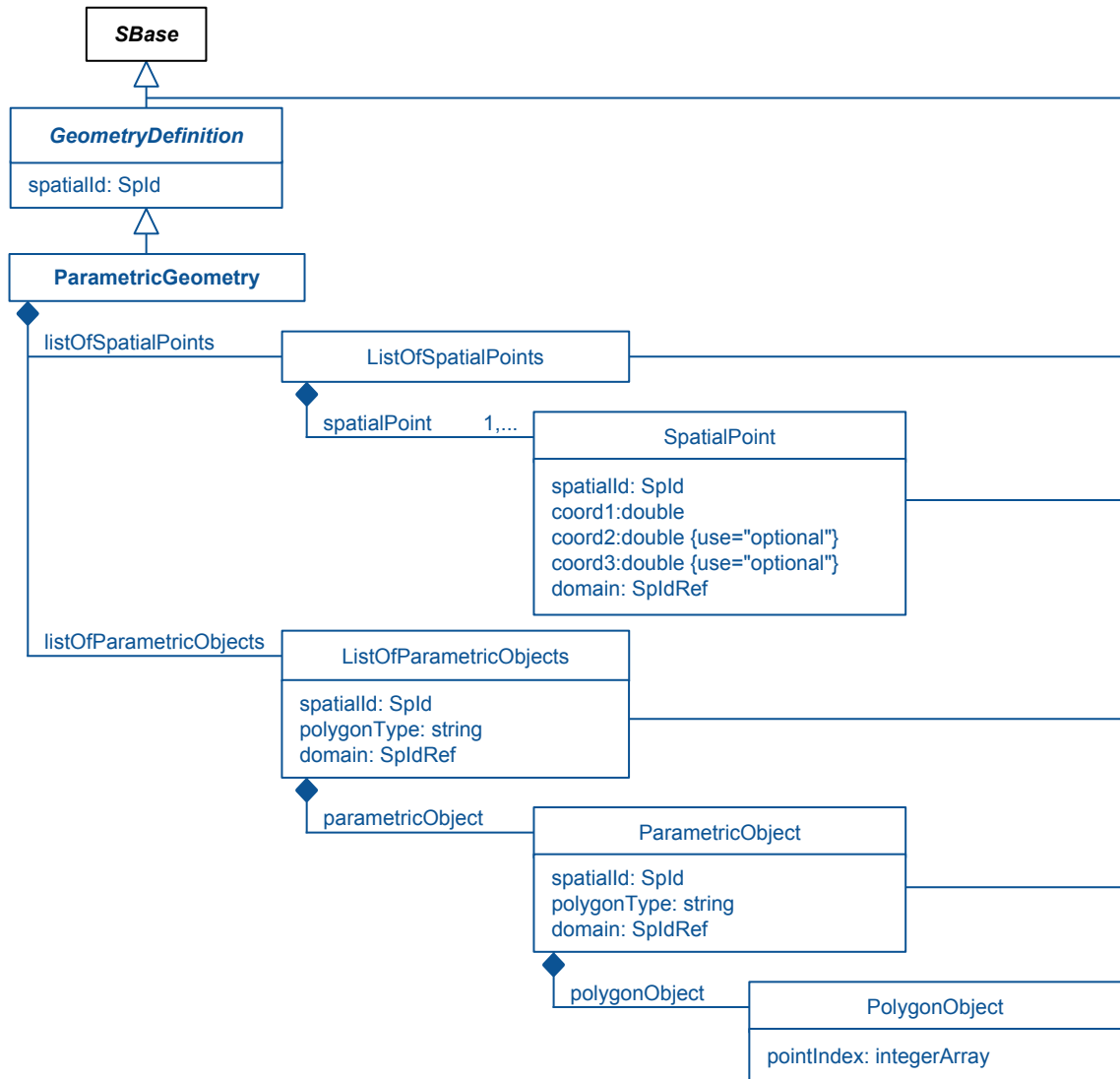


Figure 17: The definition of the **ParametricGeometry**, **ListOfSpatialPoints**, **SpatialPoint**, **ListOfParametricObjects**, **ParametricObject**, and **PolygonObject** classes.

3.44.1 The `pointIndex` attribute

The `pointIndex` attribute is an array of integers holding the indices of the points in the `SpatialPoints` array.

3.45 The **SpatialPoint** class

The **SpatialPoint** element represents a point used as a vertex in the **ParametricGeometry**.

3.45.1 The `spatialId` attribute

The `spatialId` element uniquely identifies a **SpatialPoint** element. It is a required attribute and is of type `SpId`.



Lucian: Does this element have mathematical meaning?

3.45.2 The **coord1**, **coord2**, and **coord3** attributes

The **coord1**, **coord2**, **coord3** attributes are of type **double**. They represent the 3-dimensional coordinate of the **SpatialPoint**. Depending on the dimension of the **Geometry**, one, two or all the three attributes are required.

3.45.3 The **domain** attribute

The **domain** attribute is of type **SpIdRef** and is a required attribute. It is a reference to the **spatialId** of the domain which contains this **SpatialPoint**.



Lucian: And once more:

All **InteriorPoints** of the corresponding **Domain** must be points inside the geometry this **SpatialPoint** describes.

4 Examples

1

This section will hopefully contain examples employing the Spatial package for SBML Level 3.

2

5 Interaction with the Required Elements package

The Required Elements package is designed to create a way for a modeler to denote which specific elements of an SBML model have changed due to interactions with a package. The Spatial Processes package can change the mathematical meaning of the SBML core elements **Compartment**, **Species**, **Reaction**, and **Parameter**. If the Required Elements namespace and the Spatial Processes namespace are declared in the same SBML document, the following restrictions apply. Note that these do not apply in a document without the Required Elements namespace declared.

