

SBML Level 3 Package: Dynamic Structures ('dyn')

Chris J. Myers
myers@ece.utah.edu
Electrical and Computer Engineering
University of Utah
Salt Lake City, UT, US

Harold F. Gómez
hgomez87@bu.edu
Bioinformatics
Boston University
Boston, MA, US

Thomas B. Kepler
tbkepler@bu.edu
Microbiology
Boston University
Boston, MA, US

Version 1 (Draft)

July 24, 2014 (0.1 α)

This is a draft specification for the package 'dyn' and not a normative document. Please send feedback to the Package Working Group mailing list at sbml-dynamic@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/dyn

This release of the specification is available at
[ToBeDecided](#)



Contents

1	Introduction	3
1.1	Proposal corresponding to this package specification	3
1.2	Package dependencies	3
1.3	Document conventions	3
2	Background	5
2.1	Problems with current SBML approaches	5
2.2	Past work on this problem or similar topics	5
3	Proposed syntax and semantics	6
3.1	Overview of dyn extension	6
3.2	Namespace URI and other declarations necessary for using this package	6
3.3	Primitive data types	6
3.3.1	Type <code>CBOTerm</code>	6
3.3.2	Type <code>CoordinateSystemKind</code>	7
3.3.3	Type <code>CoordinateKind</code>	7
3.4	The extended Event object	7
3.4.1	The <code>cboTerm</code> attribute	7
3.5	The extended EventAssignment object	8
3.5.1	The <code>component</code> attribute	8
3.6	The extended Compartment object	8
3.6.1	The <code>coordinateSystem</code> attribute	9
3.6.2	The <code>ListOfCoordinateComponents</code> class	9
3.7	The CoordinateComponent class	9
3.7.1	The <code>coordinateIndex</code> attribute	9
3.7.2	The <code>variable</code> attribute	10
4	The Cell Behavior Ontology and the <code>cboTerm</code> attribute	11
4.1	Cell Behavior Ontology (CBO)	11
4.1.1	Structure of the Cell Behavior Ontology	11
4.2	Using CBO and <code>cboTerm</code>	12
4.2.1	Supported CBO terms in Event Components	12
4.2.2	Tradeoffs in using CBO terms	13
4.2.3	Relationships to the SBML annotation element	13
4.3	Discussion	13
5	Examples	14
5.1	Cell Division on a 2D grid	14
6	Best practices	15
A	Validation Rules	16
	Acknowledgments	17
	References	18

1 Introduction

Intrinsic dynamic cellular behaviors characteristic of multicellular systems (e.g., proliferation, differentiation, endocytosis, exocytosis, and cell death) may be modeled using a variety of mathematical and spatial frameworks. However, fully representing systems that undergo discrete structural changes during simulation cannot currently be accomplished using either SBML Level 3 Version 2 Core's structural constructs or available language extensions. While mathematical encoding of dynamic behaviors can be approximated, common constructs and behavior-specific semantics are still needed to signal how static constructs interact to emulate dynamic cellular behavior.

This package proposal is completely independent of all other package proposals. Though initially envisioned as being dependent on the Arrays and Sets package since dynamic structures are usually associated with compound structures such as lists, sets, and arrays, this proposal does not use constructs from any other language extension and can be used independently. Nevertheless, to facilitate the detailed encoding multicellular systems displaying dynamic cellular events, this extension is designed to work with already existing SBML packages. For example, the Hierarchical Model Composition package could be used to encapsulate portions of a model that need to change dynamically in response to a particular cellular process. The Spatial package could also be used to either describe time-varying species concentration fields as partial differential equations to match native modeling paradigms, or to specify the geometric representation of specific cellular components.

There are a sufficiently large number of multicellular simulators modeling dynamic cellular behaviors to justify the effort of creating a dynamic structure modeling extension. It is the purpose of this SBML Level 3 Version 2 proposal to define a common representation that enables modelers to encode the initial conditions and dynamics of models that include dynamic processes regardless of the spatial or mathematical methods used for simulation.

1.1 Proposal corresponding to this package specification

NEED THE URL TO UPLOADED PROPOSAL



Harold: After proposal acceptance, all dev efforts need a current editor to be part of the package working group (dev requirement). Any ideas as to who would be most suitable /willing to take on the job?

