Numerical Results Markup Language (NuML)

Level 1 Version 1 (Draft)

Prepared by
Joseph O. Dada
The University of Manchester
Manchester, UK

Date: 20/10/2011

Latest release is available at
http://code.google.com/p/numl/

Comments and suggestions on the NuML specification should be sent to the mailing list at
numl-discuss@googlegroups.com

Authors' contact:
joseph.dada@manchester.ac.uk

Contents

1 Introduction 3
1.1 System models and their states 3
1.2 Model analyses 4
1.3 Organizing numerical results 5
1.4 Document conventions 5
1.4.1 Color conventions 5
1.4.2 Typographical convection for names 6
1.4.3 UML Notation 6
2 Overview of NuML 9
3 Preliminary definitions and principles 11
3.1 Primitive data types 11
3.1.1 Type NMId 11
3.1.2 Type DataType 11
3.2 Type NMBase 12
3.2.1 The metaid attribute 12
3.2.2 The notes association 13
3.2.3 The annotation association 13
4 NuML components 13
4.1 The NuML container 13
4.2 Ontology Term 14
4.2.1 The id attribute 15
4.2.2 The term, sourceTermId, and sourceURI attributes 15
4.2.3 Example 15
4.3 Result component 15
4.3.1 The id attribute 16
4.4 The dimension description 16
4.5 The composite description 17
4.6 The tuple description 17
4.7 The atomic description 17
4.8 The dimension 17
4.9 The composite value 17
4.10 The tuple 18
4.11 The atomic value 18
5 Examples of numerical results encoded in NuML 18
5.1 Example using xpath as attribute value in composite description and composite value 18
5.2 Example with tuple and atomic value 19
5.3 Example with composite value(s) and atomic value 20
5.4 Example with composite value, tuple and atomic value 20
5.5 Online example of NuML 21
Acknowledgements 21
Appendix B: XML Schema for NuML 22
References 26

1 Introduction

The introduction of SBML (Hucka et al., 2003) as an exchange format for models of biochemical systems has been a major factor in advancing computational systems biology. The community of scientists and software developers involved with SBML (hereafter called the "SBML community") has also made several other technical advances as a consequence, for example by creating software libraries that make it easier to develop programs compatible with SBML, and by creating collections of models that act as gold standards for testing and therefore also for validating and comparing software. There have also been several other standardization activities, for example in defining what information should be communicated when models are constructed (Le Novère et al., 2005), description of simulation experiments (Kohn & Le Novère, 2008), markup language for associating systems biology data with models (Dada et al., 2010), etc. However, one aspect that has until now not been addressed in terms of standardization is a universal language for encoding numerical data. This document is intended to propose a solution addressing this issue. It is hoped that it will generate discussion and after appropriate modifications result in a final specification for a Numerical Results Markup Language (NuML). NuML originates from the numerical aspects of SBRML: Systems Biology Results Markup Language (Dada et al., 2010) with the aim of re-using it in multiple other standardization efforts. It is envisioned that NuML will provide a standardized and universal way for exchanging and archiving numerical results.

1.1 System models and their states

For the purposes of this document, it is important to first provide an operational definition of what system model consists of.

A computational/mathematical view of models is that they are a set of *functions* and associated *parameter values* that map a transformation of a system from an initial *state* to a final *state*. A simulation can then be seen as a transformation between two states carried out by a set of mathematical equations (embodied in computational methods). This view is generic and includes several types of simulation (e.g. time courses, steady states, etc.) and different mathematical frameworks (sets of ODEs, Gillespie's SSA, Petri nets, etc.). Such models can be represented in different exchange formats e.g. SBML, CellML.

An important note regards the very special model entity known as "time". This is usually seen as a special independent variable of models and a large set of analyses are concerned with how the system changes its state along monotonically increasing values of time (time course simulations). However, for the purposes of this document and of the proposed markup language (NuML), it is useful to regard "time" as just any other parameter of the model.

As will be argued in more detail below, the product of computational systems analyses are always collections of states of the system, each one associated with the values of the parameters that generate them. So it becomes important to first create a data model for the state of a system.

A single state of a system/model is thus characterized by 1) a set of parameter values, and 2) a set of

values of the state variables. In more detail we have:

1. A set of parameter values of the model that specify the state. The parameter values however depend on the system being model. For computational systems biology where NuML originated from, the parameter values are:
   1.1. kinetic constants of the rate laws of all reactions
   1.2. initial concentrations of all species
   1.3. values of any arbitrary constants
   1.4. initial values of any arbitrary dependent variables
   1.5. initial values of compartment volumes
   1.6. initial time value (usually zero, but can be different in non-autonomous models)

2. A set of values of the state variables (that uniquely defines the state). Again the state variables depend on the model system. Example of state variables in systems biology are:
   2.1. species concentrations or particle numbers
   2.2. compartment volumes
   2.3. arbitrary dependent variables

A state of a model also includes a number of other quantities that can be derived from the set in 2. above. For example, in biochemical model, the reaction fluxes are a function of the species concentrations (through the reaction kinetic rate laws). A list of such quantities includes:

1. Derived state variables:
   1.1. Reaction fluxes (either in concentration or particle numbers)
   1.2. Jacobian matrix (contains the partial derivatives of the ODEs)
   1.3. Eigenvalues of the Jacobian matrix
   1.4. Arbitrary sensitivity coefficients
   1.5. Lyapunov exponents
   1.6. Elasticity coefficients
   1.7. Concentration-control coefficients
   1.8. Flux-control coefficients
   1.9. Response coefficients (defined as the total derivative of a variable towards some parameter
   1.10. Any other function of the primary state variables (or from any other derived state variable

Any quantity that, in order to be calculated requires the determination of any primary state variable is thus a derived state variable. The distinction between primary and derived state variables is important because the primary state variables are the only ones required to determine unequivocally the state of the system – they are the most basic properties of that system. Given the complete set of primary state variables of a model and its equations, it is possible to determine a finite number of sets of parameter values that could lead to such states (even if this is technically difficult).

1.2 Model analyses

The simplest operation that can be applied to this type of model is the calculation of a “final” state of the system based on a given initial state. The majority of operations are actually composed of several of these calculations where something is changed in the model, thus the results file should be able to represent not just one “final” state, but rather several of them and they should be associated with the conditions particular to each one of them.

One can calculate the state of the system for any value of time, and thus a time series is nothing more than a set of states of the system, each one associated with a different value of time. Steady states are special instances of the system where the state corresponds to the case of time=infinity (because they are asymptotic).

1.3 Organizing numerical results

Numerical results can be expressed for a single state of the model or for multiple states of the model. For the purpose of this markup language it is useful to first define how to specify results for a single state and then to organize collections of states. There are two options: *i*) list all states as tuples containing parameter values and values of state variables; or *ii*) have tuples that contain only values of state variables and then index them by parameter values. The first case corresponds to a simple list of results and is appropriate when the parameter values are not ordered and/or have irregular intervals. The second is most appropriate when the parameter values change in regular ways and take many different values. An example of the first case would be a list of results from a random sampling of values of five parameters of the model; an example of the second would be the time series resulting from a specific initial condition, which is then iterated for several different values of four parameters in a regular sweep (*i.e.* exploring the dynamics at all values of those four parameters). The former case would be a simple list; the latter would be a four dimensional data cube.

1.4 Document conventions

In this section, we describe the conventions we use in this specification document in an effort to communicate information more effectively and consistently.

1.4.1 Color conventions

Throughout this document, we use coloring to carry additional information for the benefit of those viewing the document on media that can display color:

We use blue color in text to indicate a hyperlink from one point in this document to another. Clicking your computer's pointing device on blue-colored text will cause a jump to the section, figure, table or page to which the link refers. (Of course, this capability is only available when using electronic formats that support hyper linking, such as PDF and HTML.)

