The Distributions Package for SBML Level 3

Authors

Stuart L Moodie
moodie@ebi.ac.uk
EMBL-EBI
Hinxton, UK

Lucian P Smith
lpsmith@uw.edu
California Institute of Technology
Seattle, WA, USA

Contributors

Nicolas Le Novère
lenov@babraham.ac.uk
Babraham Institute
Babraham, UK

Darren Wilkinson
darren.wilkinson@ncl.ac.uk
University of Newcastle
Newcastle, UK

Maciej Swat
mjswat@ebi.ac.uk
EMBL-EBI
Hinxton, UK

Sarah Keating
skeating@ebi.ac.uk
EMBL-EBI
Hinxton, UK

Colin Gillespie
c.gillespie@ncl.ac.uk
University of Newcastle
Newcastle, UK

Version 0.16 (Draft)

April, 2015

DRAFT

Disclaimer: This is a working draft of the SBML Level 3 "distib" package proposal. It is not a normative document. Please send comments and other feedback to the mailing list:
sbml-distrib@lists.sourceforge.net.

Contents

1 Introduction and motivation 4
- 1.1 What is it? 4
- 1.2 Scope 4
- 1.3 This Document 4
- 1.4 Conventions used in this document 5

2 Background 6
- 2.1 Problems with current SBML approaches 6
- 2.2 Past work on this problem or similar topics 6
  - 2.2.1 The Newcastle Proposal 6
  - 2.2.2 Seattle 2010 6
  - 2.2.3 Hinxton 2011 7
  - 2.2.4 HARMONY 2012: Maastricht 8
  - 2.2.5 COMBINE 2012: Toronto 8
  - 2.2.6 2013 Package Working Group discussions 8
  - 2.2.7 HARMONY 2013: Connecticut 8

3 Proposed syntax and semantics 10
- 3.1 Overview 10
- 3.2 Namespace URI and other declarations necessary for using this package 10
- 3.3 Primitive data types 11
  - 3.3.1 Type `UncertId` 11
  - 3.3.2 Type `UncertIdRef` 11
- 3.4 Defining Distributions 11
  - 3.4.1 The approach 11
- 3.5 The extended **FunctionDefinition** class 12
- 3.6 The **DrawFromDistribution** class 13
- 3.7 The **DistribInput** class 13
- 3.8 The **Distribution** class 13
- 3.9 Discrete vs. continuous sampling 14
- 3.10 Truncation 14
- 3.11 Examples using the extended **FunctionDefinition** 14
  - 3.11.1 Defining and using a normal distribution with UncertML 14
  - 3.11.2 Defining a 'die roll' PMF with UncertML 15
  - 3.11.3 Defining a 'pick one' sample with UncertML 16
- 3.12 Equivalence with Fallback Function 18
- 3.13 The extended ***SBase*** class 19
- 3.14 The **Uncertainty** class 19
- 3.15 The **AbstractUncertainty** class 20
- 3.16 Examples using extended ***SBase*** 20
  - 3.16.1 Basic **AbstractUncertainty** example 20
  - 3.16.2 Defining a Random Variable 21

4 Interaction with other packages 23
- 4.1 Custom annotations for function definitions 23
- 4.2 The Arrays package 23
- 4.3 The Required Elements package 24

5 Use-cases and examples 25
- 5.1 Sampling from a distribution: PK/PD Model 25
- 5.2 Truncated distribution 27
- 5.3 Multivariate distribution 28

6 Prototype implementations 31

7 Acknowledgements 32

A Changes anticipated in UncertML 3.0 33

B UncertML Distributions 34

References 39

Revision History

Version	Date	Author	Comments
0.1 (Draft)	15 Oct 2011	Stuart Moodie	First draft
0.2 (Draft)	16 Oct 2011	Stuart Moodie	Added introductory text and background info. Other minor changes etc.
0.3 (Draft)	16 Oct 2011	Stuart Moodie	Filled empty invocation semantics section.
0.4 (Draft)	4 Jan 2012	Stuart Moodie	Incorporated comments from NlN, MS and SK. Some minor revisions and corrections.
0.5 (Draft)	6 Jan 2012	Stuart Moodie	Incorporated addition comments on aim of package from NlN.
0.6 (Draft)	19 Jul 2012	Stuart Moodie	Incorporated revisions discussed and agreed at HARMONY 2012.
0.7 (Draft)	6 Aug 2012	Stuart Moodie	Incorporated review comments from Maciej Swat and Sarah Keating.
0.8 (Draft)	21 Dec 2012	Stuart Moodie	Incorporated changes suggested at combine and subsequently through list discussions.
0.9 (Draft)	9 Jan 2013	Stuart Moodie	Incorporated corrections and comments from Maciej Swat and Sarah Keating.
0.10 (Draft)	10 Jan 2013	Stuart Moodie	Modified based on comments from MS.
0.11 (Draft)	17 May 2013	Lucian Smith	Modified based on Stuart's proposals and PWG discussion.
0.12 (Draft)	June 2013	Lucian Smith and Stuart Moodie	Modified based on HARMONY 2013 discussion.
0.13 (Draft)	July 2013	Lucian Smith and Stuart Moodie	Modified based PWG discussion, particularly with respect to UncertML.
0.14 (Draft)	March 2015	Lucian Smith	Modified to match UncertML 3.0.
0.15 (Draft)	March 2015	Lucian Smith and Sarah Keating	Modified to match UncertML 3.0 for real this time.
0.16 (Draft)	March 2015	Lucian Smith	Added information about UncertML 3.0 distributions, and the distributions custom annotations.

1 Introduction and motivation

1.1 What is it?

The Distributions package (also affectionately known as *distrib* for short) provides an extension to SBML Level 3 that enables a model to encode and sample from both discrete and continuous probability distributions, and provide the ability to annotate elements with information about the distribution their values were drawn from. Applications of the package include for instance descriptions of population based models: an important subset of which are pharmacokinetic/pharmacodynamic (PK/PD) models[1], which are used to model the action of drugs.

Note that originally the package was called Distributions and Ranges, but Ranges and the use of probability distributions to describe statistical uncertainly are no longer in the scope, hence the name change.

1.2 Scope

The Distributions package adds support to SBML for sampling from a probability distribution. In particular the following are in scope:

- Sampling from a continuous distribution.
- Sampling from a discrete distribution.
- Sampling from user-defined discrete probability density function.
- The specification of descriptive statistics (mean, standard deviation, standard error, etc.).

At one point the following were considered for inclusion in this package but are now **out of scope**:

- Sampling from user-defined probability density function.
- Stochastic differential equations.
- Other functions used to characterise a probability distribution, such as cumulative distribution functions (CDF) or survival functions, etc.

1.3 This Document

This proposal describes the consensus view of workshop participants and subscribers to the sbml-distrib mailing list. Although it was written by the listed authors it does not soley reflect their views nor is it their proposal. Rather, it is their understanding of the consensus view of what the Distributions package should do and how it should do it. The contributors listed have made significant contributions to the development and writing of this specification and are credited accordingly, but a more comprehensive attribution is provided in the acknowledgements (Section 7 on page 32).

Finally, the authors would encourage the reader to consider them and contribute their ideas or comments — indeed any feedback about this proposal — to the *distrib* discussion list[2].

Once the proposal is finalised this will be the first step towards the formal adoption of the *distrib* as a package in SBML Level 3. After this, two implementations based on this proposal are required and then the SBML editors must agree that the implementations and specification are complete. The proposal will then provide the basis for a future package specification document. More details of the SBML package adoption process can be found at: http://sbml.org/Documents/SBML_Development_Process.

[1] for more information see: http://www.pharmpk.com/.

[2] sbml-distrib@lists.sourceforge.net

1.4 Conventions used in this document

As we are early in the package proposal process there will be some parts of this proposal where there is no clear consensus on the correct solution or only recent agreement or agreement by a group which may not be representative
? of the SBML community as a whole. These cases are indicated by the question mark in the left margin (illustrated). The reader should pay particular attention to these points and ideally provide feedback, especially if they disagree with what is proposed. Similarly there will be points — especially as the proposal is consolidated — which are
☞ agreed, but which the reader should take note of and perhaps read again. These points are emphasised by the hand pointer in the left margin (illustrated).

2 Background

2.1 Problems with current SBML approaches

SBML Level 3 Core has no direct support for encoding random values within a model. Currently there is no workaround within the core language itself, although it is possible to define such information using annotations within SBML itself. Frank Bergmann had proposed such an semi-formalised extension for use with SBML L2 [REF?].

2.2 Past work on this problem or similar topics

2.2.1 The Newcastle Proposal

In 2005 there was a proposal from Colin Gillespie and others [3] to introduce support for probability distributions in the SBML core specification. This was based on their need to use such distributions to represent the models they were creating as part of the BASIS project (http://www.basis.ncl.ac.uk).

They proposed that distributions could be referred to in SBML using the **csymbol** element in the MathML subset used by the SBML Core specification. An example is below:

```
<xmlns=''http://www.w3.org/1998/Math/MathML''>
  <apply>
    <csymbol encoding=''text''
        definitionURL=''http://www.sbml.org/sbml/symbols/uniformRandom''>
      uniformRandom
    </csymbol>
    <ci>mu</ci>
    <ci>sigma</ci>
  </apply>
</math>
```

This required that a library of definitions be maintained as part of the SBML standard and in their proposal they defined an initial small set of commonly used distributions. The proposal was never implemented.