1.2 Package dependencies

The dynamic structures package has no dependencies on other SBML Level 3 packages. It is also designed with the goal of being able to work seamlessly with other SBML Level 3 packages. This means that all extensions to existing SBML classes are defined as optional.

1.3 Document conventions

Following the precedent set by the SBML Level 3 Core specification document, we use UML 1.0 (Unified Modeling Language; Eriksson and Penker 1998; Oestereich 1999) class diagram notation to define the constructs provided by this package. We also use color in the diagrams to carry additional information for the benefit of those viewing the document on media that can display color. The following are the colors we use and what they represent:

- **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.

We also use the following typographical conventions to 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 classes that are never instantiated directly, but rather serve as parents of other object 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 generally all tokens *other* than SBML UML class names, are printed in an upright typewriter typeface. Primitive types defined by SBML begin with a capital letter; SBML also makes use of primitive types defined by XML Schema 1.0 ([Biron and Malhotra, 2000](#); [Fallside, 2000](#); [Thompson et al., 2000](#)), but unfortunately, XML Schema does not follow any capitalization convention and primitive types drawn from the XML Schema language may or may not start with a capital letter.

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

2 Background

2.1 Problems with current SBML approaches

Representation of dynamical systems that undergo discrete structural changes during simulation cannot currently be accomplished mainly because SBML's structural constructs are fixed. This means that it is not possible to define the creation or removal of species, compartments, reactions, or other components from within a model. To simulate the creation or destruction of compartments, one has to use tricks. For example, a model could define all the compartments it could ever need and use variables to indicate which compartments are actually "active" at any given time. However, this would only work if the total number of compartments needed is known at the beginning of a simulation. This approach hard-codes the anatomical description of a model, which limits portability and becomes unpractical beyond a few compartments.

Another limitation is the lack of appropriate constructs to represent the spatial location rearrangement that follows dynamic cellular processes. It can be argued that though unavailable in Core, SBML Level 3 Version 1 extensions such as Layout and Spatial provide the necessary constructs to represent the spatial positioning of modeling elements. However, location as implemented in the Layout package only indicates how components are to be positioned only for user visualization, which may be different from where elements are during simulation. Similarly, though the Spatial package enables the spatial mapping of modeling elements, the position of mapped components is not readily accessible, so it can not be updated as cells move, die and proliferate throughout simulation.

In defense of SBML's limitation in this area, it should be pointed out that the advent of software tools supporting dynamic cellular processes was only a handful until a few years ago. It is now clear that a variety modeling problems and platforms would benefit from this capability.

2.2 Past work on this problem or similar topics


A previous proposal for this language extension outlined the need to use compound data structures available in the Arrays and Sets package to accommodate for dynamic behavior of static SBML components. In this way, cell proliferation could be represented by adding a new cell to an array of cells, or increasing the dimension of an array of compartments. Similarly, cell death could be represented by the removal of a cell from a set. Nonetheless, dynamic creation/destruction of components using arrays proved to be unpractical for large dynamic simulations. The current approach involves extending already existing SBML components and introducing new ones to emulate dynamic cellular processes as opposed to using constructs from other SBML extensions.

3 Proposed syntax and semantics

In this section, we define the syntax and semantics of the Dynamic Structures package for SBML Level 3 Version 2. We delineate the data types and constructs defined in this package, then in [Section 5 on page 14](#), we provide complete examples of using the constructs in example SBML models.

3.1 Overview of dyn extension

The primary mechanism this package uses to support dynamic cellular behavior are the **Event** and **EventAssignment** constructs. The Dynamic Structures package extends these components not only to allow modelers to indicate at which point during simulation (i.e., under what cellular conditions), but how certain SBML structural components (compartments or submodels) are affected in response to dynamic cellular processes. To fully enable this, dyn also extends the **Compartment** object class to facilitate the representation of spatial rearrangements that follow dynamic cellular processes. Though such representations vary across tools based on the chosen modeling paradigm, this language extension defines the necessary constructs to enable tools to encode dynamic cellular behavior regardless of mathematical method or spatial representation framework.

