Food Safety Knowledge Markup Language (FSK-ML)

Software Developer Guide

Version 1.0

Matthias Filter (Chair)	Federal Institute for Risk Assessment, Germany
Guido Correia Carreia	Federal Institute for Risk Assessment, Germany
Alexander Falenski	Federal Institute for Risk Assessment, Germany
Miguel de Alba Aparicio	Federal Institute for Risk Assessment, Germany

Contact:

Matthias Filter (matthias.filter@bfr.bund.de)

July 2016

List of tables

1. Introduction

Food safety risk assessments, control of food production processes as well as the development of new food products are nowadays supported by application of mathematical modeling and data analysis techniques. This creates an increasing demand for resources facilitating the efficient, transparent and quality proven exchange of relevant information, e.g. analytical data, mathematical models, simulation setting as well as simulated data. For example, new parameterized microbial models are frequently made publicly available only in written mode via scientific publications. However, in order to apply these models to a given practical decision support question (e.g. on the growth/no-growth of a microorganism in a specific food matrix under given processing conditions) the interested end-user would have to re-implement the model based on information provided in a publication. Here it would be more efficient if those who create parameterized models could provide their model additionally as a file complying with a standardized file format that is also capable of transferring all relevant meta-information. Such a file could e.g. be provided as a supplement to the publication and could be read-in by the end user's software tools (thus overcoming an error-prone re-implementation process).

The required standardized file format has been proposed in the "Predictive Modelling in Food Markup Language (PMF-ML) Software Developer Guide". This document describes in detail how experimental data and mathematical models from the domain of predictive microbial modeling (and beyond) can be saved and encoded in a **software independent manner**. Here we further extend the PMF-ML format in order to enable in addition the exchange of knowledge / information that is **embedded in specific script-based programming languages** (e.g. "R", Matlab, Python). I.e. the FSK-ML guidance document primarily aims at harmonizing the exchange of food safety knowledge (e.g. predictive models) including the associated meta-information where this knowledge is only available in a **software-dependent** format.

The FSK-ML format therefore relaxes and adapts certain specifications of the PMF-ML format while at the same time maintaining the highest possible synergies between both formats. This will also help to make sure that food safety models encoded in a software-

independent manner (using PMF-ML) can easily be interpreted by FSK-ML import and export software functions in the future.

1.1. Objectives

This guidance document is primarily designed for software developers and project managers that want to enhance their software tools with import and export functions for food safety models, simulations or food safety data or for those who want to develop new tools.

The document describes the structure and requirements of FSK-ML including the structure and requirements of a newly defined file format which is based on the Open Modeling EXchange guidelines (OMEX). The document also describes information needed to make encoded script-based models executable. Future versions of the document will also describe how to encode simulation settings and settings for data visualization.

1.2. Document conventions

This document uses the conventions defined in the SED-ML specification document (Bergmann, Cooper, Le Novère, Nickerson, & Waltermath, 2015).

UML 1.0 (Unified Modelling Language), (OMG, 2009), notation is used in this document to define the constructs provided by this package. The following colours are used to provide more meaningful diagrams.

- **Black**. Items coloured black in the UML diagrams are components taken unchanged from their definition in the SED-ML Level 1 Version 2 specification document.
- **Green**. Items coloured green are components that exist in SED-ML Level 1 Version 2, but are extended by this package.
- **Blue**. Items coloured blue are new components introduced in this package specification. They have no equivalent in the SED-ML Level 1 Version 2 specification.

The following typographical conventions distinguish the names of objects and data types from other entities.

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 SED-ML Level 1 Version 2, 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 SED-ML Level 1 Version 2 specification.)

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

2. Related standards

2.1. SBML

See “Predictive Modelling in Food Markup Language (PMF-ML) Software Developer Guide”.

2.2. SED-ML

See “Predictive Modelling in Food Markup Language (PMF-ML) Software Developer Guide”.

2.3. PMF-ML

See “Predictive Modelling in Food Markup Language (PMF-ML) Software Developer Guide”.