2.2.2 Seattle 2010

The "distrib" package was discussed at the Seattle SBML Hackathon[4] and this section is an almost verbatim reproduction of Darren Wilkinson's report on the meeting[5]. There Darren presented an overview of the problem[6][7], building on the old proposal from the Newcastle group (see above: Section 2.2.1). There was broad support at the meeting for development of such a package, and for the proposed feature set. Discussion following the presentation led to a consensus on the following points:

- There is an urgent need for such a package.
- It is important to make a distinction between a description of uncertainty regarding a model parameter and the mechanistic process of selecting a random number from a probability distribution, for applications such as parameter scans and experimental design
- It is probably worth including the definition of PMFs, PDFs and CDFs in the package
- It is worth including the definition of random distributions using particle representations within such a package, though some work still needs to be done on the precise representation

[3] http://sbml.org/Community/Wiki/SBML_Leve\T1\l_3_Proposals/Distributions_and_Ranges
[4] http://sbml.org/Events/Hackathons/The_2010_SBML-BioModels.net_Hackathon
[5] http://sbml.org/Forums/index.php?t=tree&goto=6141&rid=0
[6] Slides: http://sbml.org/images/3/3b/Djw-sbml-hackathon-2010-05-04.pdf
[7] Audio: http://sbml.org/images/6/67/Wilkinson-distributions-2010-05-04.mov

- It could be worth exploring the use of xinclude to point at particle representations held in a separate file
- Random numbers must not be used in rate laws or anywhere else that is continuously evaluated, as then simulation behaviour is not defined
- Although there is a need for a package for describing extrinsic noise via stochastic differential equations in SBML, such mechanisms should not be included in this package due to the considerable implications for simulator developers
- We probably don't want to layer on top of UncertML (www.uncertml.org), as this spec is fairly heavy-weight, and somewhat tangential to our requirements
- A random number seed is not part of a model and should not be included in the package
- The definition of truncated distributions and the specification of hard upper and lower bounds on random quantities should be considered.

It was suggested that new constructs should be introduced into SBML by the package embedded as user-defined functions using the following syntax:

```
<listOfFunctionDefinitions>
  <functionDefinition id="myNormRand">
    <distrib:####>
      #### distrib binding information here ####
    </distrib:####>
    <math>
      <lambda>
        <bvar>
          <ci>mu</ci>
          <ci>sigma</ci>
        </bvar>
        <ci>mu</ci>
      </lambda>
    </math>
  </functionDefinition>
</listOfFunctionDefinitions>
```

which allows the use of a "default value" by simulators which do not understand the package (but simulators which do will ignore the <math> element). The package would nevertheless be "required", as it will not be simulated correctly by software which does not understand the package.

Informal discussions following the break-out covered topics such as:

- how to work with vector random quantities in the absence of the vector element in the MathML subset used by SBML
- how care must be taken with the semantics of random variables and the need to both:
  - reference multiple independent random quantities at a given time
  - make multiple references to the same random quantity at a given time.

2.2.3 Hinxton 2011

Detailed discussion was continued at the Statistical Models Workshop in Hinxton in June 2011[8]. There those interested in representing Statistical Models in SBML came together to work out the details of how this package would work in detail. Dan Cornford from the UncertML project[9] attended the meeting and described how that resource could be used to describe uncertainty and in particular probability distributions. Perhaps the most

[8] http://sbml.org/Events/Other_Events/statistical_models_workshop_2011
[9] http://www.uncertml.org/

significant decision at this meeting was to adopt the UncertML resource as a controlled vocabulary that is referenced by the Distributions package.

Much has changed since this meeting, but the output from this meeting was the basis for the first version of this proposal.

2.2.4 HARMONY 2012: Maastricht

Two sessions were dedicated to discussion of Distributions at HARMONY based around the proposals described in version 0.5 of this document. In addition there was discussion about the Arrays proposal which was very helpful in solving the problem of multivariate distributions in Distributions. The following were the agreed outcomes of the meeting:

- The original proposal included UncertML markup directly in the function definition. This proved unwieldy and confusing and has been replaced by a more elegant solution that eliminates the UncertML markup and integrates well with the fallback function (see details below).
- Multivariate distributions can be supported using the Arrays package to define a covariance matrix.
- User defined continuous distributions would define a PDF in MathML.
- Usage semantics were clarified so that invokation of a function definition implied a value was sampled from the specified distribution.
- It was agreed from which sections of an SBML model a distribution could be invoked.
- Statistical descriptors of variables (for example mean and standard deviation) would be separated from Distributions and either provided in a new package or in a later version of SBML L3 core.

2.2.5 COMBINE 2012: Toronto

The August proposal was reviewed and an improvement was agreed to the user-defined PMF part of the proposal. In particular is was agreed that the categories should be defined by *distrib* classes rather than by passing in the information as an array. Questions were also raised about whether UncertML was suitably well defined to be used as an external definition for probability distributions. This was resolved subsequent to the meeting with a teleconference to Dan Cornford and colleagues. These changes are incorporated here. Finally, there was considerable debate about whether to keep the dependence of *distrib* on the Arrays package in order to support multi-variate distributions. The outcome was an agreement that we would review this at the end of 2012, based on the results of an investigation into how feasible it would be to implement Arrays as a package.

2.2.6 2013 Package Working Group discussions

Early 2013 saw a good amount of discussion on the *distrib* Package Working Group mailing list, spurred by proposals by Stuart Moodie[10]. While not all of his suggestions ended up being fully accepted by the group, several changes were accepted, including:

- To use UncertML as actual XML, instead of as a set of reference definitions.
- To use UncertML to encode descriptive statistics of SBML elements such as mean, standard deviation, standard error, etc.) bringing this capability back in scope for this package.

2.2.7 HARMONY 2013: Connecticut

At HARMONY at UConn in Connecticut, further discussions revealed the importance of distinguishing the ability to describe an element as a distributed variable vs. a function call within the model performing a draw from a distribution.

[10] http://thestupott.wordpress.com/2013/03/12/an-improved-distrib-proposal/

We also decided to discard the encoding of explicit PDFs for now, as support for it is remarkably complicated, and there no demand for it. The current design could be extended to support this feature so if there is demand for it in the future support for explicit PDFs could be reintroduced.

3 Proposed syntax and semantics

3.1 Overview

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

Class: Names of ordinary (concrete) classes begin with a capital letter and are printed in an upright, bold, sans-serif typeface. In electronic document formats, the class names are also hyperlinked to their definitions in this specification document. (However, as in the previous case, class names are not hyperlinked if they are for classes that are unchanged from their definitions in the SBML Level 3 Core specification.)

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

`[elementName]`: In some cases, an element may contain a child of any class inheriting from an abstract base class. In this case, the name of the element is indicated by giving the abstract base class name in brackets, meaning that the actual name of the element depends on whichever subclass is used, with capitalization following the capitalization of the name in brackets.

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

3.2 Namespace URI and other declarations necessary for using this package

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

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

In addition, SBML documents using a given package must indicate whether the package may be used to change the mathematical meaning of SBML Level 3 Version 1 Core elements. This is done using the attribute `required` on the `<sbml>` element in the SBML document. For the Distributions package, the value of this attribute must be "`true`", as the **DrawFromDistribution** element overrides the core definition of a **FunctionDefinition**. Note that the value of this attribute must *always* be set to "`true`", even if the particular model does not contain any **DrawFromDistribution** elements.

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

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

3.3 Primitive data types

The Distributions package uses the "`string`" primitive data type described in Section 3.1 of the SBML Level 3 Version 1 Core specification, and adds two additional primitive types described below.

3.3.1 Type `UncertId`

The type `UncertId` is derived from `SId` (SBML Level 3 Version 1 Core specification Section 3.1.7) and has identical syntax. The `UncertId` type is used to create local IDs that can be used in the extended **FunctionDefinition** objects to refer to the arguments of the function, in much the same way that the identities of the `bvar` elements are used in MathML `lambda` elements. Each `UncertId` has a scope local to the **DrawFromDistribution** in which it is found. The equality of `UncertId` values is determined by an exact character sequence match; i.e., comparisons of these identifiers must be performed in a case-sensitive manner.

3.3.2 Type `UncertIdRef`

Type `UncertIdRef` is used for references to identifiers of type `UncertId`. This type is derived from `UncertId`, but with the restriction that the value of an attribute having type `UncertIdRef` must match the value of a `UncertId` found in the same parent **DrawFromDistribution** as the `UncertIdRef`. As with `UncertId`, the equality of `UncertIdRef` values is determined by exact character sequence match; i.e., comparisons of these identifiers must be performed in a case-sensitive manner.

3.4 Defining Distributions

3.4.1 The approach

The Distributions package has two very simple purposes. First, it provides a mechanism for sampling a random value from a probability distribution. This implies that it must define the probability distribution and then must sample a random value from that distribution.

Secondly, it provides a mechanism for describing elements with information about their uncertainty. One common use case for this will be to provide the standard deviation for a value. Another may be describing a parameter's distribution, so that a better search can be performed in a parameter scan experiment.

Both purposes are achieved by using UncertML. Probability density functions (PDFs) are defined in the "`Distributions`" branch of UncertML, probability mass functions (PMFs) are defined in the "`Samples`" branch, and summary statistics in the "`Statistics`" branch.

It is technically possible to provide an explicit PDF in MathML instead of using the pre-defined PDFs from UncertML. However, one advantage of using the UncertML pre-defined distributions is that software can easily recognise the distribution and use an optimised built-in implementation rather than interpreting the distribution from the PDF

definitions. For some applications such optimisations make important performance differences. Another advantage is that some software may only support certain types of distributions, and having them predefined makes it simpler for the software to inform a user that a particular distribution is not supported.

It is hoped that if users find the need to define distributions not covered by UncertML, that they will either be able to encode those distributions as combinations of other predefined distributions, or that they will be able to persuade UncertML to add the new distribution to the list.

When a distribution is defined in a **FunctionDefinition**, it is sampled when it is invoked. To reuse a sampled value, the value must be assigned to a parameter first, such as through the use of an **InitialAssignment** or **EventAssignment**. When a distribution is defined elsewhere, that information may be used outside of the model, using whatever methodology is appropriate to answer the question being pursued.

3.5 The extended FunctionDefinition class

To model random processes, this package extends the **FunctionDefinition** class as can be seen in the UML representation in Figure 1. The redefined **FunctionDefinition** optionally contains a single `drawFromDistribution` child.