 **Harold:** Out of the types of off-lattice frameworks, our proposed approach only covers center-based (objects spatially identified by the coordinates of their center of mass) and doesn't support vertex-based (where each object has several vertices that together uniquely describe its position)

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 SBML 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 Dynamic Structures package for SBML Level 3 Version 2:

`"http://www.sbml.org/sbml/level3/version2/dyn/version1"`

In addition, SBML documents using a given package must indicate whether understanding the package is required for complete mathematical interpretation of a model, or whether the package is optional. This is done using the attribute **required** on the `<sbml>` element in the SBML document. For the Dynamic Structure package, the value of this attribute must be set to **"true"**. The following fragment illustrates the beginning of a typical SBML model using SBML Level 3 Version 2 Core and this version of the dyn package:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version2/core" level="3" version="2"
      xmlns:fb="http://www.sbml.org/sbml/level3/version2/dyn/version1" arrays:required="true">
```

3.3 Primitive data types

The Dynamic Structure package uses a number of the primitive data types described in Section 3.1 of the SBML Level 3 Version 2 Core specification as well as a number of XML Schema 1.0 data types ([Biron and Malhotra, 2000](#)). More specifically, we make use of **SIId**, **int**, and **SIIdRef**. This package also adds additional primitive data types described below.

3.3.1 Type **CBOTerm**

The type **CBOTerm** is used as the data type of the attribute `cboTerm` on the **Event** object class. **CBOTerm** follows the syntax defined for the **anyURI** data type, which considers it a character string data type whose values are interpretable as URIs (Universal Resource Identifiers; ([Harold and Means, 2001](#)); ([W3C, 2000](#))) as described by the W3C document RFC 3986 ([Berners-Lee et al., 2005](#)). Examples of valid string values of type **CBOTerm** are:

http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#Apoptosis

http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#Movement

These values are meant to be the identifiers of terms from the Cell Behavior Ontology (CBO) whose curated vocabulary describes cellular entities and processes in computational models. [Section 4 on page 11](#) provides more information about the ontology, supported ontological terms and general principles for their use in SBML models.

3.3.2 Type **CoordinateSystemKind**

Type **CoordinateSystemKind** is a primitive data type that is used in extending the **Compartment** object to indicate the coordinate system in which the position of modeling elements is to be specified. **CoordinateSystemKind** is derived from the basic XML type **string**, but with restrictions about the sequences in which characters may appear. Values for attributes of this data type can only include: “**cartesian**”. The meaning of each of these values is discussed in the context of the extended **Compartment** object in [Section 3.6 on the following page](#).

Harold: If we indicate a coordinateSystem such as cartesian, we should also support other such as "cylindrical", "spherical", and "polar" (which the spatial package also supports). It would make for a more

If SED-ML is used, using more coordinate systems would mean that not only would we need to worry about translating how dynamic processes are represented across spatial frameworks, but coordinate systems! (Another layer of complexity)

3.3.3 Type **CoordinateKind**

The **CoordinateKind** primitive data type is used in the definition of the **CoordinateComponent** class. **CoordinateKind** is derived from the basic XML type **string** though it restricts possible values attributes of this data type can take. Supported values include: “**cartesianX**”, “**cartesianY**”, “**cartesianZ**”. Attributes of **CoordinateKind** type are discussed in the context of the extended **Compartment** object in [Section 3.6 on the next page](#).

Harold: If cylindrical, spherical and polar coordinate systems are used, the list of possible values for this data type would need to be extended to include: "sphericalRadius", "sphericalAzimuth", "sphericalElevation", "cylindricalRadius", "cylindricalAzimuth", "cylindricalHeight", "polarRadius", and "polarAzimuth".

3.4 The extended **Event** object

The **Event** class is extended in the Dynamic Structure package. The addition of a **cboTerm** attribute is designed to allow a modeler or a software package to attach semantical information to events triggered in response to dynamic cellular processes in an SBML model. [Figure 1](#) provides the corresponding UML diagram.

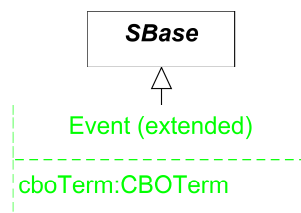