5.1 Compartments

When a spatial geometry is defined in the SBML model, the **Compartment** element may be extended for the spatial package to represent a spatially-defined area with particular boundaries whose mathematical value is defined as its **unitSize** multiplied by the size of the **DomainType** to which it maps. Any **Compartment** with a child **CompartmentMapping** element must therefore have a **ChangedMath** child pointing to the Spatial Processes namespace. Its **viableWithoutChange** attribute may be set to “true” if the compartment’s size is set with the **size** attribute (setting it through a **Rule** or **InitialAssignment** is illegal for the purposes of the spatial package). For example, the modeling package may precalculate the compartment size and store it in the **Compartment size** attribute. However, due to slight differences in mesh generation between simulators, the domain size actually used within computations may vary slightly from simulator to simulator, and will thus introduce small errors that may break mass conservation.

A **Compartment** without a child **CompartmentMapping** element remains unaffected by the Spatial package, and must therefore not have a **ChangedMath** child that points to the Spatial Processes namespace.

5.2 Species

Any **Species** with the **isSpatial** attribute set to “true” must have a **ChangedMath** child that points to the Spatial Processes namespace, as the model assumes a spatially-distributed level of that **Species**, instead of considering it to be a well-mixed pool. Its **viableWithoutChange** attribute may be set to “true” if the model specifies the **initialAmount** or **initialConcentration** attributes of the **Species** or if the initial condition is specified by a **Rule** or an **InitialAssignment**.

A **Species** with the **isSpatial** attribute set to “false” remains unaffected by the Spatial package, and must therefore not have a **ChangedMath** child that points to the Spatial Processes namespace.

5.3 Reactions

Any **Reaction** with an **isLocal** attribute set to “true” must have a **ChangedMath** child that points to the Spatial Processes namespace, as the model treats such a **Reaction** as defining a change in local substrate concentrations over time, instead of as a change in global substrate amounts over time. Its **viableWithoutChange** attribute will therefore almost always be set to “false”, as the units of the **KineticLaw** have been changed. However, concentration over time can be numerically identical to amount over time in **Compartments** of unit volume; in this situation, the value of that attribute may be set to “true”. However, this practice is discouraged.

A **Reaction** with the **isLocal** attribute set to “false” remains unaffected by the Spatial package, and must therefore not have a **ChangedMath** child that points to the Spatial Processes namespace.

5.4 Parameters

A **Parameter** object with a **SpatialSymbolReference** child does not take its value from its **value** attribute, but rather from the Spatial object with which it is linked. Therefore, all **Parameter** objects with a **SpatialSymbolReference** child must have a **ChangedMath** child that points to the Spatial Processes namespace. Its **viableWithoutChange** attribute may be set to “true” if the **Parameter**’s **value** is set, and/or if there is an **InitialAssignment** or **Rule** that sets

that value.

Parameter objects with **DiffusionCoefficient**, **AdvectionCoefficient**, or **BoundaryCondition** children, on the other hand, still take their values from the **value** attribute and/or other SBML Level 3 Version 1 Core elements, and remain unchanged by any Spatial construct. Therefore, these and any other **Parameter** elements without **SpatialSymbolReference** children may not be given a **ChangedMath** child that points to the Spatial Processes namespace.

5.5 General

No other SBML Level 3 Version 1 Core element is affected by the Spatial Processes package, and none may therefore not have a **ChangedMath** child that points to the Spatial Processes namespace.

A Validation of SBML documents using Spatial 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 Spatial 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 Spatial 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 Spatial 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 Spatial package.

- 🗉 For convenience and brevity, we use the shorthand “**spatial:x**” to stand for an attribute or element name **x** in the namespace for the Spatial package, using the namespace prefix **spatial**. In reality, the prefix string may be different from the literal “**spatial**” used here (and indeed, it can be any valid XML namespace prefix that the modeler or software chooses). We use “**spatial:x**” because it is shorter than to write a full explanation everywhere we refer to an attribute or element in the Spatial package namespace.

General rules about the Spatial package

- spatial-10101** ☑ To conform to Version 1 of the Spatial 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/spatial/version1>”. (References: SBML Level 3 Package Specification for Spatial, Version 1, [Section 3.2 on page 8](#).)
- spatial-10102** ☑ Wherever they appear in an SBML document, elements and attributes from the Spatial package must be declared either implicitly or explicitly to be in the XML namespace
“<http://www.sbml.org/sbml/level3/version1/spatial/version1>”. (References: SBML Level 3 Package Specification for Spatial, Version 1, [Section 3.2 on page 8](#).)

Rules for the extended SBML class

- spatial-10201** ☑ In all SBML documents using the Spatial package, the **SBML** object must include a value for the attribute **spatial:required** attribute. (References: SBML Level 3 Version 1 Core, Section 4.1.2.)
- spatial-10202** ☑ The value of attribute **spatial:required** on the **SBML** object must be of the data type **boolean**. (References: SBML Level 3 Version 1 Core, Section 4.1.2.)

spatial-10203 ✓ The value of attribute **spatial:required** on the **SBML** object must be set to “**true**” (References: SBML Level 3 Package Specification for Spatial, Version 1, [Section 3.2 on page 8.](#))

General rules about attributes

 **Lucian:** Obviously this will need to be filled out.

Acknowledgments

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

This work was funded by the National Institutes of Health (USA) under grant [Something?]

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>.