The **FunctionDefinition** class must still contain the MathML block containing the standard SBML function definition, because that element is required in SBML Level 3 Version 1 (in SBML Level 3 Version 2, this requirement has been dropped). This also ensures a degree of backwards compatibility for SBML readers and validators that do not understand the *distrib* package.

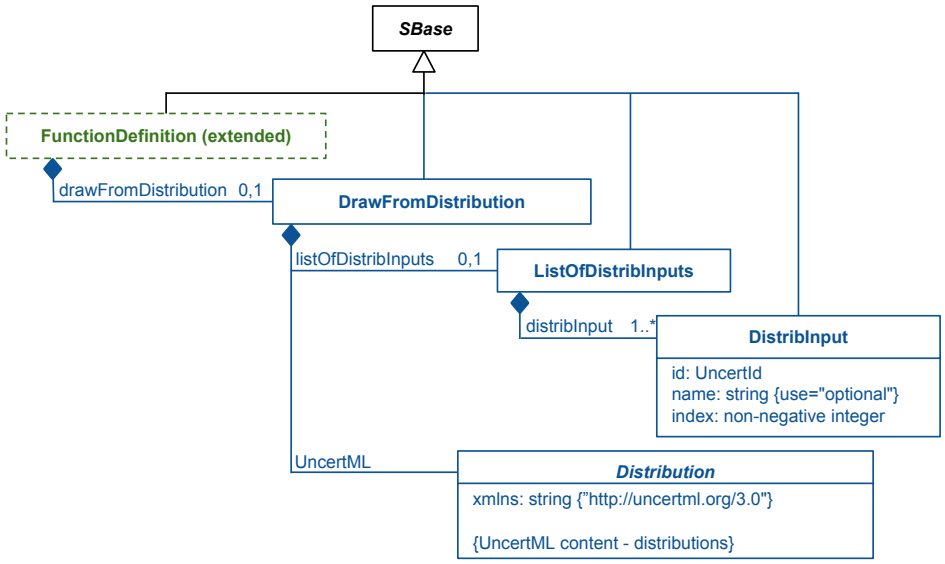

Figure 1: *The definition of the extended* ***FunctionDefinition*** *class, plus the* ***DrawFromDistribution***, ***ListOfDistribInputs***, ***DistribInput***, *and* ***Distribution*** *classes. A* ***DrawFromDistribution*** *element must have exactly one* ***Distribution*** *child. Together, these classes provide a way to transform a* ***FunctionDefinition*** *to sample from a distribution.*

As outlined above, the **FunctionDefinition** class is extended to contain a **DrawFromDistribution** child. Because a **FunctionDefinition** must have a **Lambda** child according to SBML Level 3 Version 1 Core, a valid function will still have one, but the `drawFromDistribution` child, if present, will override any definition found there. However, software that does not support *distrib* could potentially invoke the function found in the **Lambda** element (see Section 3.12).

In the **Model**, an extended **FunctionDefinition** may be used in any MathML to perform a draw from a distribution. This draw will be unique for every use of the **FunctionDefinition**, whether or not the draw is performed at the same

simulation time as a different draw (for example, if used in two different **InitialAssignment** elements).

3.6 The DrawFromDistribution class

As illustrated in Figure 1 on the preceding page, the **DrawFromDistribution** class may have a **ListOfDistribInputs** child, which must in turn contain one or more **DistribInput** children, which act as the arguments to the function–they serve the same role as the `bvar` elements of the **Lambda** child of a **FunctionDefinition**. The order of arguments is determined by the `index` attribute: the first argument (if any) must have an index of "`0`", the second of "`1`", etc.

It must also have a **Distribution** child, representing a probability density function (PDF) or probability mass function (PMF) defined by UncertML. Within the UncertML, the `UncertId`s defined by the **DistribInput** objects are used as the variables within the distribution.

3.7 The DistribInput class

The **DistribInput** class mimics the `bvar` elements of MathML lambda functions. It must have an `id` attribute of type `UncertId` and an `index` attribute of type `non-negative integer`. It may additionally have a `name` attribute of type `string`, which may be used in the same manner as other `name` attributes on SBML Level 3 Version 1 Core objects; please see Section 3.3.2 of the SBML Level 3 Version 1 Core specification for more information.

Each **DistribInput** element represents an argument to the function, and serves as a local identifier, referenced only by the UncertML in the sibling **Distribution** class. See the examples in Section 3.11 for more details.

Because the **Lambda** child of the **FunctionDefinition** is required, it must have the same number of `bvar` children as the **DrawFromDistribution** has **DistribInput** children. They do not, however, have to have the same IDs: the `bvar` ids are defined as being local to the **Lambda** function in much the same way that the **DistribInput** IDs are defined as being local to the **DrawFromDistribution** object.

Each `index` attribute on a **DistribInput** within a **ListOfDistribInputs** element must have a unique value, numbered consecutively from "`0`": if one **DistribInput** is present, its `index` value must be "`0`"; if there are two, they must have `index` values of "`0`" and "`1`", etc.

3.8 The Distribution class

The **Distribution** class is an `UncertML` element that contains a single element from the `Distributions` branch of UncertML. There are 29 'Distribution' elements derived from this class defined at `http://uncertml.org/dictionary`, with the namespace "`http://uncertml.org/3.0`". Note that as of this writing, UncertML 3.0 is only defined with an XML schema. However, the *distrib* Package Working Group has been in communication with the developers of UncertML, and feel certain that the change needed in UncertML to accomodate its use in this package (namely, the possible substitution of IDs for numbers) will be made for UncertML 3.0. The actual name of this element within an SBML document will be identical to the name of the derived class itself, i.e. "`NormalDistribution`", "`LogisticDistribution`", etc.

When a **Distribution** is encountered, its parent **FunctionDefinition** is defined as sampling from the defined distribution, and returning that sample. It may contain any number of `UncertIdRef` strings, each of which must correspond to an `UncertId` defined in a **DistribInput** in the same function.

The full list of the 29 distributions and how they can be used is provided in Section B. Four of these distributions (Dirichlet, Multinomial, Multivariate Normal, and Multivariate Student T) use vectors as both input and output. It is possible, if tedious, to provide vector input to these distributions by simply defining each element of the required vector as a numeric value or as an `UncertId`. However, it is not possible in SBML Level 3 Version 1 Core to take a single vector and simultaneously assign its values to different elements, or even to use a vector within MathML. While it would be theoretically possible to define new elements in this specification to work around this limitation, such capabilities are more obviously the domain of the Arrays package within SBML, or of SED-ML[11] (to set a

[11] `http://sed-ml.org/`

suite of element values). Unfortunately, as of this writing, the Arrays package has not yet been finalized, and many aspects of it have not been set. Therefore, it is left to the future finalized Arrays package to define how to utilize a **FunctionDefinition** that returns a vector, and how to define a **FunctionDefinition** that takes a vector as input. It is also possible for other individuals or groups to come up with custom annotations that define how to do this, and in fact, this is encouraged for any group that requires the use of any of these four distributions for their models. If no such definitions exist, however, any numerical results from the use of these functions remain undefined, and models using this technique are unsimulatable. (Such models may still be useful descriptions of certain situations, however.) A final possibility is that SED-ML could be extended to extract the function definition, perform the sampling itself, and use the resulting vector to assign initial values to certain elements.

☞ Note that the **ExplicitPMF** class in previous versions of this specification did not use UncertML, and instead defined its own way of listing samples. The functionality (with the addition of IDs instead of numbers in UncertML 3.0) is replicated in the CategoricalDistribution class of UncertML 3.0.

3.9 Discrete vs. continuous sampling

The `SId`s of **FunctionDefinition** elements can be used in SBML Level 3 Version 1 Core in both discrete and continuous contexts: **InitialAssignment**, **EventAssignment**, **Priority**, and **Delay** elements are all discrete, while **Rule**, **KineticLaw**, and **Trigger** elements are all continuous in time. For discrete contexts, the behavior of *distrib*-extended **FunctionDefinition** elements is well-defined: one or more random values are sampled from the distribution each time the function definition is invoked. Each invocation implies one sampling operation. In continuous contexts, however, their behavior is ill-defined. More information than is defined in this package (such as autocorrelation values or full conditional probabilities) would be required to make random sampling tractable in continuous contexts, and is beyond the scope of this version of the package. If some package is defined in the future that adds this information, or if custom annotations are provided that add this information, such models may become simulatable. However, this package does not define how to handle sampling in continuous contexts, and recommends against it: a warning may be produced by any software encountering the use of a *distrib*-extended **FunctionDefinition** in a continuous context. Assuming such models are desirable, and the information is not provided in a separate package, this information may be incorporated into a future version of this specification.

Any other package that defines new contexts for MathML will also either be discrete or continuous. Discrete situations (such as those defined in the Qualitative Models package) are, as above, well-defined. Continuous situations (as might arise within the Spatial Processes package, over space instead of over time) will most likely be ill-defined. Those packages must therefore either define for themselves how to handle *distrib*-extended **FunctionDefinition** elements, or leave it to some other package/annotation scheme to define how to handle the situation.

3.10 Truncation

In order to perform truncation, one or both ends of the distribution are cut off, and the remaining function is re-scaled so the area under the curve is once again 1.0. This capability is provided in UncertML directly, through the use of the "`truncationLowerInclusiveBound`" and "`truncationUpperInclusiveBound`" elements. As those element names imply, the first is used to truncate the distribution at a lower bound, and the second at an upper bound. The fact that both are inclusive mean that the value at that boundary may possibly be returned by the function. Care should thus be taken to ensure that (for example) if zero is a *non-inclusive* bound for one's model, that a value slightly larger than zero be used as the distribution's inclusive lower bound.

3.11 Examples using the extended FunctionDefinition

Several examples are given below that illustrate various uses of an extended **FunctionDefinition**.