1.4.2 Typographical convection for names

The following typographical notations are used in this document to distinguish objects and data types from other kinds of entities:

AbstractClass: Abstract classes are classes that 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 (e.g. ***AbstractClass***). In electronic document formats, the class names are also hyper linked to their definitions in the specification. For example, in the PDF and HTML versions of this document, clicking on the word NMBase will send the reader to the section containing the definition of this class.

Class: Names of ordinary (concrete) classes begin with a capital letter and are printed in an upright, bold, sans-serif typeface (e.g. **MyClass**). In electronic document formats, the class names are also hyper linked to their definitions in the specification. For example, in the PDF and HTML versions of this document, clicking on the word OntologyTerm will send the reader to the section containing the definition of this class.

SomeThing, *otherThing*: Attributes of classes, data type names, literal XML, and generally all tokens other than NuML UML class names, are printed in an upright typewriter typeface (e.g. myAttributeName). Primitive types defined by NuML begin with a capital letter, but unfortunately, XML Schema 1.0 does not follow any convention and primitive XML types may either start with a capital letter (e.g. ID) or not (e.g., double).

1.4.3 UML Notation

The Unified Modeling Language (UML) is the Object Management Group (OMG, http://www.omg.org/) specification defining a graphical language for visualising, specifying, constructing, and documenting the artifacts of object-based software. NuML uses a subset of the UML standard, the class diagram, which represent a static view of the data structure. There are three main advantages of using UML as basis for defining NuML data objects. First, compared to using other notations or a programming language, the UML visual representations are generally easier to grasp by readers who are not computer scientists. Second, the notation is implementation-neutral: the objects can be encoded in any concrete implementation language—not just XML, but C, Java and other languages as well. Third, UML is a *de facto* industry standard that is documented in many resources. Readers are therefore more likely to be familiar with it than other notations.

Object class definitions

Object classes in UML diagrams are drawn as simple tripartite boxes, as shown in Figure 1.1. The top box contains the name of the class. The middle box contains a list of attributes. Attributes in NuML data model are defined as optional (“0..1”) or mandatory (“1”) by the cardinality specified next to class attribute name. The bottom box of a UML class contains a listing of the operations that are performed by the class. This box is not used in NuML because we are only interested in data model; hence the box is empty.

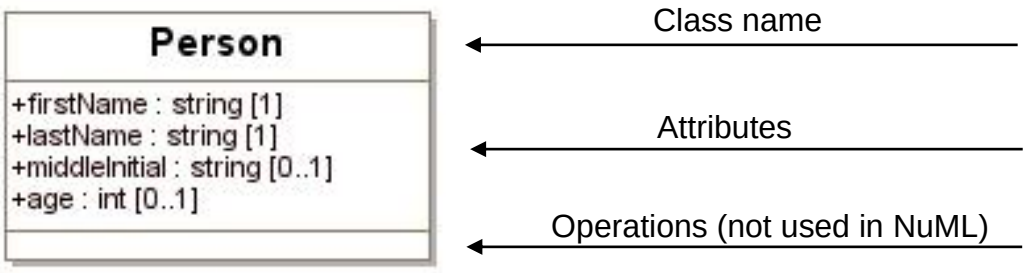


Figure1.1: UML Class with boxes outlined

As mentioned above, the names of ordinary (concrete) classes begin with a capital letter and are printed in an upright, bold, san-serif typeface. The names of attributes begin with a lower-case letter and generally use a mixed case (sometimes called "camel case") style when the name consists of multiple words. Attributes and their data types appear in the part below the class name, with one attribute defined per line. The colon character on each line separates the name of the attribute (on the left) from the type of data that it stores (on the right). The subset of data types permitted for NuML attributes is given in Section 3.1. In the diagram of Figure 1.1, the firstName, lastName, middleInitial and age represent attributes of the object class **Person**. The data type of firstName, lastName, and middleInitial is string, and the data type of age is int. In the scheme used by NuML for translating UML to XML, object attributes map directly to XML attributes. Thus, in XML, **Person** would yield an element of the form <element firstName="John" lastName="Pama" age="23">.

Notice that the element name is not <**Person** ...>. Somewhat paradoxically, the name of the element is not the name of the UML class defining its structure. The reason for this may be subtle at first, but quickly becomes obvious: object classes define the form of an object's content, but a class definition by itself does not define the label or symbol used to mark up an instance of that content. It is this label that becomes the name of the XML element. In XML, this symbol is most naturally equated with an element name. This point will hopefully become clearer with additional examples below.

UML relationships

There are two types of relationships between classes in NuML: associations and generalisations. Association relationships between classes are shown as an arc joining the two classes (Figure 1.2), with the name of the association next to the arc. The multiplicity/cardinality of associations specifies how many instances of one class may relate to a single instance of another class. This is shown by numbers and stars at the end of an association path. The multiplicity may be one-to-one, one-to-many, optional, and many-to-many. Associations are either bi-directional, meaning each association end is accessible from the other (hence both classes can traverse the relation), or unidirectional, meaning only one association end is accessible and the relation can only be traversed in one direction. Only unidirectional associations are used in NuML.

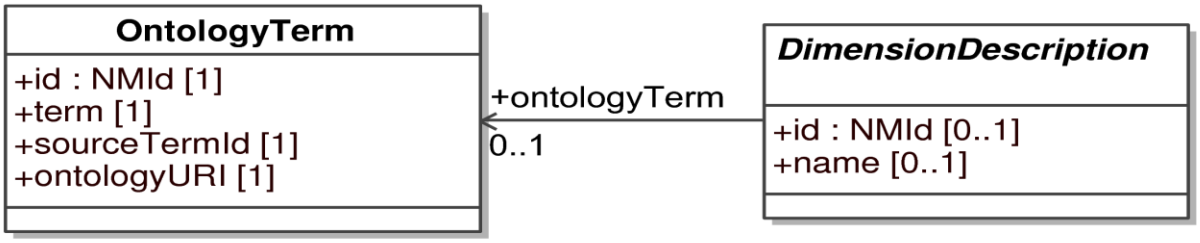


Figure 1.2: Example illustrating unidirectional association between DimensionDescription and OntologyTerm classes. In this example, the unique ID of the reference element is an attribute of the associating element, with the name of the association.

Composition and aggregation associations

Composition and aggregation are special forms of relationship in which the classes are tightly bound by the whole/part relationship. The composition association is a stronger form of whole/part relationship than the aggregation relationship. We use UML composition and aggregation association to indicate a class instance can have other class instances as parts. Such containment hierarchies map directly to element-subelement relationships in XML. Figure 1.3 gives an example.

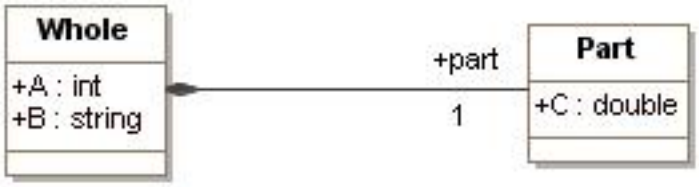


Figure 1.3: Example illustrating composition association: the definition of one class of objects employing another class of objects in a part-whole relationship. In this particular example, an instance of a Whole class object must contain exactly one instance of a Part class object, and the symbol referring to the Part class object is part. In XML, this symbol becomes the name of a subelement and the content of the subelement follows the definition of Part.