Figure 1: A UML representation of the extended **Event** class. See [Section 1.3](#) for conventions related to this figure.

3.4.1 The **cboTerm** attribute

The attribute called **cboTerm** on the **Event** class supports the use of Cell Behavior Ontology(CBO) terms to provide semantic encoding of dynamic cellular events in SBML models. By adding CBO terms, modelers can semantically

annotate event constructs to explicitly indicate which cellular processes are being modeled in each **Event** instance. The relationship is of the form "the Event is-a X", where X is the CBO term. The term chosen should be the most precise one that captures the role of the Event in the model. The value of **cboTerm** must conform to the syntax permitted by the **CBOTerm** data type previously described in [Section 3.3.1 on page 6](#) of the current specification. The scope of the **cboTerm** attribute is local to the enclosing object definition and is not visible outside the object definition. For a discussion on supported values for **cboTerm** and CBO in general see [Section 4 on page 11](#)

3.5 The extended **EventAssignment** object

In every instance of an event definition in a model, the subobject **ListOfEventAssignments** element must have a non-empty list of one or more elements of class **EventAssignment**. This package extends the **EventAssignment** class to add a **component** attribute to allow modelers to indicate which cellular structures are involved in an event executed to model dynamic cellular behavior. The definition for the extension of the **EventAssignment** element is shown in [Figure 2](#).

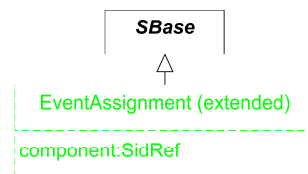


Figure 2: A UML representation of the extended **EventAssignment** class. See [Section 1.3](#) for conventions related to this figure.

3.5.1 The **component** attribute

Component is a mandatory attribute of the type **SidRef** that is used to uniquely identify the cellular structure (either compartment or submodels) to be dynamically altered when a dynamic event is executed. When this attribute points to **Compartment** elements, they must have a child **ListOfCoordinateComponents** as seen in [Section 3.6](#). This ensures that when an extended event is executed, referenced cellular compartments will have an assigned spatial location, which is necessary when modeling cellular behavior. If this is not the case, the **EventAssignment** is not considered to be part of the dynamic representation of the model.

Harold: How do we exactly indicate what happens to all the compartments within? Do we have separate **EventAssignments** one for each contained element?

3.6 The extended **Compartment** object

The **Compartment** class must be extended to represent the spatial location and subsequent rearrangement that follows dynamic cellular processes emulated by extended **Event** constructs. Refer to [Section 4 on page 11](#) for a list of supported dynamic cellular processes and associated ontological terms. This package adds the mandatory attribute **coordinateSystem** and subobjects **ListOfCoordinateComponents** and **CoordinateComponent** to describe the spatial location of modeling objects. [Figure 3](#) provides the corresponding UML diagram of the various features of the extended **Compartment** class.

Harold: If we add a rotation attribute here to support 2D representations (bacterial growth) where rotation is important in spatially describing objects, how exactly do we do it? Rotation would be measured in what units (radians, degrees)? Would rotation be relative to a given coordinate component?

Also, should the **CoordinateSystem** attribute reside in **Compartment**? If we only have one allowed value (cartesian) it doesn't matter. However, if we allow several possible coordinate systems (spherical, cylindrical, etc) users may indicate different ones for extended **Compartments** in the same model, which would be an error. As we cannot use

defaults, I think this attribute should be more general so it applies to all Compartments at once.

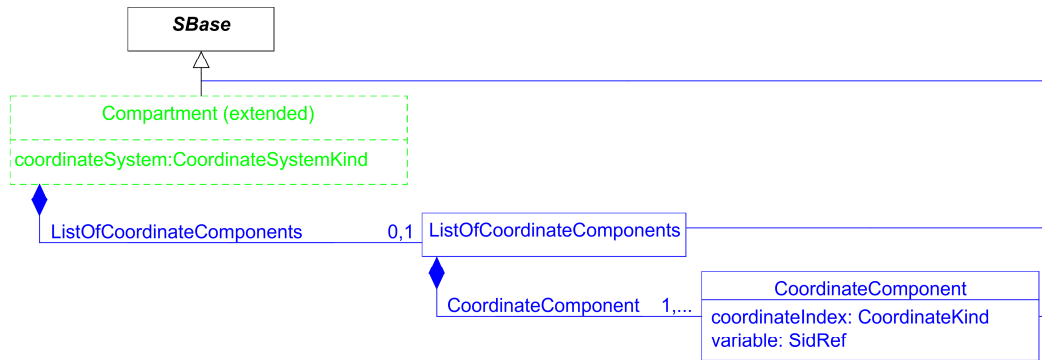


Figure 3: UML diagram for the extension of the **Compartment** object and the definition of **ListOfCoordinateComponents** and **CoordinateComponent** classes. See [Section 1.3](#) for conventions related to this figure.

3.6.1 The coordinateSystem attribute

The **coordinateSystem** attribute is a required attribute of type **CoordinateSystemKind**. It represents the coordinate system used by a **Compartment** to indicate its spatial location. A value of “cartesian” indicates that the **Compartment** is in a cartesian coordinate system (1, 2 or 3D) and that the coordinate components correspond to the x, y, and z cartesian components.

Harold: Do we expect to model something of higher dimensions than 3D cartesian coordinate system? Specifying how the actual dynamic events are carried out in SED-ML is not only dependent on translating between spatial frameworks (off-lattice vs lattice-based), but on the dimension of the spaces indicated. We need to be careful!