3.11.1 Defining and using a normal distribution with UncertML

In the following example, a **FunctionDefinition** is extended to define a draw from an UncertML-defined normal distribution:

```
...
  <listOfFunctionDefinitions>
    <functionDefinition id="normal">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <!-- Overridden MathML -->
      </math>
      <distrib:drawFromDistribution>
         <distrib:listOfDistribInputs>
           <distrib:distribInput distrib:id="avg" distrib:index="0"/>
           <distrib:distribInput distrib:id="sd" distrib:index="1"/>
         </distrib:listOfDistribInputs>
         <UncertML xmlns="http://www.uncertml.org/3.0">
           <NormalDistribution definition="http://www.uncertml.org/distributions">
             <mean>
               <var varId="avg"/>
             </mean>
             <stddev>
               <var varId="sd"/>
             </stddev>
           </NormalDistribution>
         </UncertML>
      </distrib:drawFromDistribution>
    </functionDefinition>
  </listOfFunctionDefinitions>
...
```

Here, the **DistribInput** children of **DrawFromDistribution** define the local `UncertId`s "`avg`" and "`sd`", which are then used by the **Distribution** as the `mean` and `stddev` of a normal distribution, as defined by UncertML. This function could then be used anywhere the **FunctionDefinition** id "`normal`" can be used, as for example in an **InitialAssignment**:

```
...
  <listOfInitialAssignments>
    <initialAssignment symbol="y">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <ci> normal </ci>
          <ci> z </ci>
          <cn> 10 </cn>
        </apply>
      </math>
    </initialAssignment>
  </listOfInitialAssignments>
...
```

This use would apply a draw from a normal distribution with mean "`z`" and standard deviation "`10`" to the SBML element "`y`".

3.11.2 Defining a 'die roll' PMF with UncertML

In the following example, a **FunctionDefinition** is extended to define a draw from an UncertML-defined set of explicit PMFs:

```
...
  <listOfFunctionDefinitions>
    <functionDefinition id="rolld4">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <!-- Overridden MathML -->
      </math>
        <distrib:drawFromDistribution>
          <UncertML xmlns="http://www.uncertml.org/3.0">
            <CategoricalDistribution definition="http://www.uncertml.org/distributions">
              <categoryProb>
                <name>1</name>
                <prob>
                  <pVal>0.25</pVal>
                </prob>
              </categoryProb>
              <categoryProb>
                <name>2</name>
                <prob>
                  <pVal>0.25</pVal>
                </prob>
              </categoryProb>
              <categoryProb>
                <name>3</name>
                <prob>
                  <pVal>0.25</pVal>
                </prob>
              </categoryProb>
              <categoryProb>
                <name>4</name>
                <prob>
                  <pVal>0.25</pVal>
                </prob>
              </categoryProb>
            </CategoricalDistribution>
          </UncertML>
        </distrib:drawFromDistribution>
    </functionDefinition>
  </listOfFunctionDefinitions>
...
```

No inputs are provided. The four `categoryProb` children of the `CategoricalDistribution` all have equal values for their `prob` children, and sum to 1.0, as they must. Each `name` is therefore equally likely to be chosen, resulting in this function returning "`1`", "`2`", "`3`", or "`4`", each with equal probability.

3.11.3 Defining a 'pick one' sample with UncertML

In the following example, a **FunctionDefinition** is extended to define a draw from an UncertML-defined set of samples:

```
...
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core"
      xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
      level="3" version="1" distrib:required="true">
  <model>
    <listOfFunctionDefinitions>
      <functionDefinition id="pickone">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <lambda>
            <bvar>
              <ci> A </ci>
            </bvar>
            <bvar>
```

```
            <ci> B </ci>
          </bvar>
          <bvar>
            <ci> C </ci>
          </bvar>
          <bvar>
            <ci> D </ci>
          </bvar>
          <notanumber/>
        </lambda>
      </math>
      <distrib:drawFromDistribution>
        <distrib:listOfDistribInputs>
          <distrib:distribInput distrib:id="A" distrib:index="0"/>
          <distrib:distribInput distrib:id="B" distrib:index="1"/>
          <distrib:distribInput distrib:id="C" distrib:index="2"/>
          <distrib:distribInput distrib:id="D" distrib:index="3"/>
        </distrib:listOfDistribInputs>
        <UncertML xmlns="http://www.uncertml.org/3.0">
          <CategoricalDistribution definition="http://www.uncertml.org/distributions">
            <categoryProb>
              <name>A</name>
              <prob>
                <pVal>0.25</pVal>
              </prob>
            </categoryProb>
            <categoryProb>
              <name>B</name>
              <prob>
                <pVal>0.25</pVal>
              </prob>
            </categoryProb>
            <categoryProb>
              <name>C</name>
              <prob>
                <pVal>0.25</pVal>
              </prob>
            </categoryProb>
            <categoryProb>
              <name>D</name>
              <prob>
                <pVal>0.25</pVal>
              </prob>
            </categoryProb>
          </CategoricalDistribution>
        </UncertML>
      </distrib:drawFromDistribution>
    </functionDefinition>
  </listOfFunctionDefinitions>
  <listOfParameters>
    <parameter id="y"/>
  </listOfParameters>
  <listOfInitialAssignments>
    <initialAssignment symbol="y">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <ci> pickone </ci>
          <cn type="integer"> 1 </cn>
          <cn type="integer"> 3 </cn>
          <cn type="integer"> 4 </cn>
          <cn type="integer"> 7 </cn>
        </apply>
      </math>
    </initialAssignment>
  </listOfInitialAssignments>
</model>
```

```
</sbml>
...
```

In this example, the function 'pickone' is defined, with four arguments, "A", "B", "C", and "D". When called, each argument has an equal chance of being chosen as the return value. The parameter 'x' is initialized by calling this function with the arguments 1, 3, 4, and 7, each of which has an equally likely chance of being chosen.

? Stuart: This seems a bit idiosyncratic to me. Are you trying to get round the fact that SBML doesn't support strings? If so I'd leave that up to the modeller. They can map numeric values to their categories.

? Lucian: It may be idiosyncratic; I was trying to illustrate a case where variables were used in the values instead of the probabilities. Is there a better example you can think of? I've added an initial assignment that uses the function by way of illustration.

3.12 Equivalence with Fallback Function

The MathML definition directly contained by the **functionDefinition** is not used, but is required by SBML Level 3 Version 1 Core. To ensure the continued validity of the model, the following rules must be followed:

- the lambda function should have the same number of arguments as its equivalent distribution (defined by *distrib*).
- Each argument should match the type of the equivalent argument in the external function.
- The lambda function should have the same return type as the *sampled* distribution. For example, if a predefined PDF when sampled returns a scalar value, the dummy function should also do so.

Clearly, these rules can only be enforced by a *distrib*-aware validator.

In the following example, the fallback function is coded to simply return "mean", the first argument of the function. Note that the arguments have been given different local IDs ("mean" and "s" instead of "avg" and "sd"); their equivalence is based on order, not string matching.

```
      <functionDefinition id="normal">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <lambda>
            <bvar>
              <ci> mean </ci>
            </bvar>
            <bvar>
              <ci> s </ci>
            </bvar>
            <ci> mean </ci>
          </lambda>
        </math>
        <distrib:drawFromDistribution>
          <distrib:listOfDistribInputs>
            <distrib:distribInput distrib:id="avg" distrib:index="0"/>
            <distrib:distribInput distrib:id="sd" distrib:index="1"/>
          </distrib:listOfDistribInputs>
          <UncertML xmlns="http://www.uncertml.org/3.0">
            <NormalDistribution definition="http://www.uncertml.org/distributions">
              <mean>
                <var varId="avg"/>
              </mean>
              <stddev>
                <var varId="sd"/>
              </stddev>
            </NormalDistribution>
          </UncertML>
        </distrib:drawFromDistribution>
      </functionDefinition>
```

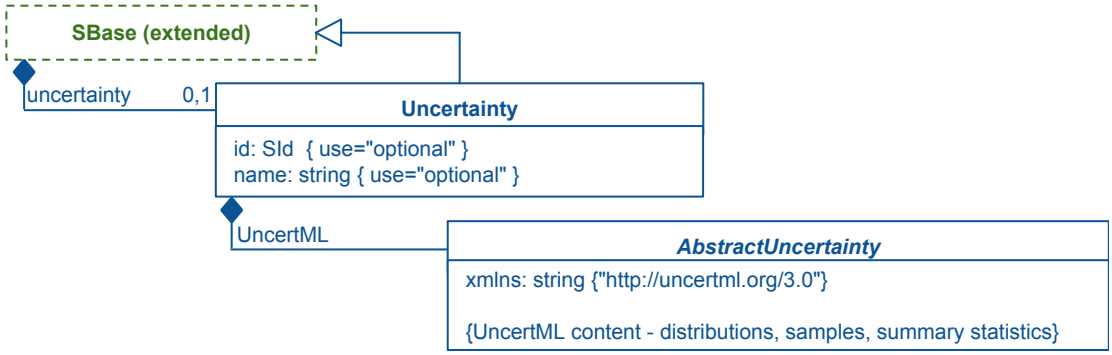

Figure 2: *The definition of the extended* ***SBase*** *class to include a new optional* ***Uncertainty*** *child, which in turn has an* ***AbstractUncertainty*** *child in the UncertML namespace. Intended for use with any element with mathematical meaning, or with a* ***Math*** *child.*

3.13 The extended *SBase* class

As can be seen in Figure 2, the SBML base class ***SBase*** is extended to include an optional **Uncertainty** child, which must contain an **AbstractUncertainty** child containing what information about the uncertainty of its parent element. In SBML Level 3 Version 1 Core, one should only extend those ***SBase*** elements with mathematical meaning (so, **Compartment**, **Parameter**, **Reaction**, **Species**, and **SpeciesReference**), or those ***SBase*** elements with **Math** children (so, **Constraint**, **Delay**, **EventAssignment**, **FunctionDefinition**, **InitialAssignment**, **KineticLaw**, **Priority**, **Rule**, and **Trigger**). This is added here to ***SBase*** instead of to each of these the various SBML elements so that other packages inherit the ability to extend their own elements in the same fashion: the **FluxBound** class from the Flux Balance Constraints package has mathematical meaning, for example, and can be given information about the distribution or set of samples from which it was drawn. Similarly, the **FunctionTerm** class from the Qualitative Models package has a **Math** child, which could be similarly extended.