The line with the black diamond indicates composition association, with the diamond located on the "container" side and the other end located at the object class being contained. A hollow diamond indicates aggregation association. The label on the line is the symbol used to refer to instances of the contained object, which in XML, maps directly to the name of an XML element. The class pointed to by the aggregation relationship (Part in Figure 1.3) defines the contents of that element. Thus, if we are told that some element named barney is of class Whole, the following is an example XML fragment consistent with the class definition of Figure 1.2:

```
<barney A="210" B="some string">
       <part C="222.2">
</barney>
```

Sometimes numbers are placed above the line near the "contained" side of an aggregation to indicate how many instances can be contained. The common cases in NuML are the following: (0..*) to signify a list containing zero or more; (1..*) to signify a list containing at least one; (1) to signify exactly one; and (0..1) to signify exactly zero or one. This notation appears throughout this specification

document.

Inheritance or generalisation

Classes can inherit properties from other classes. Since NuML only uses data attributes and not operations, inheritance in NuML simply involves data attributes from a parent class being inherited by child classes. Inheritance is indicated by a line between two classes, with an open triangle next to the parent class; Figure 1.3 illustrates this. In this example, the instances of object class **Child** would have not only attributes C and D, but also attributes A and B. All of these attributes would be required (not optional) on instances of class **Child** because they are mandatory on both **Parent** and **Child**.

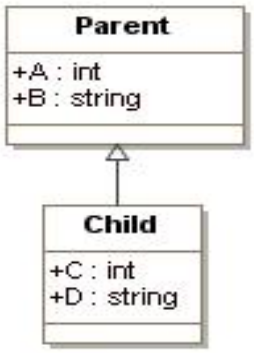


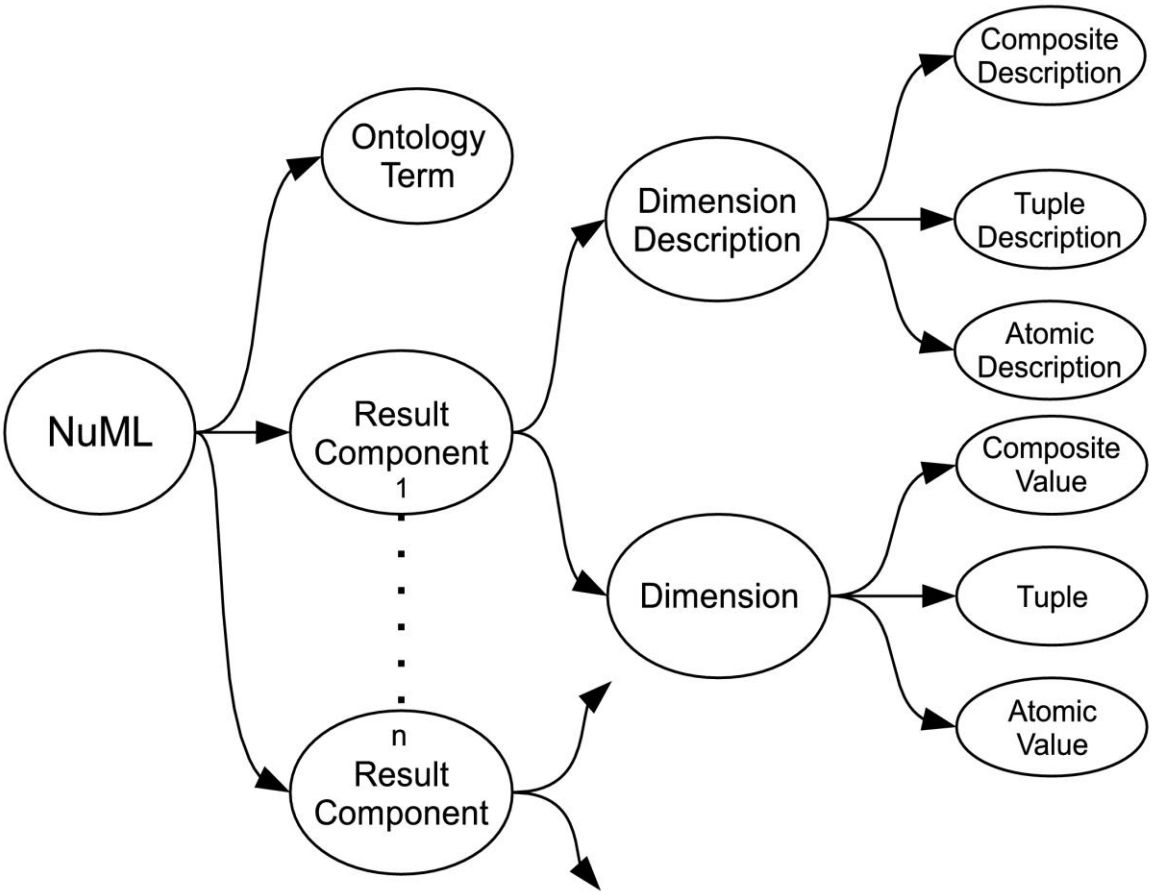
Figure 1.3: Inheritance

Additional notations for XML purposes

Not everything is easily expressed in plain UML. For example, it is often necessary to indicate some constraints placed on the values of an attribute. In computer programming uses of UML, such constraints are often expressed using Object Constraint Language (OCL), but since we are most interested in the XML rendition of NuML, in this specification we use XML Schema 1.0 (when possible) as the language for expressing value constraints.

2 Overview of NuML

NuML is intended to allow any type of numerical results to be represented. It is currently structured as follows: On the top level, NuML consists of ontology term and one or more result components. The nest level is the description of the data being represented and the data itself within the result component. The basic structure of the NuML object model (http://www.uml.org) and XML Schema (Biron & Malhotra, 2000) is presented in Figure 2.1



Ontology Term	Provides a flexible means for referencing external ontology sources for vocabularies/terms used in NuML.
Result Component	Captures the individual component of result.
Dimension Description	Header of a ResultComponent. It describes the data in the component.
Dimension	Captures the actual data of the numerical result in a result component.
Composite Description	Describes the nesting of dimensions.
Tuple Description	Describes numerical results which contain structured components that are not represented as distinct dimensions.
Atomic Description	Describes a value in a result that can no longer be subdivided.
Composite Value	Represents a nesting relationship between dimensions.
Tuple	Provides means of representing structured values that are not described as dimensions.

Atomic Value	Represents individual result within dimension or tuple.

Figure 2.1: Basic structure of NuML

3 Preliminary definitions and principles

This section covers certain concepts and constructs that are used repeatedly in the rest of NuML Level 1.

3.1 Primitive data types

Most primitive types in NuML are taken from the data types defined in XML Schema 1.0 (Biron and Malhotra, 2000; Fallside, 2000; Thompson et al., 2000). In particular, this is the case for string, positiveInteger, ID and double. A few other primitive types are defined by NuML itself. What follow is a summary of the definitions of the NuML-specific types. Readers should consult the XML Schema 1.0 specification for the normative definitions of the XML types used by NuML.

3.1.1 Type NMId

The type NMId is the type of the id attribute found on the majority of NuML components. NMId is a data type derived from the basic XML type string, but with restrictions about the characters permitted and the sequences in which those characters may appear. The definition is shown in Figure 3.1.

```
letter   ::=   'a'..'z', 'A'..'Z'
digit    ::=   '0'..'9'
idChar   ::=   letter | digit | '_'
NMId     ::=   ( letter | '_' ) idChar*
```

Figure 3.1: The definition of the type NMId expressed in the variant of BNF used by the XML 1.0 specification (Bray et al., 2004). The characters (and) are used for grouping, the character * indicates "zero or more times", and the character |indicates "or". The production letter consists of the basic upper and lower case alphabetic characters of the Latin alphabet along with a large number of related characters defined by Unicode 2.0; similarly, the production digit consists of the numerals 0..9 along with related Unicode 2.0 characters.

The equality of NMId values is determined by an exact character sequence match; i.e., comparisons of these identifiers must be performed in a case-sensitive manner. This applies to all uses of NMId. The NMId is purposefully not derived from the XML ID type (introduction section). Using XML's ID would force all NuML identifiers to exist in a single global namespace, which would affect future NuML extensions for supporting result composition. Finally, unlike ID, NMId does not include Unicode character codes; the identifiers are plain text.