2.4. OMEX

Open Modelling EXchange format, OMEX (Bergmann, et al., 2014), aims to support the exchange of information necessary for modelling and simulating experiments in biology. OMEX defines a COMBINE Archive or OMEX files as a ZIP container containing at least a file with a listing of content in the archive (manifest file), optional metadata and the files describing e.g. food safety models. The use of COMBINE standard file formats for the encoding of models is encouraged, although several Internet media types are also supported.

3. FSK-ML Specifications

3.1. FSK Terminology

3.1.1. Model

A model is a mathematical description of a system. Models can be parameterized, meaning that certain model parameters carry pre-defined fixed values (which might have been obtained in a parameter fitting process). The main purpose and mode of application of (parameterized) models is generating predictions for certain variable settings where no real world or experimental data are available.

In parameterised models some independent variables have defined values while others do not. The variables with predefined values are called parameters. For instance, a model may be parameterized based on experimental data on the growth of a certain pathogen in a given temperature range. In this example the fitted growth rate parameter is a model parameter and is specific to the studied pathogen. The parameterized model should therefore only used for generating predictions on the pathogen studied.

3.1.2. Simulation

Model-based predictions generated from different parameterizations of the same model are called simulations. I.e. generating model-based predictions from a set of definite values for all model parameters will be called a simulation. However different sets of definite values for all model parameters might originate from different simulation scenarios.

3.2. The FSKX format

In order to support the exchange of sets of related files a FSKX container file can be created. A FSKX container file is a ZIP file containing an arbitrary (at least one) number of files. We recommend the file extension ".fskx". The FSKX container has to comply with the Open Modeling EXchange format (OMEX) specifications set out in (Bergmann, Adams et al. 2014). It allows to exchange the following files:

- a manifest file ("manifest.xml") - mandatory; providing a listing of all files inside the FSKX container
- a metadata file ("metadata.rdf") providing additional information on the archive and its content
- files related to the model(s):
    - model metadata (e.g. "model_1.fsk") providing all relevant metadata on referenced model script(s) / executable software (black-box) model(s)
    - model script(s) (e.g. "model_1.R") / executable software (black-box) model(s) (e.g. "model_1.exe"/ pmf-ml files (e.g. "model_1.pmf")
    - Third party (including binary) libraries required to run the model script(s) / executable software (black-box) model(s)
    - Data used for model generation / validation (e.g. "experiment_1.numl")
- file(s) related to simulation(s) / prediction and simulation / prediction result(s):
    - Simulation settings (e.g. "Simulation_1.sed")
    - Simulation results (e.g. "Simulation_1_res.numl")
    - Visualization script (e.g. "Simulation_1_plot1.R")
- Other file(s)
    - Model reference description (e.g. "Paper_model_1.pdf")