A few SBML elements can interact in interesting ways that can confuse the semantics here. A **Reaction** element and its **KineticLaw** child, for example, both reference the exact same mathematics, and should therefore have the same **Uncertainty**. Similarly, if an **InitialAssignment** assigns to a constant element (**Parameter**, **Species**, etc.), the uncertainty for both should be the same, or only one should be provided.

Other elements not listed above should probably not be given an **Uncertainty** child, as it would normally not make sense to talk about the uncertainty of something that doesn't have a corresponding mathematical meaning. However, because packages or annotations can theoretically give new meaning (including mathematical meaning) to elements that previously did not have them, this is not a requirement.

It is important to note that the uncertainty described by the **Uncertainty** class is defined as being the uncertainty at the moment the element's mathematical meaning is calculated, and does not describe the uncertainty of how that element changes over time. For a **Species**, **Parameter**, **Compartment**, and **SpeciesReference**, this means that it is the uncertainty of their initial values, and does not describe the uncertainty in how those values evolve in time. The reason for this is that other SBML constructs all describe how (or if) the values change in time, and it is those other constructs that should be used to describe a symbol's time-based uncertainty. For example, a **Species** whose initial value had uncertainty due to instrument precision could have an **Uncertainty** child describing this. A **Species** whose value was known to change over time due to unknown processes, but which had a known average and standard deviation could be given an **AssignmentRule** that set that **Species** amount to the known average, and the **AssignmentRule** itself could be given an **Uncertainty** child describing the standard deviation of the variability.

3.14 The Uncertainty class

The **Uncertainty** class is a container for **AbstractUncertainty** (from UncertML) that descibes the uncertainty in the parent element's mathematical meaning.

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

3.15 The AbstractUncertainty class

The **AbstractUncertainty** class is defined as describing the uncertainty in its parent element's mathematics. It is the abstract base class, in UncertML, for that language's `Distribution`, `Sample`, and `Statistics` elements. This means it has the capability to provide anything from a full distribution to a `Sample` element to summary statistics such as `StandardDeviation` and `Mean`.

For convenience, UncertML provides a `StatisticsCollection`, which may be used to collect multiple other elements. For clarity, if multiple UncertML elements are present in a `StatisticsCollection`, they should not conflict with each other.

The namespace for IDs used in the UncertML is the SId namespace of elements with mathematical meaning in the model, including from other packages. If that other package is not understood by an interpreter, the **AbstractUncertainty** element may be ignored. If an interpreter does not understand an ID and cannot tell whether that ID came from a not-understood package, it may issue a warning.

Note that the described uncertainty for elements that change in value over time apply only to the element's uncertainty at a snapshot in time, and not the uncertainty in how it changes in time. For typical simulations, this means the element's initial condition. Note too that the description of the uncertainty of a **Species** should describe the uncertainty of its `amount`, not the uncertainty of its `concentration`. The 'primary' mathematical meaning of a **Species** in SBML is always the amount; the concentration may be used, but is considered to be derived.

3.16 Examples using extended *SBase*

Several examples are given to illustrate the use of the **Uncertainty** class:

3.16.1 Basic AbstractUncertainty example

In this examples, a species is given an **Uncertainty** child to describe its standard deviation:

```
...
      <species id="S1" compartment="C" initialAmount="3.22" hasOnlySubstanceUnits="true"
               boundaryCondition="false" constant="false">
        <distrib:uncertainty>
          <UncertML xmlns="http://www.uncertml.org/3.0">
            <StandardDeviation definition="http://www.uncertml.org/distributions">
              <value>
                <prVal>0.3</prVal>
              </value>
            </StandardDeviation>
          </UncertML>
        </distrib:uncertainty>
      </species>
...
```

Here, the species with an initial amount of 3.22 is described as having a standard deviation of 0.3, a value that might be written as "`3.22 ± 0.3`". This is probably the simplest way to use the package to introduce facts about the uncertainty of the measurements of the values present in the model.

It is also possible to include additional information about the species, should more be known:

```
...
```

```
      <species id="S1" compartment="C" initialAmount="3.22" hasOnlySubstanceUnits="true"
               boundaryCondition="false" constant="false">
        <distrib:uncertainty>
          <UncertML xmlns="http://www.uncertml.org/3.0">
            <StatisticsCollection definition="http://www.uncertml.org/statistics">
              <Mean definition="http://www.uncertml.org/statistics">
                <value>
                  <rVal>3.2</rVal>
                </value>
              </Mean>
              <StandardDeviation definition="http://www.uncertml.org/statistics">
                <value>
                  <prVal>0.3</prVal>
                </value>
              </StandardDeviation>
              <Variance definition="http://www.uncertml.org/statistics">
                <value>
                  <prVal>0.09</prVal>
                </value>
              </Variance>
            </StatisticsCollection>
          </UncertML>
        </distrib:uncertainty>
      </species>
...
```

In this example, the initial amount of 3.22 is noted as having a mean of 3.2, a standard deviation of 0.3, and a variance of 0.09. Since there is more than one piece of information included, a `StatisticsCollection` UncertML element is used. Note that the standard deviation can be calculated from the variance (or visa versa), but the modeler has chosen to include both here for convenience. Note too that this use of the **Uncertainty** element does not imply that the species amount comes from a normal distribution with a mean of 3.2 and standard deviation of 0.3, but rather that the species amount comes from an unknown distribution with those qualities. If it is known that the value was drawn from a particular distribution, that distribution should be used, rather than the `Mean` and `StandardDeviation` statistical values.

Note also that 3.22 (the `initialAmount`) is different from 3.2 (the `Mean`): evidently, this model was constructed as a realization of the underlying uncertainty, instead of trying to capture the single most likely model of the underlying process.

3.16.2 Defining a Random Variable

In addition to describing the uncertainty about an experimental observation one can also use this mechanism to describe a parameter as a random variable. In the example below the parameter, Z, is defined as following a normal distribution, with a given mean and variance. No value is given for the parameter so it is then up the modeller to decide how to use this random variable. For example they may choose to simulate the model in which case they may provide values for mu_Z and var_Z and then sample a random value from the simulation. Alternatively they may choose to carry out a parameter estimation and use experimental observations to estimate mu_Z and var_Z.

```
    <listOfParameters>
      <parameter id="mu_Z" value="10" constant="true"/>
      <parameter id="var_Z" value="0.1" constant="true"/>
      <parameter id="Z" constant="true">
        <distrib:uncertainty>
          <UncertML xmlns="http://www.uncertml.org/3.0">
            <NormalDistribution definition="http://www.uncertml.org/distributions">
              <mean>
                <var varId="mu_Z"/>
              </mean>
              <variance>
                <var varId="var_Z"/>
              </variance>
```

```
            </NormalDistribution>
          </UncertML>
        </distrib:uncertainty>
      </parameter>
    </listOfParameters>
```

? Stuart:This example illustrates the kind of use cases I was describing at HARMONY. I hope this isn't controversial, but I want us to be clear that we all happy using this interpretation of uncertainty too. If so then it makes sense to have an example.

4 Interaction with other packages

4.1 Custom annotations for function definitions

Before this package was available, a collection of SBML simulator authors developed an *ad-hoc* convention for exchanging annotated **FunctionDefinition** objects that represented draws from distributions. This convention is described by Frank T. Bergmann at https://docs.google.com/file/d/0B_wMqVOQLkZ3TVZHblNNRWgzNTg/, and represents a basic starting point for any modeler interested in exchanging SBML models containing draws from distributions.

When implementing Distributions support, it would be possible to include 'backwards' support for this annotation convention by annotating any **FunctionDefinition** using UncertML to also include these annotations, where appropriate.

The following table is taken from the above document by Frank Bergmann, and can be used to annotate **FunctionDefinition** elements that have been extended by Distributions to perform the same functions, providing the arguments are presented in the same order. The suggested fallback function returns the mean of the distribution.

Id	Name	URL	Fallback
uniform	Uniform distribution	http://en.wikipedia.org/wiki/Uniform_distribution_(continuous)	$lambda(a, b, \frac{a+b}{2})$
normal	Normal distribution	http://en.wikipedia.org/wiki/Normal_distribution	$lambda(m, s, m)$
exponential	Exponential distribution	http://en.wikipedia.org/wiki/Exponential_distribution	$lambda(l, \frac{1}{l})$
gamma	Gamma distribution	http://en.wikipedia.org/wiki/Gamma_distribution	$lambda(a, b, a * b)$
poisson	Poisson distribution	http://en.wikipedia.org/wiki/Poisson_distribution	$lambda(mu, mu)$
lognormal	Lognormal distribution	http://en.wikipedia.org/wiki/Log-normal_distribution	$lambda(z, s, e^{z+\frac{s^2}{2}})$
chisq	Chi-squared distribution	http://en.wikipedia.org/wiki/Chi-squared_distribution	$lambda(nu, nu)$
laplace	Laplace distribution	http://en.wikipedia.org/wiki/Laplace_distribution	$lambda(a, 0)$
cauchy	Cauchy distribution	http://en.wikipedia.org/wiki/Cauchy_distribution	$lambda(a, a)$
rayleigh	Rayleigh distribution	http://en.wikipedia.org/wiki/Rayleigh_distribution	$lambda(s, s * \sqrt{\pi/2})$
binomial	Binomial distribution	http://en.wikipedia.org/wiki/Binomial_distribution	$lambda(p, n, p * n)$
bernoulli	Bernoulli distribution	http://en.wikipedia.org/wiki/Bernoulli_distribution	$lambda(p, p)$

4.2 The Arrays package

This package is dependent on no other package, but relies on the Arrays package to provide vector and matrix structures if those are desired/used.