3.6.2 The ListOfCoordinateComponents class

Compartments mapped by an **EventAssignment** construct must contain an element called **listOfCoordinateComponents**, of class **ListOfCoordinateComponents**. This object contains a list of **CoordinateComponent** objects that define the coordinate system in which the component is modeled. If mapped, a **Compartment** cannot have an empty **ListOfCoordinateComponents** list.

3.7 The CoordinateComponent class

Instances of the **CoordinateComponent** class represent individual coordinate components of the spatial location of mapped **Compartment** structures that are involved in dynamic events. This construct defines two attributes: **coordinateIndex** and **variable**. Being derived from **SBase**, this class also has all the usual attributes and elements of its parent class.

Harold: So far we’ve defined the coordinate system and components to each x,y,z triplet for all compartments involved but nowhere have we defined the minimum and maximum values of the coordinate axis along which objects are modeled.

3.7.1 The coordinateIndex attribute

The attribute **coordinateIndex** of type **CoordinateKind** is added to uniquely identify a coordinate axes in the defined **coordinateSystem**. **CoordinateIndex** may take one of all the possible **CoordinateKind** values specified in [Section 3.3.3 on page 7](#). A single “cartesianX” **CoordinateComponent** can be used to define one-dimensional systems; two-dimensional systems are in turn characterized by having two **CoordinateComponent** children with **coordinateIndex** values of “cartesianX” and “cartesianY”; and three-dimensional ones can be defined by

having three **CoordinateComponent** elements with **coordinateIndex** values of “**cartesianX**”, “**cartesianY**”, and “**cartesianZ**”.

3.7.2 The *variable attribute*

The **variable** attribute of type **SidRef** contains the identifier of a **Parameter** defined in the model to explicitly specify an object’s position. The scope of the referenced **Parameter** component must be global to the whole model and its **constant** attribute must be set to “**false**” as spatial location of the associated **Compartment** may change in response to dynamic processes.

4 The Cell Behavior Ontology and the `cboTerm` attribute


It is difficult to determine the semantics of **Event** constructs used to model intrinsic cellular behavior from SBML attributes alone. The `id` attribute on **Event** objects allows for unique identification and cross-referencing while the `name` attribute allows the assignment of human readable labels to Events. Possible values for these attributes are unrestricted so that modelers can choose whichever fits their modeling framework and preference best. However, this means that without any additional human intervention, software tools are unable to discern the semantics of an extended **Event** element modeling dynamic behavior. For instance, it would be inadvisable to interpret that an **Event** is modeling the process of cellular death even if the `id` and `name` of such **Event** have the string “Cell Death” as value. Additionally, as one may need to convert a dynamic **Event** between different representations (e.g., Cellular Potts Model vs. Center-based off-lattice Model), there is a need to provide a standard, framework-independent, way of associating **Event** components with given cellular processes.

A solution inspired by SBML Level 3 Version 2 Core is to associate model components with terms from carefully curated controlled vocabularies (CVs). This is the purpose of the `cboTerm` provided in the extended **Event** class in [Section 3.4 on page 7](#). The `cboTerm` facilitates the annotation of **Event** components with terms belonging to the Cell Behavior Ontology (CBO) ([Sluka et al., 2014](#)). In this section, we discuss CBO, its usage in SBML models via the `cboTerm` attribute and relevant modeling implications.

4.1 Cell Behavior Ontology (CBO)

The development and use of bio-ontologies stems from the need to characterize and describe domains of biology in a standard way. The Cell Behavior Ontology (CBO) provides a carefully curated, controlled vocabulary that can be used to describe the behavior of a cell over time (dynamics) in a framework-independent manner, which enables the reliable exchange of biological descriptions. The Dynamic Structures SBML extension allows modelers to use a subset of the available identifiers to tag SBML **Event** components via its attribute `cboTerm` to make the underlying biology (spatiality) of the cellular process being modeled more explicit. The relationship between a `cboTerm` term describing an extended **Event** and the CBO term being used is of the form “the Event is-A X”, where X is the CBO term. Though CBO support provides an important source of information to understand the meaning of an **Event**, software does not need to support `cboTerms` to be considered SBML-compliant.