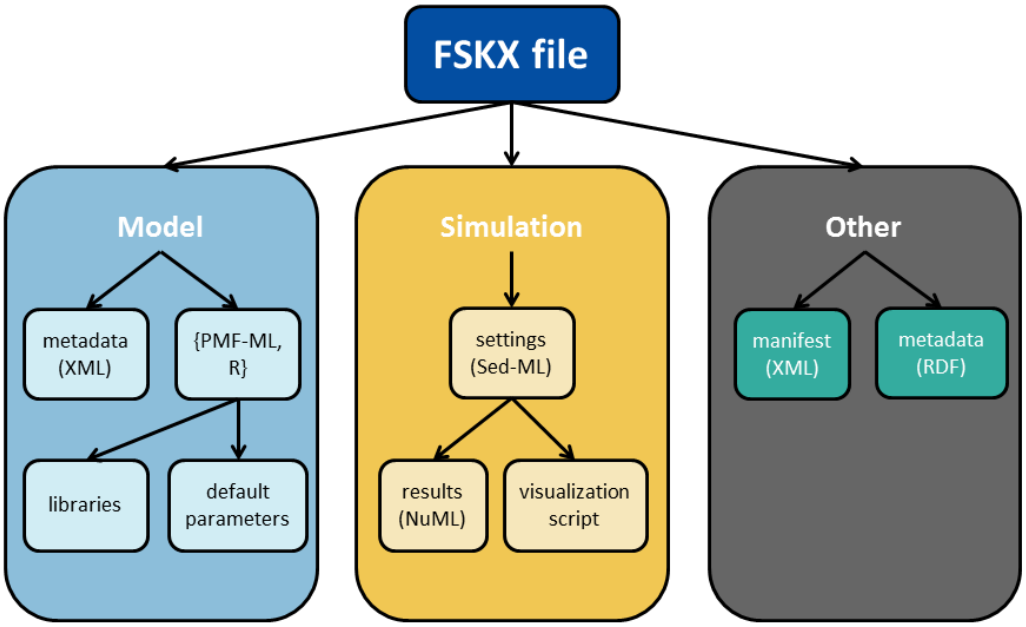

Figure 1 Structure of the FSKX file

3.3. Supported data types

- Numeric: Real numbers.
- Integer
- Character: String values
- Vector: one-dimensional array
- Matrix: two-dimensional array

Kommentar [MA1]: Need to elaborate more on the data types. Range, possible values, etc.

Code vocabulary for the supported data types:

Numeric	`numeric`
Integer	`integer`
Character	`character`
Vector	`vector`
Matrix	`matrix`

Table 1 Supported data types

The supported data types are case-insensitive.

```
# Examples of scalar data types
pi = 3.14
year = 2016
name = "Miguel, Lord of Westeros"

# Vectors
cookies_per_meeting <- c(10, 5, 20, 15, 10)
cookies_per_meeting <- c(10, "fünf", 20, "fünfzehn", 10)
cookies_per_meeting <- c("zehn", "fünf", "zwanzig", "fünfzehn",
       "zehn")

# Matrices
menus = matrix(
       c("Hänchen", "Reis", "Fishfilet", "Geschnezeltes", "Spinach",
              "Boulette"),
   nrow = 2, ncol = 3, byrow = TRUE,
   dimnames = list(
       c("Menu 1", "Menu 2"), c("Montag", "Dienstag", "Mittwoch")))
```

Figure 2 Examples variables

3.4. Files related to models

3.4.1. Model script

The model script is a script stored within the FSKX archive that calculates the values of the model.

```
########################################################################
test_model_1 = function(a, b, c){
  res = 100*(rbeta(a, shape1 = b + 1, shape2 = c - b + 1))
  return(res)
}
```

Figure 3 Example model script

3.4.2. (Binary) Model libraries

For an autonomous run the model need to be provided such that required libraries are provided together with the model. In the case of an R model the corresponding libraries would be binary libraries for the system the model was created on.

While the placement of the libraries in the archive is free, they can be stored anywhere in the FSKX file; it is advisable to save them into a separated “lib” folder within the archive.

3.4.3. Model metadata

The model metadata are stored in a PMF file (.pmf) adopting the PMF-ML specifications on encoding of model metadata.