3.1.2 Type DataType

The type DataType (Figure 3.2) is an enumeration type for specifying the type of valueType attribute on AtomicDescription and indexType attribute on CompositeDescription.

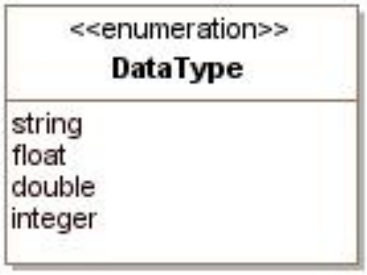


Figure 3.2: The definition of DataType

3.2 Type NMBase

Nearly every object composing a NuML Level 1 model definition has a specific data type that is derived directly or indirectly from a single abstract type called NMBase. In addition to serving as the parent class for most other classes of objects in NuML, this base type is designed to allow a user or a software package to attach arbitrary information to each major element in NuML model. The definition of NMBase (Figure 3.3) is the same as SBase in SBML. The only difference is that there is no SBOTerm attribute in NuML. All ontology terms in NuML are defined in the OntologyTerm class. NMBase contains one attribute and has two subelements, all of which are optional: metaid, notes and annotation. These are discussed separately in the following subsections.

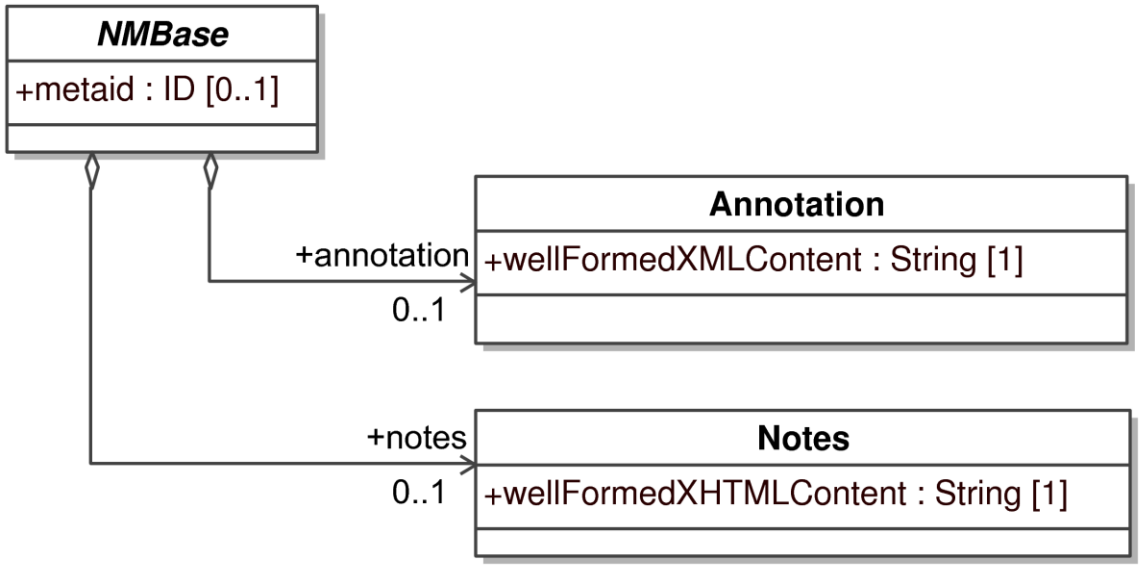


Figure 3.3: The definition of NMBase.

3.2.1 The metaid attribute

The metaid attribute is present for supporting metadata annotations using RDF (Resource Description

Format; Lassila and Swick, 1999). It has a data type of XML ID (the XML identifier type; see Section 3.1.6), which means each metaid value must be globally unique within a NuML file. The metaid value serves to identify a model component for purposes such as referencing that component from metadata placed within annotation elements (see Section 3.2.3). Such metadata can use RDF description elements, in which an RDF attribute called "rdf:about" points to the metaid identifier of an object defined in the NuML model.

3.2.2 The notes association

The association notes in SBRbase represents a container element for XHTML 1.0 (Pemberton et al., 2002) content. It is intended to serve as a place for storing optional information intended to be seen by humans. An example use of the notes association would be to contain formatted user comments about the NuML element in which the notes container element is enclosed. Every object derived directly or indirectly from type SBRbase can have a separate value for notes, allowing users considerable freedom when adding comments to simulation results.

3.2.3 The annotation association

Whereas the notes association described above represents a container element for content to be shown directly to humans, the annotation association represents a container element for optional software-generated content not meant to be shown to humans. Every object derived from SBRbase can have its own value for annotation. The element's content type is XML type any, allowing essentially arbitrary well-formed XML data content. The same restrictions placed on the organization of the annotation content in SBML also apply to NuML.

4 NuML components

In this section, we define each of the major components of NuML. We use the UML notation described in the introduction for defining classes of objects. We also illustrate the use of NuML components by giving partial model definitions in XML. Section 5 provides many full examples of NuML in XML

4.1 The NuML container

All well-formed XML documents must begin with an XML declaration, which specifies both the version of XML assumed and the document character encoding. The declaration begins with the characters <?xml followed by the XML version and encoding attributes. NuML Level 1 uses XML version 1.0 and requires a document encoding of UTF-8. Following this XML declaration, the outermost portion of simulation results expressed in NuML Level 1 Version 1 consists of an object of class Numl, defined in Figure 4.1. This class contains three required attributes, for the NuML namespace, level and version, and two required subelements; resultComponent and ontologyTerms. The resultComponent subelement is defined in the ResultComponent class. The ontologyTerm subelement is defined in the OntologyTerm class. The resultComponent and ontologyTerms are container elements. The resultComponent element must contain at least one instance of DimensionDescription and CompositeValue while ontologyTerms element contains instances of OntologyTerm.

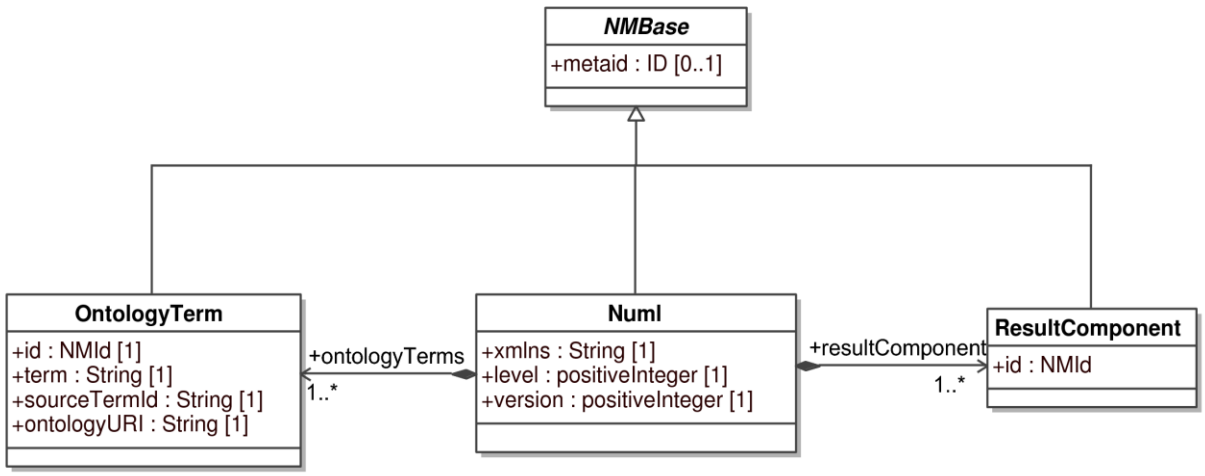


Figure 4.1: The definition of class Numl. The classes OntologyTerm and ResultComponent are defined in Subsection 4.2, 4.3, respectively.