4.3 The Required Elements package

If the Required Elements package is used, any **FunctionDefinition** that has been given an **DrawFromDistribution** child must be given a **ChangedMath** child referencing this package's namespace. If the fallback function provides a complete **Lambda** function, its `viableWithoutChange` atribute *may* be set "`true`" if the modeler considers that function an acceptable alternative to the draw from the distribution, otherwise it must be set "`false`". The **Uncertainty** class does not affect the mathematics of any element, so no other element may be given a **ChangedMath** child referencing this package's namespace.

5 Use-cases and examples

The following examples are more fleshed out than the ones in the main text, and/or illustrate features of this package that were not previously illustrated.

5.1 Sampling from a distribution: PK/PD Model

This is a very straightforward use of an UncertML-defined distribution. The key point to note is that a value is sampled from the distribution and assigned to a variable when it is invoked in the initialAssignments element in this example. Later use of the variable does not result in re-sampling from the distribution. This is consistent with current SBML semantics.

? Stuart:I'd like to add another example here that defines the model without sampling. We can have 2 versions of the same model. I'll come back to it...

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core"
      xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
      level="3" version="1" distrib:required="true">
  <model>
    <listOfFunctionDefinitions>
      <functionDefinition id="logNormal">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <lambda>
            <bvar>
              <ci> scale </ci>
            </bvar>
            <bvar>
              <ci> shape </ci>
            </bvar>
            <notanumber/>
          </lambda>
        </math>
        <distrib:drawFromDistribution>
          <distrib:listOfDistribInputs>
            <distrib:distribInput distrib:id="scale" distrib:index="0"/>
            <distrib:distribInput distrib:id="shape" distrib:index="1"/>
          </distrib:listOfDistribInputs>
          <UncertML xmlns="http://www.uncertml.org/3.0">
            <LogNormalDistribution definition="http://www.uncertml.org/distributions">
              <logScale>
                <var varId="scale"/>
              </logScale>
              <shape>
                <var varId="shape"/>
              </shape>
            </LogNormalDistribution>
          </UncertML>
        </distrib:drawFromDistribution>
      </functionDefinition>
    </listOfFunctionDefinitions>
    <listOfCompartments>
      <compartment id="central" size="0" constant="true"/>
      <compartment id="gut" size="0" constant="true"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="Qc" compartment="central" initialAmount="1" hasOnlySubstanceUnits="true"
               boundaryCondition="false" constant="false"/>
      <species id="Qg" compartment="gut" initialAmount="1" hasOnlySubstanceUnits="true"
               boundaryCondition="false" constant="false"/>
    </listOfSpecies>
    <listOfParameters>
```

```
      <parameter id="ka" value="1" constant="true"/>
      <parameter id="ke" value="1" constant="true"/>
      <parameter id="Cc" value="1" constant="false"/>
      <parameter id="Cc_obs" value="1" constant="false"/>
    </listOfParameters>
    <listOfInitialAssignments>
      <initialAssignment symbol="central">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply>
            <ci> logNormal </ci>
            <cn> 0.5 </cn>
            <cn> 0.1 </cn>
          </apply>
        </math>
      </initialAssignment>
      <initialAssignment symbol="ka">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply>
            <ci> logNormal </ci>
            <cn> 0.5 </cn>
            <cn> 0.1 </cn>
          </apply>
        </math>
      </initialAssignment>
      <initialAssignment symbol="ke">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply>
            <ci> logNormal </ci>
            <cn> 0.5 </cn>
            <cn> 0.1 </cn>
          </apply>
        </math>
      </initialAssignment>
    </listOfInitialAssignments>
    <listOfRules>
      <assignmentRule variable="Cc">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply>
            <divide/>
            <ci> Qc </ci>
            <ci> central </ci>
          </apply>
        </math>
      </assignmentRule>
      <assignmentRule variable="Cc_obs">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply>
            <plus/>
            <ci> Cc </ci>
            <cn type="integer"> 1 </cn>
          </apply>
        </math>
      </assignmentRule>
    </listOfRules>
    <listOfReactions>
      <reaction id="absorption" reversible="false" fast="false">
        <listOfReactants>
          <speciesReference species="Qg" stoichiometry="1" constant="true"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="Qc" stoichiometry="1" constant="true"/>
        </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
              <times/>
```

```
              <ci> ka </ci>
              <ci> Qg </ci>
            </apply>
          </math>
        </kineticLaw>
      </reaction>
      <reaction id="excretion" reversible="false" fast="false">
        <listOfReactants>
          <speciesReference species="Qc" stoichiometry="1" constant="true"/>
        </listOfReactants>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
              <divide/>
              <apply>
                <times/>
                <ci> ke </ci>
                <ci> Qc </ci>
              </apply>
              <ci> central </ci>
            </apply>
          </math>
        </kineticLaw>
      </reaction>
    </listOfReactions>
  </model>
</sbml>
```

5.2 Truncated distribution

To encode a truncated distribution we may use the optional `truncationLowerInclusiveBound` as well as the `truncationUpperInclusiveBound` child elements of the `NormalDistribution` element. Many other UncertML distributions have these optional child elements as well. Note that the values for truncation are here provided as arguments to the function definition, but could instead be hard-coded.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core"
      xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
      level="3" version="1" distrib:required="true">
  <model>
    <listOfFunctionDefinitions>
      <functionDefinition id="truncatedNormal">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <lambda>
            <bvar>
              <ci> mu </ci>
            </bvar>
            <bvar>
              <ci> sigma </ci>
            </bvar>
            <bvar>
              <ci> lower </ci>
            </bvar>
            <bvar>
              <ci> upper </ci>
            </bvar>
            <notanumber/>
          </lambda>
        </math>
        <distrib:drawFromDistribution>
          <distrib:listOfDistribInputs>
            <distrib:distribInput distrib:id="mu" distrib:index="0"/>
            <distrib:distribInput distrib:id="sigma" distrib:index="1"/>
```

```
            <distrib:distribInput distrib:id="lower" distrib:index="2"/>
            <distrib:distribInput distrib:id="upper" distrib:index="3"/>
          </distrib:listOfDistribInputs>
          <UncertML xmlns="http://www.uncertml.org/3.0">
            <NormalDistribution definition="http://www.uncertml.org/distributions">
              <mean>
                <var varId="mu"/>
              </mean>
              <variance>
                <var varId="sigma"/>
              </variance>
              <truncationLowerInclusiveBound>
                <var varId="lower"/>
              </truncationLowerInclusiveBound>
              <truncationUpperInclusiveBound>
                <var varId="upper"/>
              </truncationUpperInclusiveBound>
            </NormalDistribution>
          </UncertML>
        </distrib:drawFromDistribution>
      </functionDefinition>
    </listOfFunctionDefinitions>
    <listOfParameters>
      <parameter id="V" constant="true"/>
      <parameter id="V_pop" value="105" constant="true"/>
      <parameter id="V_omega" value="0.7" constant="false"/>
      <parameter id="V_lower" value="15" constant="false"/>
      <parameter id="V_upper" value="150" constant="false"/>
    </listOfParameters>
    <listOfInitialAssignments>
      <initialAssignment symbol="V">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply>
            <ci> truncatedNormal </ci>
            <ci> V_pop </ci>
            <ci> V_omega </ci>
            <ci> V_lower </ci>
            <ci> V_upper </ci>
          </apply>
        </math>
      </initialAssignment>
    </listOfInitialAssignments>
  </model>
</sbml>
```

5.3 Multivariate distribution

In this example two correlated parameters are sampled from a multivariate distribution. The correlation is defined using a covariance matrix and the sampled values are returned as a vector of 2 values, and assigned to the variable "`correlated_params`". This vector is then used to assign values to "`V`" and "`C1`", thereby associating those two values with the same draw from the multivariate normal distribution. The use of various array and matrix MathML here is speculative: in the absence of a finalized Arrays package, it is impossible to tell exactly what form that will take. However, all of the functionality expressed here will need to be incorported in some form into the Arrays package, and much if not all of it may take the form illustrated here.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
      xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
      distrib:required="true"
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1"
      arrays:required="true">
  <!-- NOTE: This requires the arrays package! -->
  <model id="MultivariateExample" name="Multivariate Example">
```

```
    <listOfFunctionDefinitions>
      <functionDefinition id="multivariateNormal">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <lambda>
            <bvar>
              <ci> meanVec </ci>
            </bvar>
            <bvar>
              <ci> covarianceM </ci>
            </bvar>
            <ci> meanVector </ci>
          </lambda>
        </math>
        <distrib:drawFromDistribution>
          <distrib:listOfDistribInputs>
            <distrib:distribInput distrib:id="meanVec" distrib:index="0"/>
            <distrib:distribInput distrib:id="covarianceM" distrib:index="1"/>
          </distrib:listOfDistribInputs>
          <UncertML xmlns="http://www.uncertml.org/3.0">
            <MultivariateNormalDistribution definition="http://www.uncertml.org/distributions">
              <meanVector arrayVar="meanVec"/>
              <covarianceMatrix>
                <values arrayVar="covarianceM"/>
              </covarianceMatrix>
            </MultivariateNormalDistribution>
          </UncertML>
        </distrib:drawFromDistribution>
      </functionDefinition>
    </listOfFunctionDefinitions>
    <listOfParameters>
      <parameter id="V" constant="false"/>
      <parameter id="V_pop" value="105" constant="true"/>
      <parameter id="V_omega" value="0.70" constant="true"/>
      <parameter id="Cl" constant="true"/>
      <parameter id="Cl_pop" value="73" constant="true"/>
      <parameter id="Cl_omega" value="0.70" constant="true"/>
      <parameter id="covariance" constant="true">
        <arrays:listOfDimensions>
          <arrays:dimension id="i" lowerLimit="1" upperLimit="2" />
          <arrays:dimension id="j" lowerLimit="1" upperLimit="2" />
        </arrays:listOfDimensions>
      </parameter>
      <parameter id="correlated_means" constant="true">
        <arrays:listOfDimensions>
          <arrays:dimension id="i" lowerLimit="1" upperLimit="2" />
        </arrays:listOfDimensions>
      </parameter>
      <parameter id="correlated_params" constant="true">
        <arrays:listOfDimensions>
          <arrays:dimension id="i" lowerLimit="1" upperLimit="2" />
        </arrays:listOfDimensions>
      </parameter>
    </listOfParameters>
    <listOfInitialAssignments>
      <initialAssignment symbol="covariance">
        <math xmlns="http://www.w3.org/1998/Math/MathML"
              xmlns:sbml="http://www.sbml.org/sbml/level3/version1/arraymaths/version1">
          <!-- This is an unresolved issue. The l3 V1 Core mathml subset does not support
              matrices. One solution - as above is to use a different MathML subset definition.
         -->
          <matrix>
            <matrixrow>
              <apply><times/><ci>V_omega</ci><ci>V_omega</ci></apply>
              <apply><times/><ci>V_omega</ci><ci>C_omega</ci><ci>V_C_rho</ci></apply>
            </matrixrow>
            <matrixrow>
```