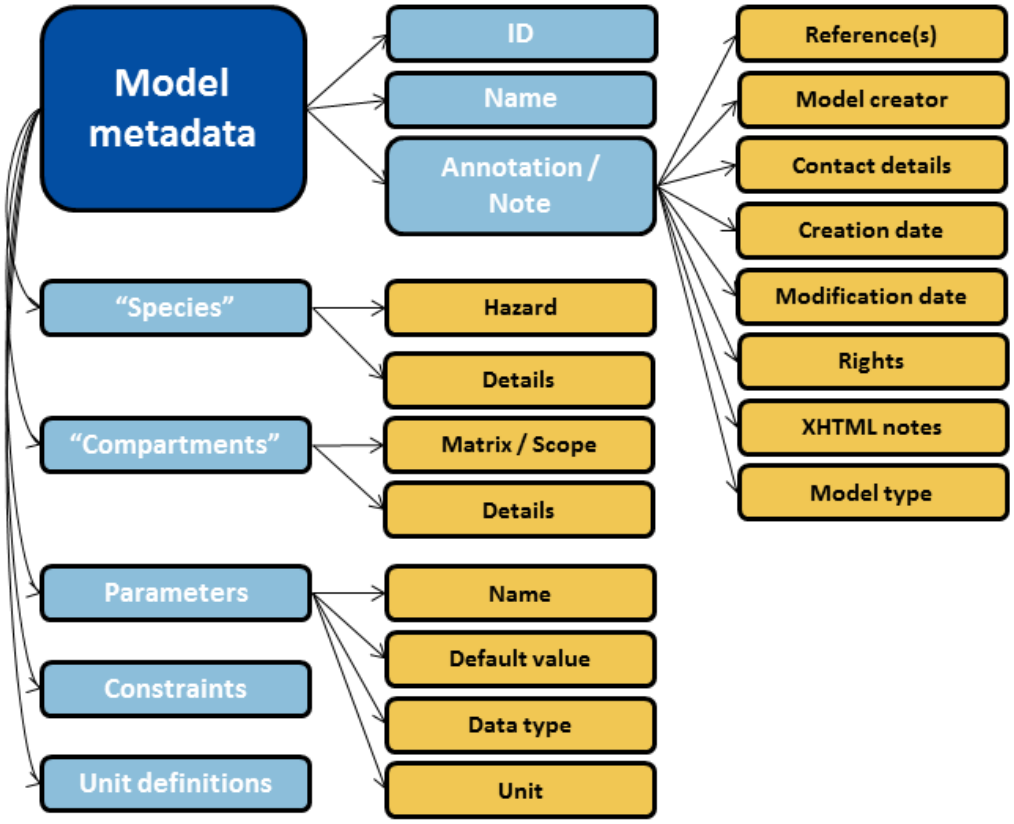

Figure 4 Model metadata encoded in a standardized way within the FSK-ML file

Model	Creator	The person contributed to the encoding of the model in its present form by implementing the model in the software solution.
	Family	Model family.
	Contact	Contact information.
	Catalogue model	Reference to catalogue model.
	Creation date	Date upon which the model was created.
	Modification date	Date of last modification to the model.
	Rights	Information about rights held in and over the resource. Typically, rights information includes a statement about various property rights associated with the resource, including intellectual property rights.
	Notes	Model notes

	Id	Model id
	Name	Model name
Species	Hazard	Hazard name
	Detail	Hazard detail
Compartment	Matrix/Scope	Compartment name
	Detail	Compartment detail
Parameters	Name	Parameter name
	Default value	Default value of the parameter
	Constant	Boolean stating where the parameter value is constant
	Data type	
UnitDefinition	Units	List of SBML units.
	Transformation	Transformation name.
Constraint	Min & Max	Minimum and maximum value of a non-constant parameter.

Table 2 Model metadata

3.4.4. FSK Types

FSK types characterize the particular model using up to two attributes. The first attribute subsumes the model into the OIE and CODEX Alimentarius scheme (Codex Attribute). The second attribute would come from a controlled vocabulary and for the Codex Attribute this is certainly feasible. But at the current state of development (i.e. without larger input from the QMRA community) it would be very limiting to impose mandatory Model Type attributes. Therefore Codex attributes are controlled while for model type attributes we provide suggestions but allow also free text inputs. The Codex attributes are case insensitive.

Codex attribute	Model type attribute
Entry assessment	Process model, epidemiological model, supply chain model
Exposure assessment	Process model, consumption model, population model, household generation model
Hazard characterization	Dose response model, epidemiological model (for secondary effects)
Risk characterization	Sensitivity analysis, DALY model, cost of illness model

Table 3 FSK types

3.5. Simulation files

FSK-ML provides an opportunity to define, store and exchange simulation scenarios. All relevant settings (including the reference to the used model files) are stored in XML files using the FSK-SED-ML format (see section 4).

The results of simulations can be stored either as software independent NuML XML file or as a scripting language specific binary file where the results from executing a defined simulation are

stored. In case of the R scripting language a valid simulation result file would be a "R workspace" generated after the simulation of one R model.

3.5.1. FSK-SED-ML files

Kommentar [miguel2]: TODO:
- Purpose of FSK-SED-ML files
- Contents: parameters, outputs, etc.

The FSK-SED-ML files are XML files that describe the simulation of a model:

- The parameter values for the simulation
- The type of simulation: deterministic, statistic or probabilistic
- The output with the results of the simulation

FSK-SED-ML is described in more detail in 4 FSK-SED-ML.