The following is an abbreviated example of the associated XML elements for a NuML Level 1 Version 1 document:

```
<?xml version="1.0" encoding="UTF-8"?>
<numl xmlns="http://www.numl.org/nml/level1/version1" version="1" level="1">
       <ontologyTerms>……</ontologyTerms>
       <resultComponent id="rComp1">...</resultComponent>
       <resultComponent id="rComp2">...</resultComponent>
       .....
       <resultComponent id="rCompn">...</resultComponent>
</numl>
```

The attribute xmlns declares the default XML namespace used within the numl element. The URI for NuML Level 1 Version 1 is http://www.numl.org/numl/level1/version1. All elements must be placed in this namespace either by assigning the default namespace as shown above, or using a tag prefix on every element.

A NuML XML document must not contain elements or attributes in the NuML namespace that are not defined in this NuML Level 1 Version 1 Release 1 specification. Documents containing unknown elements or attributes placed in the NuML namespace do not conform to this NuML specification.

4.2 Ontology Term

The use of vocabularies/terms from standard ontology source to describe various types of data associated with the model is very important in order for software tools to correctly interpret the data. There is no single ontology source that can provide all the terms needed for the description of the very diverse numerical data. NuML provides a good model and flexible mechanism for referencing terms from any ontology sources. All terms already defined in a model that generates the numerical results encoded in NuML, for example species, reaction, etc. as in SBML could be used for data

description without specifying any ontology source for them provided the NuML document is linked with the model. The definition of OntologyTerm object class for representing vocabularies and ontologies is presented in Figure 4.2.

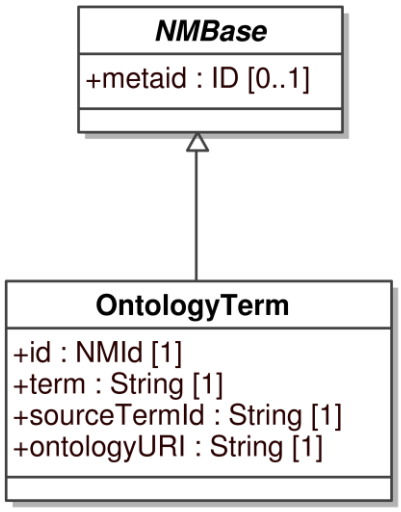


Figure 4.2: The definition of class OntologyTerm. A sequence of one or more instances of OntologyTerm used elsewhere in the NuML document are located in the ontologyTerms container element of Numl as earlier described.

4.2.1 The id attribute

The mandatory id attribute is of type NMId, and provides a unique identifier for the ontology term.

4.2.2 The term, sourceTermId, and sourceURI attributes

The term attribute stores the term itself while the sourceTermId is a string that is used within the ontology source to identify uniquely the concept being referenced by the NuML object. The ontologyURI specifies the unique identifier of the ontology source.

4.2.3 Example

```
<numl>
    <ontologyTerms>
      <ontologyTerm id="term1" term="concentration" sourceTermId="SBO:0000196"
ontologyURI="http://www.ebi.ac.uk/sbo/"/>
      <ontologyTerm id="term2" term="time" sourceTermId="SBO:0000345"
ontologyURI="http://www.ebi.ac.uk/sbo/"/>
     .....
    </ontologyTerms>
    .....
</numl>
```

4.3 Result component

The actual numerical result is defined by the ResultComponent object class. Result encoded in NuML has two component parts: the description of the result represented by the dimensionDescription

association and the result itself represented by the dimension association. This approach provides a flexible structure for representing any numerical results. There must be at least one instance of ResultComponent in numl element. The ResultComponent class serves as a container for the instances of components of subclasses of abstract classes DimensionDescription and Dimension. The subclasses allow the ResultComponent to have variable contents. We use a <choice> element to implement the variable content container. This method simply lists within a <choice> element all the elements which can appear in the variable content container, and embed the <choice> element in the container element. Readers should consult the NuML Schema for implementation details. Figure 4.3 shows the ResultComponent class with all its associated classes.

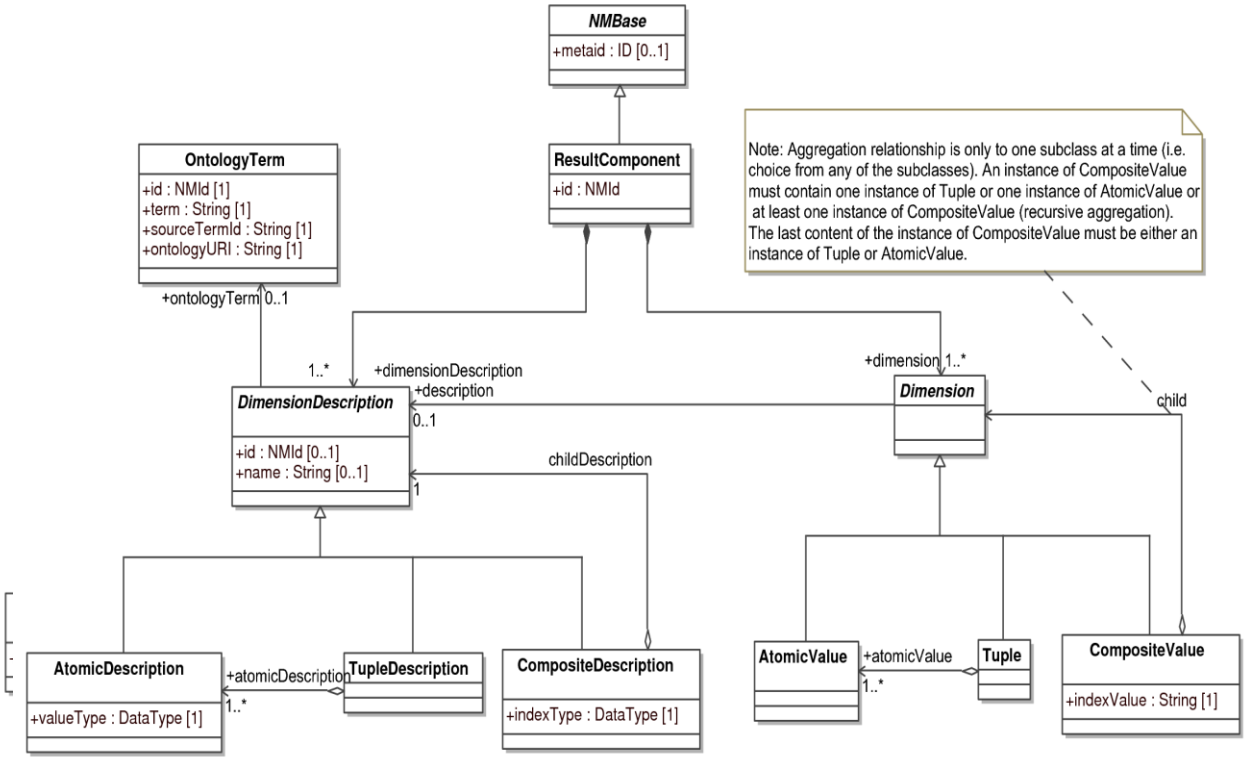


Figure 4.3: The definition of ResultComponent object class with its associated classes. A sequence of one or more instances of ResultComponent can be encoded in numl.

4.3.1 The id attribute

The required attribute id has data type NMId. The id attribute is used to give the instance of ResultComponent a unique identifier by which other parts of a NuML model definition can refer to it.

4.4 The dimension description

The structure of ResultComponent is described using the DimensionDescription abstract class (Figure 4.3) with three subclasses: AtomicDescription, TupleDescription and CompositeDescription. It has an optional id attribute of type NMId and optional name attribute of type string. It also has a relationship to the OntologyTerm object class, which allows the control vocabulary/ontology defined by the external ontology sources for the name attribute to be referenced.