```
              <ci>0</ci>
              <apply><times/><ci>C_omega</ci><ci>C_omega</ci></apply>
            </matrixrow>
          </matrix>
        </math>
      </initialAssignment>
      <initialAssignment symbol="correlated_means">
        <math xmlns="http://www.w3.org/1998/Math/MathML"
              xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
          <vector>
            <ci>V_pop</ci>
            <ci>C_pop</ci>
          </vector>
        </math>
      </initialAssignment>
      <initialAssignment symbol="correlated_params" >
        <math xmlns="http://www.w3.org/1998/Math/MathML"
              xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
          <apply>
              <ci>multivariateNormal</ci>
              <ci>correlated_means</ci>
              <ci>covariance</ci>
          </apply>
        </math>
      </initialAssignment>
      <initialAssignment symbol="V">
        <math xmlns="http://www.w3.org/1998/Math/MathML"
              xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
          <apply>
            <selector/>
            <ci>correlated_params</ci>
            <cn type="integer">1</cn>
          </apply>
        </math>
      </initialAssignment>
      <initialAssignment symbol="Cl">
        <math xmlns="http://www.w3.org/1998/Math/MathML"
              xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
          <apply>
            <selector/>
            <ci>correlated_params</ci>
            <cn type="integer">2</cn>
          </apply>
        </math>
      </initialAssignment>
    </listOfInitialAssignments>
    <!-- This is an incomplete model snippet, sufficient to illustrate the use of
         a multivariate distribution. -->
  </model>
</sbml>
```

6 Prototype implementations

As of this writing (March 2015), libsbml has full support for elements defined by Distributions, as well as limited support for UncertML. Antimony (`http://antimony.sf.net/`) has support for a limited number of UncertML function (normal, uniform, exponential, gamma, poisson, and truncated versions of these) for model creation only (no simulation). LibRoadRunner (`http://libroadrunner.org`) also supports the normal and uniform functions (though not their truncated forms), and is a full simulator. Neither Antimony nor LibRoadRunner support the `uncertainty` child of ***SBase***, and support the extended **FunctionDefinition** only.

7 Acknowledgements

Much of the initial concrete work leading to this proposal document was carried out at the Statistical Models Workshop in Hinxton in 2011, which was organised by Nicolas le Novère. A list of participants and recordings of the discussion is available from http://sbml.org/Events/Other_Events/statistical_models_workshop_2011. Before that a lot of the ground work was carried out by Darren Wilkinson who led the discussion on *distrib* at the Seattle SBML Hackathon and before that Colin Gillespie who wrote an initial proposal back in 2005. The author would also like to thank the participants of the *distrib* sessions during HARMONY 2012 and COMBINE 2012 for their excellent contributions in helping revising this proposal; Sarah Keating, Maciej Swat and Nicolas le Novère for useful discussions, corrections and review comments; and Mike Hucka for LaTeX advice and the beautiful template upon which this document is based.

A Changes anticipated in UncertML 3.0

Now that the UncertML 3.0 xsd is out, there are probably not going to be many more changes, if any.

B UncertML Distributions

In this table, all UncertML 3.0 distributions are listed, along with their types (Continuous, Categorical, or Discrete), whether they're univariate or multivariate, and a brief description. The UncertML element name is the name of the distribution with spaces removed and "`Distribution`" appended, so, for example, the "`Exponential`" distribution becomes "<`ExponentialDistribution`>", and the "`Student T`" distribution becomes "<`StudentTDistribution`>". The exception is that all of the mixture models (ones that end with "`Mixture Model`"), only have spaces removed but nothing appended: the Continuous Univariate Mixture Model becomes "<`ContinuousUnivariateMixtureModel`>", etc.

All of these distributions inherit from the abstract UncertML *AbstractDistribution* class. Additionally, the appropriate distributions inherit from the *AbstractContinuousDistribution*, *AbstractCategoricalDistribution*, and *AbstractDiscreteDistribution* abstract classes, and further from the *AbstractContinuousUnivariateDistribution*, *AbstractContinuousMultivariateDistribution*, etc. classes, which are related to one another as one would expect.

All descriptions are copied from `http://www.uncertml.org/`, where more information can be found. Do note that any XML content there based on UncertML 2.0 will not be applicable to the Distributions package, which uses UncertML 3.0 instead. The arguments and definitions will, however, still be relevant.

Distributions are listed grouped by category (type and univarite/multivariate), and alphabetical within those categories.

Distribution	Type	Variables	Description
Beta	Continuous	Univariate	A random variable x is Beta distributed if the probability density function (pdf) is of the form $\frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}$, where $B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$. The distribution is usually denoted as $x \sim Be(\alpha,\beta)$ with parameters α and β, both positive real values. As the domain of the random variable is defined to be $[0,1]$ the Beta distribution is normally used to describe the distribution of a probability value.
Cauchy	Continuous	Univariate	A random variable x follows a Cauchy distribution if the probability density function (pdf) is of the form $\frac{1}{\pi\gamma}\left[1+\left(\frac{x-\theta}{\gamma}\right)^2\right]^{-1}$. The Cauchy distribution is equivalent to a Student-T distribution with 1 degree of freedom. It is widely used in physics, optics and astronomy. It is also known as the Lorenz or the Breit-Wigner distribution.
Chi Square	Continuous	Univariate	A random variable x is Chi-square distributed if the probability density function (pdf) is of the form $\frac{1}{\Gamma(\nu/2)2^{\nu/2}} x^{\nu/2-1} exp(-x/2)$. The distribution is usually denoted as $x \sim \chi_\nu$ where ν is known as the degrees of freedom parameter. ν has to be positive and x has to be non-negative for the density to be defined. The Chi-square distribution is a special case of the Gamma distribution where $\chi \sim \Gamma(k=\nu/2, \theta=2)$.
Continuous Univariate Mixture Model	Continuous	Univariate	A mixture model is a linear combination of base distributions. A widely used case is where the base distributions are Gaussian in which case the model is known as the Gaussian Mixture Model.
Dirac Delta	Continuous	Univariate	[New to UncertML 3.0–no definition yet.]

Exponential	Continuous	Univariate	A random variable x follows an exponential distribution if the probability density function (pdf) is of the form $\lambda e^{-\lambda x}$. It is often represented as $x \sim \text{Exp}(\lambda)$. It is used to model the time between events for a Poisson process and is used in simulation of stochastic systems.
F	Continuous	Univariate	A random variable x follows an F distribution if the probability density function (pdf) is of the form $\frac{1}{B(\nu_1/2,\nu_2/2)}\left(\frac{\nu_1}{\nu_2}\right)^{\nu_1/2} x^{\nu_1/2-1}\left(1+\frac{\nu_1}{\nu_2}x\right)^{-\frac{\nu_1+\nu_2}{2}}$ where $B(.)$ is the Beta function. It often arises as the ratio of two random variables that are identically Chi-Square distributed.
Gamma	Continuous	Univariate	A random variable x is Gamma distributed if the probability density function (pdf) is of the form $\frac{1}{\gamma(k)\theta^k}x^{k-1}$. The distribution is usually denoted as $x \sim \mathcal{N}(\mu,\sigma^2)$ where k is known as the shape parameter and θ the scale parameter. Both parameters have be positive and x has to be non-negative for the density to be defined. In practice the Gamma distribution is often use to model the distribution of non-negative quantities such as variances.
Inverse Gamma	Continuous	Univariate	A random variable x is Inverse Gamma distributed if the probability density function (pdf) is of the form $\frac{\beta^\alpha}{\gamma(\alpha)}x^{-\alpha-1}exp(-\beta/x)$. If variable x is Inverse Gamma distributed, $1/x$ is gamma distributed. The Inverse Gamma distribution function can be obtained from the Gamma distribution by a transformation of variables.
Laplace	Continuous	Univariate	A random variable x is Laplace distributed if the probability density function (pdf) is of the form $\frac{1}{2b}exp(-\frac{abs(x-\mu)}{b})$ where abs denotes the absolute value. It can be thought of as a combination of two exponential distributions.
Log Normal	Continuous	Univariate	A random variable x is Log Normal distributed if the probability density function (pdf) is of the form $\frac{1}{x\sqrt{2\pi\sigma^2}}exp(-\frac{(ln(x)-\mu)^2}{2\mu^2})$. If variable x is normally distributed, exp(x) is Log Normal distributed. The Log Normal distribution function can be obtained from the normal distribution by a transformation of variables. It is often used for variables that must be positive.
Logistic	Continuous	Univariate	A random variable x is Logistic distributed if the probability density function (pdf) is of the form $\frac{exp(-(x-\mu)/s)}{s(1+exp(-(x-\mu)/s))^2}$.
Normal	Continuous	Univariate	A random variable x is normally distributed if the probability density function (pdf) is of the form $\frac{1}{\sqrt{2\pi\sigma^2}}exp(-\frac{(x-\mu)^2}{2\sigma^2})$. The distribution is usually denoted as $x \sim \mathcal{N}(\mu,\sigma^2)$ where μ is known as the mean parameter and σ^2 the variance parameter. If the random variable x is a vector of length greater than one, the normal distribution can be generalised to the Multivariate normal. A reason for the widespread usage of the normal distribution is the Central limit theorem which states that the distribution of the mean of a large number of independent identically distributed random variables tends to a normal distributions as the number of random variables increases.