3.5.2. Visualization scripts

Visualization scripts can be included and might contain a number of commands in a scripting language corresponding to the one used for the model. It can be used to create plots or charts using from the results generated in a simulation. Visualization scripts might also be referenced from within FSK-SED-ML files.

```
hist(test_model_1(200, 20, 100), breaks=50, main="Headline", xlab="Text", col="32")
```

Figure 5 Example visualization script

3.5.3. Results (NuML)

The results of the model are encoded with NuML. Each of the supported data types are encoded in a different fashion.

General guidelines

The variables of the results are kept within a NuML **CompositeDescription** in the **Dimension Description** of a **ResultComponent**. This **CompositeDescription** keeps the names and types of the variables. The values of the variables are saved into a single **Dimension** corresponding to the **Composite Description**.

```
<numl>
...
  <resultComponent id=”rc1”>
    <dimensionDescription>
      <compositeDescription>
      Variables ...
    </dimensionDescription>
    <dimension>
      Values ...
    </dimension>
  </resultComponent>
</numl>
```

3.5.3.1. Integer variable

An integer variable is described with a single **AtomicDescription** of type `integer`. Its value is described with an **AtomicValue**.

```
<atomicDescription name=”integer_variable” valueType=”integer” />
```
Table 4 NuML Integer variable description

```
<atomicValue>10</atomicValue>
```
Table 5 NuML Integer variable value

3.5.3.2. Numeric variable

An integer variable is described with a single **AtomicDescription** of type `double`. Its value is described with an **AtomicValue**.

```
<atomicDescription name=”numeric_variable” valueType=”double” />
```
Table 5 NuML Integer variable description

```
<atomicValue>3.141592653589793</atomicValue>
```
Table 5 NuML Integer variable value

3.5.3.3. Character variable

An integer variable is described with a single **AtomicDescription** of type `character`. Its value is described with an **AtomicValue**.

```
<atomicDescription name=”character_variable” valueType=”string” />
```
Table 6 NuML Integer variable description

```
<atomicValue>3.141592653589793</atomicValue>
```
Table 5 NuML Integer variable value

3.5.3.4. Vector variable

A vector variable is described with a **CompositeDescription** with the name of the variable and with `indexType` integer. This **CompositeDescription** involves an **AtomicDescription** with a `valueType`. The types may be `integer` for integer vectors, `double` for numeric vectors and `string` for character vectors.

```
<compositeDescription name=”vector_variable” indexType=”integer”>
  <atomicDescription name=”vector_item” ontologyTerm=”dummy”
    valueType=”integer”|”double”|”string” />
</compositeDescription>
```

Table 7 NuML vector variable description

Its values are described by a **CompositeValue** coupled with a single **AtomicValue** of the same type as in its description. The values are contained in a **Dimension** corresponding to the **CompositeDescription** of the vector.

```
<dimension>
  <compositeValue indexValue=”0”><atomicValue>0</atomicValue></compositeValue>
  <compositeValue indexValue=”1”><atomicValue>1</atomicValue></compositeValue>
  <compositeValue indexValue=”2”><atomicValue>1</atomicValue></compositeValue>
  <compositeValue indexValue=”3”><atomicValue>2</atomicValue></compositeValue>
  <compositeValue indexValue=”4”><atomicValue>3</atomicValue></compositeValue>
  ...
</dimension>
```

Table 8 NuML vector variable values

3.5.3.5. Matrix variable

A matrix variable is fairly similar to a vector variable but with a second column. It is described with a **CompositeDescription** providing the name of the variable and `indexType` integer. The **CompositeDescription** involves two **AtomicDescription** with the same `valueType`. The types may be `integer` for integer vectors, `double` for numeric vectors and `string` for character vectors.

```
<compositeDescription name=”matrix_variable” indexType=”integer”>
  <atomicDescription name=”col1” ontologyTerm=”dummy” valueType=”integer|double|string” />
  <atomicDescription name=”col2” ontologyTerm=”dummy” valueType=”integer|double|string” />
</compositeDescription>
```