4.5 The composite description

The nesting of dimensions is described using the CompositeDescription class, as defined in Figure 4.3. The CompositeDescription class is a subclass of DimensionDescription object class. It has an implicit relationship to CompositeValue class in that any result described in the CompositeDescription class must be placed in the CompositeValue class. It has a require indexType attribute of type DataType that defines the type of data of the indexValue attribute of CompositeValue class. An instance of CompositeDescription must contain exactly one instance of any of AtomicDescription, TupleDescription or CompositeDescription classes, as indicated by its aggregation relationship to the DimensionDescription.

4.6 The tuple description

Where results contain structured components that are not represented as distinct dimensions, the structure is described using the TupleDescription class, as presented in Figure 4.3. TupleDescription is subclass of DimensionDescription and serves as a container for the instances of AtomicDescription object class. It has an implicit relationship to Tuple class. Any result that is described in the TupleDescription class must be placed in the Tuple class. An instance of TupleDescription class must have at least one instance of AtomicDescription object class, and only one instance of TupleDescription is allowed within the CompositeDescription as described above. The id and name attributes inherited from the super class are optional for this class.

4.7 The atomic description

Where a value in a result can no longer be subdivided, it is described using the AtomicDescription class, as defined in Figure 4.3. The AtomicDescription class is a subclass of DimensionDescription class. It has an implicit relationship to AtomicValue class. Any result that is described in the instance of AtomicDescription class must be placed in the instance of AtomicValue object class. It has a require valueType attribute of type DataType that defines the type of data contained in the instance of AtomicValue class.

4.8 The dimension

A dimension within a result is represented using Dimension abstract class (Figure 4.3) with three subclasses: AtomicValue, Tuple and CompositeValue. The optional *description* relationship to the DimensionDescription object class allows explicit referencing of the instances of subclasses of DimensionDescription object class from the instances of the subclasses of Dimension object class.

4.9 The composite value

A nesting relationship between dimensions is represented using the CompositeValue class, as defined in Figure 4.3. The CompositeValue class is a subclass of Dimension class. It has an implicit relationship

to CompositeDescription class. Any result that is described in the CompositeDescription class is placed in the CompositeValue class. It has a require indexValue attribute of type string. The actual data type of indexValue is defined in the indexType attribute of the corresponding CompositeDescription class. An instance of CompositeValue must contain one instance of AtomicValue or one instance of Tuple or at least one instance of itself (recursive aggregation). The last content of the instance of a CompositeValue must be either an instance of Tuple or AtomicValue. Please see the note attached to the aggregation relationship in Figure 4.3.

4.10 The tuple

Structured values that are not described as dimensions are represented by the Tuple class, as presented in Figure 4.3. Tuple is a subclass of Dimension and serves as a container for instances of the AtomicValue class. It has an implicit relationship to TupleDescription. Any result that is described in the TupleDescription class is placed in the Tuple class. An instance of Tuple class must have at least one instance of AtomicValue object class and only one instance of Tuple is allowed within an instance of CompositeValue, as mentioned above.

4.11 The atomic value

Individual results within dimensions or tuples are represented by the AtomicValue class, as defined in Figure 4.3. The AtomicValue class is a subclass of Dimension object class. It has an implicit relationship to AtomicDescription object class. The data that is encoded in the instance of AtomicValue class must be described in the instance of AtomicDescription class.

5 Examples of numerical results encoded in NuML

In this section, we present several examples of encoding numerical results in XML using NuML Level 1. The first example is a complete example while the remaining examples give various ways of encoding numerical results in NuML without the ontotologyTerms section. Readers should be aware that the numerical results encoded in NuML must be associated with a particular model. It is the responsibility of the users of NuML to provide mechanisms for linking the result in NuML to the model that generated the results. SBRML (Dada et al., 2010) and SED-ML (Kohn & Le Novère, 2008) standardization efforts provide good ways of doing this. These examples are required to be associated with the model that gene and model.

5.1 Example using xpath as attribute value in composite description and composite value

```
<?xml version="1.0" encoding="UTF-8"?>
<numl version="1" level="1" xmlns="http://www.numl.org/numl/level1/version1">
       <ontologyTerms>
              <ontologyTerm id="term1" term="time" sourceTermId="SBO:0000345"
                     ontologyURI="http://www.ebi.ac.uk/sbo/" />
              <ontologyTerm id="term2" term="concentration"
                     sourceTermId="SBO:0000196" ontologyURI="http://www.ebi.ac.uk/sbo/" />
       </ontologyTerms>
       <resultComponent id="component1">
```

```
            <dimensionDescription>
                  <compositeDescription name="Time" ontologyTerm="term1"
                        indexType="double">
                        <compositeDescription name="Species" indexType="xpath">
                              <atomicDescription name="Concentration"
                                    ontologyTerm="term2" valueType="double" />
                        </compositeDescription>
                  </compositeDescription>
            </dimensionDescription>
            <dimension>
                  <compositeValue indexValue="0">
                        <compositeValue

      indexValue="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='x_CO2']">
                              <atomicValue>1</atomicValue>
                        </compositeValue>
                        <compositeValue

      indexValue="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='RuBP_ch']">
                              <atomicValue>0.33644</atomicValue>
                        </compositeValue>
                        <compositeValue

      indexValue="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PGA_ch']">
                              <atomicValue>3.35479</atomicValue>
                        </compositeValue>
                  </compositeValue>
                  <compositeValue indexValue="1">
                        <compositeValue

      indexValue="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='x_CO2']">
                              <atomicValue>0.66356</atomicValue>
                        </compositeValue>
                        <compositeValue

      indexValue="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='RuBP_ch']">
                              <atomicValue>8.90632e-31</atomicValue>
                        </compositeValue>
                        <compositeValue

      indexValue="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PGA_ch']">
                              <atomicValue>3.35479</atomicValue>
                        </compositeValue>
                  </compositeValue>
                  <resultComponent id="recomponet2"> ... </resultComponent>
            </dimension>
      </resultComponent>
</numl>
```

5.2 Example with tuple and atomic value

```
<numl>
  .....
      <resultComponent id="main_fitting_result">
        <dimensionDescription>
          <tupleDescription name="Main">
            <atomicDescription name="Objective Value" valueType="float" />
            <atomicDescription name="Root Mean Square" valueType="float" />
            <atomicDescription name="Standard Deviation" valueType="float" />
          </tupleDescription>
```

```
        </dimensionDescription>
        <dimension>
          <tuple>
            <atomicValue>12.5015</atomicValue>
            <atomicValue>0.158123</atomicValue>
            <atomicValue>0.159242</atomicValue>
          </tuple>
        </dimension>
      </resultComponent>
  .....
</numl>
```

5.3 Example with composite value(s) and atomic value

```
<numl>
  .....
    <resultComponent id="species_conc">
      <dimensionDescription>
        <compositeDescription name="species" indexType="string">
          <atomicDescription name="Concentration" ontologyTerm="term3" valueType="float" />
        </compositeDescription>
      </dimensionDescription>
      <dimension>
        <compositeValue indexValue="Phosphohomoserine">
          <atomicValue>141.063</atomicValue>
        </compositeValue>
        <compositeValue indexValue="Inorganic phosphate">
          <atomicValue>10000</atomicValue>
        </compositeValue>
        <compositeValue indexValue="Cysteine">
          <atomicValue>15</atomicValue>
        </compositeValue>
      </dimension>
    </resultComponent>
  .....
</numl>
```