Pareto	Continuous	Univariate	A random variable x follows a Pareto distribution if the probability density function is of the form $\frac{\alpha x_m^\alpha}{x^{\alpha+1}}$. The distribution allows for the specification of a minimum value below which the density is 0. It is a skewed heavy-tailed distribution.
Student T	Continuous	Univariate	A random variable x follows a Student-t distribution if the probability density function (pdf) is of the form $\frac{\Gamma(\nu/2+1/2)}{\Gamma(\nu/2)(\pi\nu\sigma^2)^{1/2}}[1+\frac{(x-\mu)^2}{\nu\sigma^2}]^{-\nu/2-1/2}$. The distribution is usually denoted as $x \sim St(\mu,\lambda,\nu)$. This distribution corresponds to integrating out the variance of a normal distribution using a inverse Gamma prior. It can therefore be interpreted as an infinite mixture of normal distributions having the same mean but different variances. The three parameters are the mean (μ), degrees of freedom (ν) and variance (σ^2). Setting the variance to 1 and the mean to 0 we obtain the Student-t form found in standard statistics references such as Wikipedia. Setting the d.f. to 1 the Cauchy distribution is obtained. Setting the d.f. to infinity the normal distribution is obtained. The student-t distribution is commonly used in likelihood inference as the maximum likelihood parameter estimates are more robust to outlier observations compared to the normal distribution.
Uniform	Continuous	Univariate	A random variable x follows a uniform distribution if the probability density function (pdf) is of the form $\frac{1}{b-a}$. The distribution assigns equal probability to all events within the chosen domain between the minimum (a) and the maximum (b).
Weibull	Continuous	Univariate	A random variable x follows an Weibull distribution if the probability density function (pdf) is of the form $\frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} exp(-x/\lambda)^k$. It includes the exponential distribution as a special case. It is often used in engineering and finance.
Continuous Multivariate Mixture Model	Continuous	Multivariate	A mixture model is a linear combination of base distributions. A widely used case is where the base distributions are Gaussian in which case the model is known as the Gaussian Mixture Model.
Dirichlet	Continuous	Multivariate	A K dimensional random variable x follows a Dirichlet distribution if the probability density function (pdf) is of the form $\frac{1}{B(\mathbf{a})}\prod_{i=1}^K x_i^{\alpha_i-1}$ where $B(\mathbf{a}) = \frac{\prod_{i=1}^k \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)}$ and $\Gamma(.)$ is the Gamma function. It is the multivariate extension of the beta distribution to higher dimensions with K a positive integer greater than or equal to 2.
Multivariate Normal	Continuous	Multivariate	The Multivariate Normal is an extension of the univariate normal distribution to higher dimensional vector spaces. A random vector variable of dimension k denoted $\mathbf{x}$ is normally distributed if the probability density function (pdf) is of the form $(2\pi)^{-k/2}\det(\Sigma)^{-1/2} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu))$ where $\det(.)$ denotes the determinant and $(.)^T$ the matrix transpose. The distribution is usually denoted as $\mathbf{x} \sim \mathcal{N}(\mu,\Sigma)$ where μ is known as the mean vector parameter and Σ the covariance matrix parameter.

Multivariate Student T	Continuous	Multivariate	A random variable $\mathbf{x}$ follows a multivariate Student-t distribution if the probability density function (pdf) is of the form $\frac{\Gamma(\nu/2+k/2)}{\Gamma(\nu/2)(\pi\nu)^{k/2}\det(\Sigma)^{1/2}}\left[1+\frac{\Delta^2}{\nu}\right]^{-\nu/2-k/2}$ where $\Delta^2=(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)$ is the squared Mahalanobis distance. The distribution is usually denoted as $x\sim St(\mu,\Sigma,\nu)$. It is the extension of the univariate student-t distribution to higher dimensions. Student-t distributions are often used when tails are expected to be heavier than Gaussian or Normal, and can result from applying Bayesian inference.
Normal Inverse Gamma	Continuous	Multivariate	A Normal Inverse Gamma distribution is the conjugate prior of a normal distribution with unknown mean and variance. It is the coupled product of an Inverse Gamma distribution and a normal distribution. In particular if $p(\mathbf{X};\mu,\sigma^2)$ is the likelihood function of a Normally distributed set of random variables with mean μ and variance σ^2 and if both the mean and variance are considered unknown, the conjugate prior is $p(\mu,\sigma^2)=p(\mu;\sigma^2)p(\sigma^2)$ where $p(\mu;\sigma^2)=\mathcal{N}(\mu;\mu_0,\sigma^2/\nu)$ a Normal prior on the mean and $p(\sigma^2)=\mathrm{IG}(\sigma^2;\alpha,\beta)$ an Inverse Gamma prior on the variance. Note that the priors are not independent as the prior variance of the mean is a linear function of of the variance σ^2. It is also common to use a Normal-Gamma distribution where a conjugate prior is placed on the unknown mean and precision (i.e. inverse variance) of the Normal likelihood in which case the prior is a product of a Normal and Gamma distributions. In the case of a Multivariate normal likelihood, the corresponding conjugate prior is a Normal-Wishart distribution.
Bernoulli	Categorical	Univariate	A random variable x follows a Bernoulli distribution if the probability mass function (pmf) is of the form $\mu^x(1-\mu)^{1-x}$. It describes the distribution of a single binary variable x.
Categorial Univariate Mixture Model	Categorical	Univariate	A mixture model is a linear combination of base distributions. A widely used case is where the base distributions are Gaussian in which case the model is known as the Gaussian Mixture Model.
Categorical	Categorical	Multivariate	A Categorical distribution is a generalisation of the Bernoulli distribution to K discrete outcomes, giving the K probabilities p_i, $i=1,..,K$ for each outcome. There is no ordering in the K outcomes.
Categorial Multivariate Mixture Model	Categorical	Multivariate	A mixture model is a linear combination of base distributions. A widely used case is where the base distributions are Gaussian in which case the model is known as the Gaussian Mixture Model.
Binomial	Discrete	Univariate	A random variable x follows a Binomial distribution if the probability mass function (pmf) is of the form $\binom{n}{x}\theta^x(1-\theta)^{n-x}$, where $\binom{n}{x}$ denotes n choose x. The distribution is usually denoted as $x\sim b(n,\theta)$. The distribution describes the probability of getting x successes in n trials of independent experiments that have the same probability of success.

Discrete Univariate Mixture Model	Discrete	Univariate	A mixture model is a linear combination of base distributions. A widely used case is where the base distributions are Gaussian in which case the model is known as the Gaussian Mixture Model.
Geometric	Discrete	Univariate	A random variable x follows a geometric distribution if the probability mass function (pmf) is of the form $(1-p)^{x-1}p$. It is often represented as $x \sim \text{Geom}(p)$. It is the discrete analogue of the exponential distribution. It is used to model distribution of the number of binary (Bernoulli) trials needed to get one success, with parameter, probability p.
Hypergeometric	Discrete	Univariate	A random variable x follows a hypergeometric distribution if the probability mass function (pmf) is of the form $\frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}}$, probability of getting x successes. It describes the number of successes in a sequence of draws without replacement.
Negative Binomial	Discrete	Univariate	A random variable x follows a Negative Binomial distribution if the probability mass function (pmf) is of the form $\binom{x+r-1}{x}p^x(1-p)^r$. The distribution describes the probability of getting x successes in trials of independent experiments that have the same probability of success, and are run until we observe r failures.
Poisson	Discrete	Univariate	A random variable x follows a Poisson distribution if the probability mass function (pmf) is of the form $\frac{\lambda^x}{x!}\exp(-\lambda)$. The Poisson distribution can be used to model the number of events occurring within fixed time period of time.
Discrete Multivariate Mixture Model	Discrete	Multivariate	A mixture model is a linear combination of base distributions. A widely used case is where the base distributions are Gaussian in which case the model is known as the Gaussian Mixture Model.
Multinomial	Discrete	Multivariate	A random variable x follows a Multinomial distribution if the probability mass function (pmf) is of the form $\frac{N!}{x_1!\ldots x_k!}\prod_{i=1}^{K} p_i^{x_i}$. The Multinomial distribution is a multivariate generalisation of the Binomial distribution for a K state variable to be in state k given N observations. This can be confused with the Categorical distribution (added at version 3.0 of UncertML) which can be considered a Multinomial distribution when a 1 of K encoding is used.
Wishart	Discrete	Multivariate	A random matrix variable $\mathbf{X}$ of size $D \times D$ follows a Wishart distribution if the probability density function is of the form $\det(\mathbf{W})^{-\nu/2}\left(2^{\nu D/2}\pi^{D(D-1)/4}\prod_{i=1}^{D}\Gamma\left(\frac{\nu+1-i}{2}\right)\right)^{-1}$ $\det(\mathbf{X})^{(\nu-D-1)/2}\exp\left(-\frac{1}{2}\text{Tr}(\mathbf{W}^{-1}\mathbf{X})\right)$ where det denotes the determinant, Tr the matrix trace. The Wishart distributon is the conjugate prior for the inverse of a covariance matrix of a Multivariate Normal distribution. It is a generalistion of the gamma distribution to higher dimensions. In one dimesion the Wishart distribution is equivalent to a gamma distribution with parameters $k = \nu/2$ and $\theta = 1/2\mathbf{W}$.

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/.

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/.

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

SBML Editorial Board (2012). SBML development process for SBML level 3. Available via the World Wide Web at http://sbml.org/Documents/SBML_Development_Process/SBML_Development_Process_for_SBML_Level_3.

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/.