Table 9 NuML matrix variable description

Its values are described by a **CompositeValue** consisting of two **AtomicValue** of the same type as in its description. The values are contained in a **Dimension** corresponding to the **CompositeDescription** of the matrix.

```
<dimension>
  <compositeValue indexValue=”0”>
    <atomicValue>0</atomicValue><atomicValue>0</atomicValue>
    <atomicValue>1</atomicValue><atomicValue>1</atomicValue>
    <atomicValue>2</atomicValue><atomicValue>4</atomicValue>
    <atomicValue>3</atomicValue><atomicValue>9</atomicValue>
    <atomicValue>4</atomicValue><atomicValue>16</atomicValue>
  </compositeValue>
</dimension>
```

Table 10 NuML matrix variable values

3.6. Other

3.6.1. manifest.xml

The FSK container is a COMBINE archive and the presence of this file is a requirement of the COMBINE archive.

One file must be present at the root of any COMBINE archive, named manifest.xml. This file contains an instantiation of the OmexManifest class. It contains a number of Content children, one of which represents the COMBINE archive itself. A valid manifest needs to have at least one entry, declaring the archive itself, but may contain as many entries as need. All the files present in the archive must be listed in the manifest. The only optional entry describes the manifest.xml file. Indeed the presence of manifest.xml is mandatory and its declaration is not necessary for parsing.

```
<?xml version="1.0" encoding="UTF-8"?>
<omexManifest
       xmlns="http://identifiers.org/combine.specifications/omex-manifest">
  <content location="."
       format="http://identifiers.org/combine.specifications/omex" />
  <content location="./manifest.xml"
       format="http://identifiers.org/combine.specifications/omex-manifest" />
  <content location="./visualization.r"
       format="http://purl.org/net/mediatypes/application/text/x-r" />
  <content location="./metaData.pmf"
       format="http://purl.org/net/mediatypes/application/application/sbml+xml" />
  <content location="./workspace.r"
       format="http://purl.org/net/mediatypes/application/text/x-r" />
  <content location="./triangle_0.10.zip"
       format="http://purl.org/net/mediatypes/application/application/zip" />
  <content location="./model.r"
       format="http://purl.org/net/mediatypes/application/text/x-r" />
  <content location="./param.r"
       format="http://purl.org/net/mediatypes/application/text/x-r" />
  <content location=".\metadata.rdf"
       format="http://identifiers.org/combine.specifications/omex-metadata" />
</omexManifest>
```

Table 11 Example manifest

The `Content` class represents a file in the COMBINE archive. The format attribute in a content element describes the file type. It may take either a URI corresponding to Identifiers.org for COMBINE standards such as SBML or SedML or an Internet media type (Freed, 1996), previously known as "MIME type". FSK makes use of extra file types through the following Internet media types.

File type	Internet media type
Zip (.zip)	http://purl.org/NET/mediatypes/application/zip
R (.r)	http://purl.org/NET/mediatypes/text/x-r
PMF (.pmf)	http://purl.org/NET/mediatypes/application/x-pmf
SBML (.sbml)	http://purl.org/NET/mediatypes/application/sbml+xml
Matlab (.m)	http://purl.org/NET/mediatypes/text/x-matlab
PHP (.php)	http://purl.org/NET/mediatypes/text/x-php

Table 12 Formats used in the manifest

3.6.2. The Archive metadata

In the FSKX archive the use of Internet media types as formats allows the detection of different types of files within the archive file such as scripts and binary libraries. However, files might share the same media types, as e.g. for R scripts and R workspace files. Therefore the use of Internet media types is not enough.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:dcterms="http://purl.org/dc/terms/"
       xmlns:vCard="http://www.w3.org/2006/vcard/ns#">

  <rdf:Description rdf:about=".">
       <dc:type>mainScript</dc:type>
       <dc:source>model.r</dc:source>
  </rdf:Description>

  <rdf:Description rdf:about=".">
       <dc:type>paramScript</dc:type>
       <dc:source>param.r</dc:source>
  </rdf:Description>

  <rdf:Description rdf:about=".">
       <dc:type>visualizationScript</dc:type>
       <dc:source>visualization.r</dc:source>
  </rdf:Description>