Although the use of `cboTerm` attributes for **Event** components extended by this extension is required, the presence of a `cboTerm` is not understood to change the way the **Event** is mathematically interpreted and simulated. Annotating SBML **Event** elements with CBO terms simply adds additional semantic information that may be used to convert the **Event** from one framework to another when shared; it enables software tools to recognize precisely what the component is meant to be. For example, if the `cboTerm` has the value of http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDeath for a given **Event**, regardless of the value that the `id` or `name` attributes are given or the modeling method used, the **Event** labeled with this ontological term will be understood to model the dynamics of cellular death.

 **Harold:** The initial release of CBO assumes spatial objects are 3D and exist in a standard Cartesian coordinate system, but does this really apply to us? By simply annotating using CBO terms, we are using CBO differently from its intended use, which involves the creation of a meta-model of a computational model and subsequent linking of it to simulation results. If this is not how CBO is meant to be used, does this mean that we don't have to enforce CBOs restrictions for objects modeled in lower dimensions? We'd have to think what to do if we allow objects to be modeled in other coordinate systems?

4.1.1 Structure of the Cell Behavior Ontology

The purpose of CBO is to standardize the description of the intrinsic physical and biological characteristics of cells and tissues, which provides a basis for describing the spatial and observable dynamic behavior of cells in SBML models. To achieve this, CBO is split into controlled vocabularies for **CBO_Objects**, which describe the physical entities of a biological model and **CBO_Processes**, which describe the processes the aforementioned objects

participate in. [Figure 4](#) illustrates the taxonomy of CBO at the highest level.

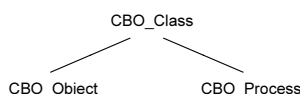


Figure 4: The controlled vocabularies that make up the main branches of CBO.

As this SBML extension uses CBO only as a reference ontology for the description of dynamic processes described as events, all of the supported vocabulary is taken from the **CBO _Process** branch. Though **CBO _Process** terms encompass length scales that range from subcellular to cell aggregates and time scales that range from seconds to decades, only the subset in [Section 4.2.1](#) is allowed.

4.2 Using CBO and cboTerm

The **cboTerm** attribute for extended **Event** constructs is always of **CBOTerm** data type, as defined in [Section 3.4.1](#) on page 7. When present, the attribute's value must be the full identifier of a single term taken from the Cell Behavior Ontology (<http://bioportal.bioontology.org/ontologies/CBO>). The term chosen should be the most precise one that best summarizes the phenomenon represented by the extended **Event** object. The relationship indicated by the presence of a non-empty **cboTerm** attribute in an **Event** is of the form "the Event is-A X", where X is the CBO term.

4.2.1 Supported CBO terms in Event Components

One of the mechanisms the Dynamic Structures package uses to support dynamic cellular behavior is the extension of the already existing SBML **Event** construct. Under this extension, **Event** objects carry a **cboTerm** attribute whose value must be a full term identifier, taken from the **CBO _Process** vocabulary branch, which describes the behavior modeled by said **Event**. Given substantial community input, the initial version of this package only supports a handful of dynamic cellular behaviors. [Table 1](#) displays supported dynamic processes and their corresponding CBO terms.

Cell Behaviors	CBO Terms
Cell Division	http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDivision
Cell Death	http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDeath
Cell Differentiation	http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDifferentiation

Table 1: Dyn-supported cell behavior and corresponding CBO terms



Harold: I am being very conservative here. Not sure differentiation belongs here and how we would support it. In consideration are growth and movement though movement is implied by already modeled terms such as differentiation. Does this mean that we need to support movement explicitly or just as a consequence of other cell behavior? Shall we include any others?

A CBO term for **Cell Division** as value for a **cboTerm** attribute indicates that the **Event** defines the mathematical conditions under which cell division is to take place by means of its **Trigger**. **Compartments** whose **id** is reference by the **component** attribute of contained **EventAssignment** constructs, and all SBML elements contained within, will be copied into a daughter compartment and placed according to each **EventAssignment**.