5.4 Example with composite value, tuple and atomic value

```
<numl>
  .....
    <resultComponent id="species_con_pnumbers">
      <dimensionDescription>
        <compositeDescription name="species"  indexType="string">
          <tupleDescription>
            <atomicDescription name="Concentration" ontologyTerm="term3" valueType="double" />
            <atomicDescription name="Particle Numbers" ontologyTerm="term4" valueType="double" />
          </tupleDescription>
        </compositeDescription>
      </dimensionDescription>
      <dimension>
        <compositeValue indexValue="Phosphohomoserine">
          <tuple>
            <atomicValue>141.063</atomicValue>
            <atomicValue>8.49503e+19</atomicValue>
          </tuple>
        </compositeValue>
        <compositeValue indexValue="Inorganic_phosphate">
          <tuple>
            <atomicValue>10000</atomicValue>
```

```
            <atomicValue>6.02214e+21</atomicValue>
          </tuple>
        </compositeValue>
        <compositeValue indexValue="Cysteine">
          <tuple>
            <atomicValue>15</atomicValue>
            <atomicValue>9.03321e+18</atomicValue>
          </tuple>
        </compositeValue>
      </dimension>
    </resultComponent>
  .....
</numl>
```

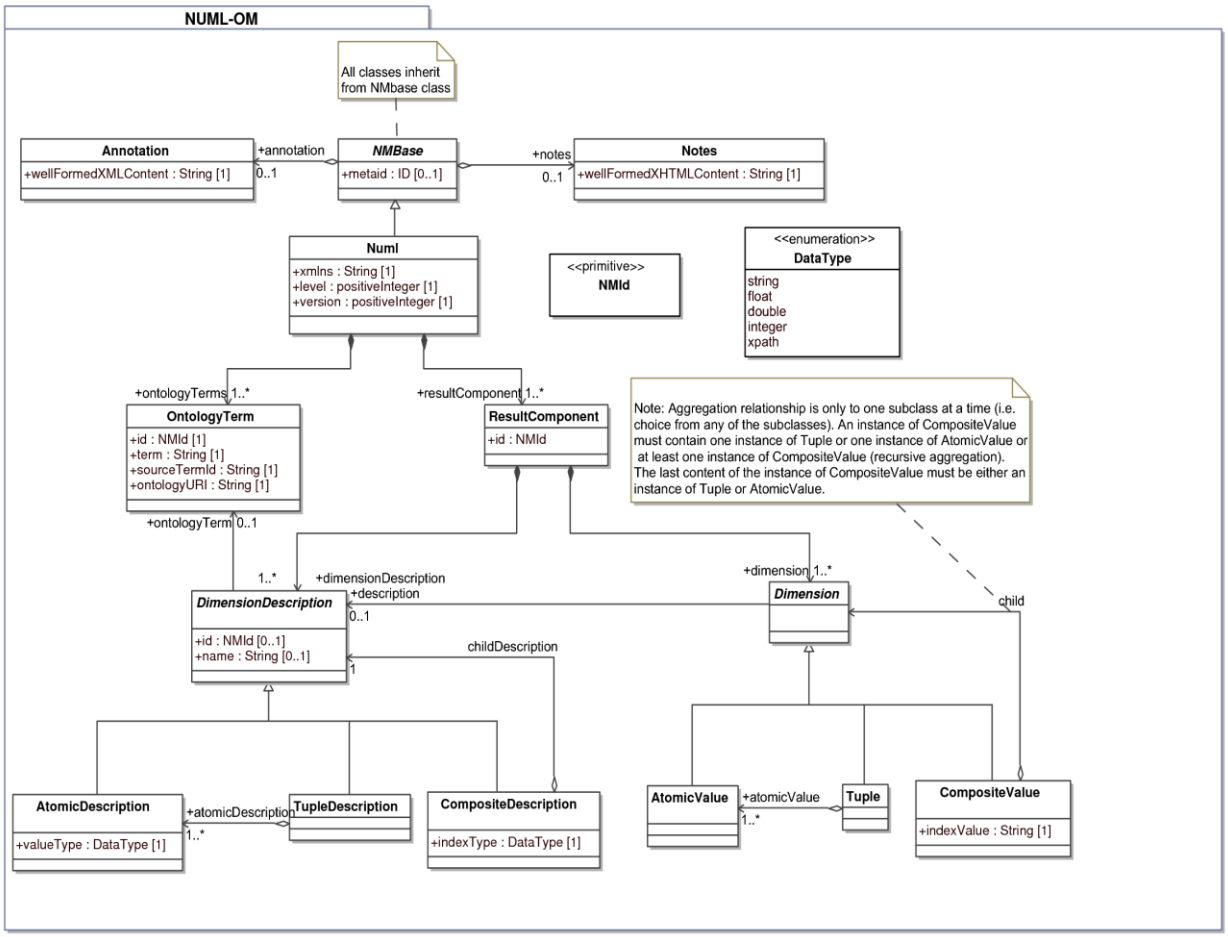
5.5 Online example of NuML

Please visit http://code.google.com/p/numl/ for more examples of simulation results encoded in NuML.

Acknowledgements

TODO

Appendix A: Complete NuML object model



Appendix B: XML Schema for NuML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.numl.org/numl/level1/version1"
	xmlns="http://www.numl.org/numl/level1/version1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
	xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">

	<!--Definition of identifier (NMId) type -->
	<xsd:simpleType name="NMId">
		<xsd:annotation>
			<xsd:documentation>
				The type NMId is used throughout Numerical Result Markup Language (NML)
as the type of the 'id' attributes on NML elements.
			</xsd:documentation>
		</xsd:annotation>
		<xsd:restriction base="xsd:string">
			<xsd:pattern value="(_|[a-z]|[A-Z])(_|[a-z]|[A-Z]|[0-9])*" />
		</xsd:restriction>
	</xsd:simpleType>

	<!-- Definition of DataType -->
	<xsd:simpleType name="DataType">
```

```
		<xsd:annotation>
			<xsd:documentation>
				The enumeration type "DataType" is used to specify the type of
"valueType" attribute of atomicDescription and "indexType" attribute of the compositeDescription
elements.
			</xsd:documentation>
		</xsd:annotation>
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="string" />
			<xsd:enumeration value="xpath" />
			<xsd:enumeration value="float" />
			<xsd:enumeration value="double" />
			<xsd:enumeration value="integer" />
		</xsd:restriction>
	</xsd:simpleType>

	<!--Definition of NMBase Type -->
	<xsd:complexType name="NMBase" abstract="true">
		<xsd:annotation>
			<xsd:documentation>
				The NMBase type is the base type of all main components in NML. It
supports attaching metadata, notes and annotations to components.
			</xsd:documentation>
		</xsd:annotation>
		<xsd:sequence>
			<xsd:element name="notes" minOccurs="0">
				<xsd:complexType>
					<xsd:sequence>
						<xsd:any namespace="http://www.w3.org/1999/xhtml"
processContents="skip" minOccurs="0" maxOccurs="unbounded" />
					</xsd:sequence>
				</xsd:complexType>
			</xsd:element>
			<xsd:element name="annotation" minOccurs="0">
				<xsd:complexType>
					<xsd:sequence>
						<xsd:any processContents="skip" minOccurs="0"
maxOccurs="unbounded" />
					</xsd:sequence>
				</xsd:complexType>
			</xsd:element>
		</xsd:sequence>
		<xsd:attribute name="metaid" type="xsd:ID" use="optional" />
	</xsd:complexType>

	<!-- Definition of NML elements -->
	<xsd:element name="resultComponent" type="ResultComponent" />

	<!-- Elements for data description -->
	<xsd:element name="dimensionDescription" type="DimensionDescription" />
	<xsd:element name="compositeDescription" type="CompositeDescription" />
	<xsd:element name="tupleDescription" type="TupleDescription" />
	<xsd:element name="atomicDescription" type="AtomicDescription" />