  <rdf:Description rdf:about=".">
       <dc:type>workspace</dc:type>
       <dc:source>workspace.r</dc:source>
  </rdf:Description>
</rdf:RDF>
```

Figure 6 Example archive metadata

The Archive metadata in FSK addresses this issue, providing a lightweight annotation using the Resource Description Format, RDF (Cyganiak, Wood, Lanthaler, Klyne, Carroll, & and McBride, 2014). This annotation involves RDF description elements that describe the type and location of the resource with DC type and source elements respectively. The accepted types are:

- `mainScript`: filename of the model script
- `paramScript`: filename of the parameters script
- `visualizationScript`: filename of the visualization script
- `workspace`: filename of the workspace with the results of the simulation

4. FSK-SED-ML

Kommentar [miguel3]: Introduced in 3.5.1

FSK-SED-ML extends SED-ML to describe the simulation of FSK-ML models. FSK-SED-ML manages this introducing new **Simulation** and **Output** classes compatible with the FSK-ML models.

4.1. Model

Like in the classical SED-ML, in the "Model" class of FSK-SED-ML changes to the default model values are notified which will characterize the simulation. However, only variables can be changed here while the parameters cannot be changed.

Kommentar [MF4]: ???
Maybe you should provide an introduction first before 4.1 There you should point out that you have to modify SED-ML in order to describe simulations with FSK-ML models

4.2. FskSimulation

In SED-ML the simulation classes act as containers for the simulation experiments. FSK-SED-ML does not use the original simulation classes in SED-ML (**UniformTimeCourse**, **OneStep** and **SteadyState**). On the contrary it introduces a new simulation class, **FskSimulation**, which can be used to encode deterministic, statistic and probabilistic simulations.

- Deterministic simulations are based on point-estimate models. Only point estimates with a fixed, determined, value are used.
- Statistic simulations are used for descriptive analysis of observational data. These simulations are meant to describe and analyse data.
- Probabilistic simulations are based on probabilistic methods like Monte Carlo and Bayesian networks.

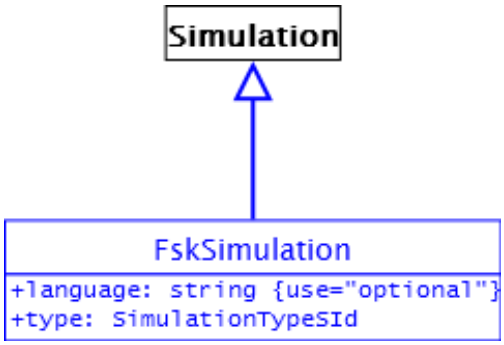

Figure 7 The definition of the FskSimulation class

The **FskSimulation** class derives two attributes from **Simulation**, id and name, which are mandatory and optional respectively. It also includes two new attributes: `language` and `type`.

`language`

`language` is an optional attribute that indicates the scripting language in which the model to simulate was written. It takes free text.

`type`

`type` is a mandatory attribute that indicates the type of the simulation. The value of the attribute must be taken from the list of reserved words in … These reserved symbols are defined in the value space of the data type `SimulationTypeSId`.

deterministic
statistic
probabilistic

Table 13 Types for FskSimulation

```
<listOfSimulations>
    <fskSimulation id="Simulation1" name="Virus Decay" language="R" type=”deterministic”/>
</listOfSimulations>
```

Table 14 Example of the FskSimulation class

4.3. Task

The **Task** class defines the combinations of Models and Simulations considered in the current FSK-SED-ML file.

4.4. DataGenerator

The **DataGenerator** class defines which values from which tasks should be considered for output. The post-processing of values may also be defined in order to bring values in appropriate form for later output.

4.5. Script

The **Output** class defines how the values specified in the **DataGenerator** are plotted. FSK deviates from classical SED-ML providing a new output component **Script**. The Script class references to a separate script file containing the plotting commands in the corresponding script language.

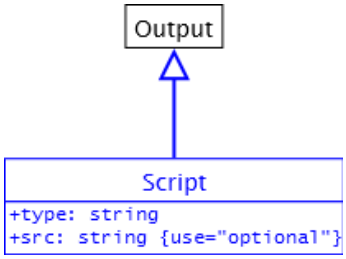

Figure 8 The definition of the OutputScript class

The **Script** class derives two attributes from Output, id and name, which are mandatory and optional respectively. It also includes two new attributes, `type` and `src`.

`type`

`type` is a mandatory attribute that identifies the scripting language of code embedded within a script element or referenced through the `src` attribute. It is specified with MIME types. Examples of MIME types are `text/x-r` and `text/x-python` for the R and Python languages respectively.

Language	MIME type
R	`text/x-r`
Python	`text/x-python`
Matlab	`text/x-matlab`

Figure 9 MIME types of interest

`src`

src is an optional attribute that indicates the URI of an external script, and represents an alternative to embedding the script directly in the SED-ML document. When the src attribute is specified, the script element should not include a script within its tag.