A CBO term for **Cell Death** as value for a **cboTerm** attribute indicates that the **Event** defines the mathematical conditions under which cell death is to take place by means of its **Trigger**. **Compartments** whose **id** is reference by the **component** attribute of contained **EventAssignment** constructs, and all SBML elements contained within, will

be removed from simulation according to each **EventAssignment**.

A CBO term for *Cell Differentiation* as value for a **cboTerm** attribute indicates that the **Event** defines the mathematical conditions under which cell differentiation is to take place by means of its **Trigger**.

 **Harold:** Is the move/positioning is carried out left to SED-ML? There are many kinds of Cell-Division, examine closer to determine what we need for each!

4.2.2 Tradeoffs in using CBO terms

The presented CBO-based approach to annotating SBML **Event** components with controlled terms has, just like the SBO-based approach presented in SBML Level 3 Version 2 Core, the following strengths:

1. The syntax required is very straight-forward and requires a single **string** containing the **id** of a supported CBO term.
2. Supported CBO terms cover a relevant portion of the cellular behaviors required by the community
3. It does not interfere with already-existing annotation schemes implemented by either Core and SBML extensions.

The following list illustrates some of the weaknesses of following the proposed approach:


1. The Cell Behavior Ontology is a recent and evolving ontology. As such, it is susceptible to minor changes in its hierarchical taxonomy. These however, should not affect the **ids** of the terms themselves.

 **Harold:** Can we think of any other benefits or weaknesses?

4.2.3 Relationships to the SBML annotation element

A way to provide additional information to that within SBML elements is the **sBase** usage of the **Annotation** component. Annotations are commonly used by software tools, which have their own vocabulary for supporting similar cellular behaviors. However, the best-practice recommendation for interoperability is to use the **cboTerm** attribute in the **Event** element rather than inside an **Annotation** component. Software tools are encouraged to translate their tool-specific **Annotation** scheme to the proposed **CBO-based** approach when writing SBML code that supports dynamic cellular behavior.

4.3 Discussion

 **Harold:** In this section SBML core touches on the implication of adding CBO support such as frequency of change of the ontology and consequences to already existing models, consistency, and Internet access and caching ontology version. This would be practically identical though I could include a few words on this

5 Examples

This section contains a variety of examples of SBML Level 3 Package Specification for Dynamic Structures, Version 1 documents employing the dyn package.

5.1 Cell Division on a 2D grid

6 Best practices

In this section, we recommend a number of practices for using and interpreting various constructs in the dyn package. Neglecting these recommendations may not render a model invalid, but may reduce the degree of interoperability that can be achieved when sharing it. Hence, we strongly advocate the following practices when using the SBML Level 3 Package Specification for Dynamic Structures, Version 1:

Acknowledgments

We thank ...

In addition we would like to thank the authors ...

1
2
3

References

Berners-Lee, T., Fielding, R., and Masinter, L. (2005). Uniform resource identifier (uri): Generic syntax.

Biron, P. V. and Malhotra, A. (2000). XML Schema part 2: Datatypes (W3C candidate recommendation 24 October 2000). Available via the World Wide Web at <http://www.w3.org/TR/xmlschema-2/>.

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

Fallside, D. C. (2000). XML Schema part 0: Primer (W3C candidate recommendation 24 October 2000). Available via the World Wide Web at <http://www.w3.org/TR/xmlschema-0/>.

Harold, E. R. and Means, E. S. (2001). *XML in a Nutshell*. OŠReilly.

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

Sluka, J. P., Shirinifard, A., Swat, M., Cosmanescu, A., Heiland, R. W., and Glazier., J. A. (2014). The Cell Behavior Ontology: Describing the Intrinsic Biological Behaviors of Real and Model Cells Seen as Active Agents. *Bioinformatics*.

Thompson, H. S., Beech, D., Maloney, M., and Mendelsohn, N. (2000). XML Schema part 1: Structures (W3C candidate recommendation 24 October 2000). Available online via the World Wide Web at the address <http://www.w3.org/TR/xmlschema-1/>.

W3C (2000). W3C’s math home page. Available via the World Wide Web at <http://www.w3.org/Math/>.