	<!-- Elements for data -->
	<xsd:element name="dimension" type="Dimension" />
	<xsd:element name="compositeValue" type="CompositeValue" />
	<xsd:element name="tuple" type="Tuple" />
	<xsd:element name="atomicValue" type="xsd:string" />

	<!-- Definition of ontologyTerm element for referencing external ontologies basically used
```

```
for data description -->
	<xsd:element name="ontologyTerm" type="OntologyTerm" />

	<!-- Definition of ontologyTerms element. This is container for all ontology terms used in
the NuML document -->
	<xsd:element name="ontologyTerms">
		<xsd:complexType>
			<xsd:sequence>
				<xsd:element ref="ontologyTerm" maxOccurs="unbounded" />
			</xsd:sequence>
	</xsd:complexType>
	</xsd:element>

	<!-- Definition of OntologyTerm type for data description -->
	<xsd:complexType name="OntologyTerm">
		<xsd:complexContent>
			<xsd:extension base="NMBase">
				<xsd:attribute name="id" use="required" type="NMId" />
				<xsd:attribute name="term" use="required" type="xsd:string" />
				<xsd:attribute name="sourceTermId" use="required" type="xsd:string" />
				<xsd:attribute name="ontologyURI" use="required" type="xsd:anyURI" />
			</xsd:extension>
		</xsd:complexContent>
	</xsd:complexType>

	<!--definition of result component type -->
	<xsd:complexType name="ResultComponent">
		<xsd:complexContent>
			<xsd:extension base="NMBase">
				<xsd:sequence>
					<xsd:element ref="dimensionDescription" />
					<xsd:element ref="dimension" />
				</xsd:sequence>
				<xsd:attribute name="id" use="required" type="NMId" />
				<xsd:attribute name="name" use="optional" type="xsd:string" />
			</xsd:extension>
		</xsd:complexContent>
	</xsd:complexType>

	<!-- definition of dimension description type -->
	<xsd:complexType name="DimensionDescription">
		<xsd:complexContent>
			<xsd:extension base="NMBase">
				<xsd:choice>
					<xsd:element ref="compositeDescription" maxOccurs="unbounded" />
					<xsd:element ref="tupleDescription" maxOccurs="unbounded" />
					<xsd:element ref="atomicDescription" maxOccurs="unbounded" />
				</xsd:choice>
			</xsd:extension>
		</xsd:complexContent>
	</xsd:complexType>

	<!-- Definition of dimension type -->
	<xsd:complexType name="Dimension">
		<xsd:complexContent>
			<xsd:extension base="NMBase">
				<xsd:choice>
					<xsd:element ref="compositeValue" maxOccurs="unbounded" />
					<xsd:element ref="tuple" maxOccurs="unbounded" />
					<xsd:element ref="atomicValue" maxOccurs="unbounded" />
				</xsd:choice>
```

```
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>

<!-- Definition of CompositeDescription type -->
<xsd:complexType name="CompositeDescription">
        <xsd:complexContent>
                <xsd:extension base="NMBase">
                        <xsd:choice>
                                <xsd:element ref="compositeDescription" />
                                <xsd:element ref="tupleDescription" />
                                <xsd:element ref="atomicDescription" />
                        </xsd:choice>
                        <xsd:attribute name="id" use="optional" type="NMId" />
                        <xsd:attribute name="name" use="required" type="xsd:string" />
                        <xsd:attribute name="ontologyTerm" use="optional" type="NMId" />
                        <xsd:attribute name="indexType" use="required" type="DataType" />
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>

<!-- Definition of TupleDescription type -->
<xsd:complexType name="TupleDescription">
        <xsd:complexContent>
                <xsd:extension base="NMBase">
                        <xsd:sequence>
                                <xsd:element ref="atomicDescription" maxOccurs="unbounded" />
                        </xsd:sequence>
                        <xsd:attribute name="id" use="optional" type="NMId" />
                        <xsd:attribute name="name" use="optional" />
                        <xsd:attribute name="ontologyTerm" use="optional" type="NMId" />
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>

<!-- Definition of AtomicDescription type -->
<xsd:complexType name="AtomicDescription">
        <xsd:complexContent>
                <xsd:extension base="NMBase">
                        <xsd:attribute name="id" use="optional" type="NMId" />
                        <xsd:attribute name="name" use="required" type="xsd:string" />
                        <xsd:attribute name="ontologyTerm" use="optional" type="NMId" />
                        <xsd:attribute name="valueType" use="required" type="DataType" />
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>

<!-- Definition of CompositeValue type -->
<xsd:complexType name="CompositeValue">
        <xsd:complexContent>
                <xsd:extension base="NMBase">
                        <xsd:choice>
                                <xsd:element ref="compositeValue" maxOccurs="unbounded" />
                                <xsd:element ref="tuple" />
                                <xsd:element ref="atomicValue" />
                        </xsd:choice>
                        <xsd:attribute name="description" use="optional" type="NMId" />
                        <xsd:attribute name="indexValue" use="required" type="xsd:string" />
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>
```

```
	<!-- Definition of Tuple type -->
	<xsd:complexType name="Tuple">
		<xsd:complexContent>
			<xsd:extension base="NMBase">
				<xsd:sequence>
					<xsd:element ref="atomicValue" maxOccurs="unbounded" />
				</xsd:sequence>
			</xsd:extension>
		</xsd:complexContent>
	</xsd:complexType>

	<!-- Definition of NML Type -->
	<xsd:complexType name="NML">
		<xsd:complexContent>
			<xsd:extension base="NMBase">
				<xsd:sequence>
					<xsd:element ref="ontologyTerms"/>
					<xsd:element ref="resultComponent" maxOccurs="unbounded" />
				</xsd:sequence>
				<xsd:attribute name="level" type="xsd:positiveInteger" use="required"
fixed="1" />
				<xsd:attribute name="version" type="xsd:positiveInteger" use="required"
fixed="1" />
			</xsd:extension>
		</xsd:complexContent>
	</xsd:complexType>

	<!--The (only) top-level element allowed in an NML document. -->
	<xsd:element name="nml" type="NML" />
</xsd:schema>
```

References

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/.

Bosak, J. and Bray, T. (1999). XML and the second-generation Web. Scientific American, 280(5):89-93.

Bray, T., D. Hollander, D., and Layman, A. (1999). Namespaces in XML. World Wide Web Consortium 14-January-1999. Available via the World Wide Web at http://www.w3.org/TR/1999/REC-xml-names-19990114/.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., and Maler, E. (2000). Extensible markup language (XML) 1.0 (second edition), W3C recommendation 6-October-2000. Available via the World Wide Web at http://www.w3.org/TR/1998/REC-xml-19980210.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (2004). Extensible markup language (XML) 1.0 (third edition), W3C recommendation 4-February-2004. Available via the World Wide Web at http://www.w3.org/TR/2004/REC-xml-20040204.

Dada, J. O., Spasic, I., Paton, N.W., Mendes, P. (2010). SBRML: a markup language for associating systems biology data with models, Bioinformatic 26(7), 932-938.

Hucka et al. (2003). SBML: The Systems Biology Markup Language: A medium for representation and exchange of biochemical network models. Bioiformatics 19, 524-531.

Hucka, M., Finney, A., Hoops, S., Keating, M., and Le Novère, N. (2007). Systems Biology Markup Language (SBML) Level 2: Structures and facilities for basic model definitions. Available via the World Wide Web at http://www.sbml.org.

Kohn, D. and Le Novère, N. (2008). SED-ML — An XML Format for the Implementation of the MIASE Guidelines, LNBI 5307, Springer–Verlag, Berlin Heidelberg, 176-190.

Le Novère et al. (2005). Minimum Information Requested In the Annotation of Biochemical Models (MIRIAM), Nature Biotechnology 23(12), 1509-1515.

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