```
<listOfOutputs>
  <script id="plot1" language="text/x-python" src="visualization-script.py" />
  <script id="plot2" language="text/x-r">
    hist(result, breaks=50, main="PREVALENCE OF PARENTS FLOCKS", xlab="Prevalence", col="32")
  </script
</listOfOutputs>
```

Figure 10 Examples of the Script class with referenced and embedded scripts

4.6. Examples

```
<listOfSimulations>
    <Deterministic id="Simulation1" name="Virus Decay" language="R" />
</listOfSimulations>

<listOfModels>
    <model id="Model1" name="Virus Decay Model Brookes" source="VirusDecay.fskx">
        <listOfChanges>
            <changeAttribute target="Model1@temp" newValue="30" />
            <changeAttribute target="Model1@time" newValue="80" />
        </listOfChanges>
    </model>
</listOfModels>

<listOfTasks>
    <task id="Task1" modelReference="Model1" simulationReference="Simulation1" />
</listOfTasks>

<listOfDataGenerators>
    <dataGenerator id="VirusEndConcentration" name="Virus concentration after decay">
        <listOfVariable>
            <variable id="DecayedConc" taskReference="task1" target="Model1@V_end" />
        </listOfVariable>
    </dataGenerator>
</listOfDataGenerators>

<listOfOutputs>
    <script id="plot1" language="R" command="reference to visualisation script" />
</listOfOutputs>
```

```
<listOfSimulations>
    <deterministic id="Simulation1" name="PRRSV Decay-DoseResponse Combination"
        language="R" />
</listOfSimulations>

<listOfModels>
    <model id="Model1" name="Virus Decay" source="VirusDecay.fskx">
        <listOfChanges>
            <changeAttribute target="Model1@V_init" newValue="4" />
        </listOfChanges>
    </model>

    <model id="Model2" name="Dose Response" source="DoseResponse.fskx">
        <listOfChanges>
            <changeAttribute target="Model2@Dose" newValue="Model1@V_end" />
        </listOfChanges>
    </model>
</listOfModels>

<listOfTasks>
    <task id="Task1" modelReference="Model1" name="intermediate result"
        simulationReference="Simulation1" />
    <task id="Task2" modelReference="Model2" name="end result"
        simulationReference="Simulation1" />
</listOfTasks>

<listOfDataGenerators>
    <dataGenerator id="EndResult" name="Probability of Infection">
        <listOfVariables>
            <variable id="InfectProbability" taskReference="Task2"
                target="Model2@P_Infect_Dose" />
        </listOfVariables>
    </dataGenerator>

    <dataGenerator id="IntermediateResult" name="VirusConcentration">
        <listOfVariables>
            <variable id="VirusConc" taskReference="Task1"
                target="Model1@V_end" />
        </listOfVariables>
    </dataGenerator>
</listOfDataGenerators>

<listOfOutputs>
    <script id="plot1" language="R" command="reference to visualisation script" />
    <script id="plot2" language="R" command="reference to visualisation script" />
</listOfOutputs>
```

5. Example models

5.1. Virus decay example model

This is a parameterised virus decay model which gives the end concentration of PRRS Virus (V_{end}) after a certain time that meat with an initial virus concentration (V_{init}) was stored at a certain temperature *temp*.

$$V_{end}(time) = V_{init} \times e^{-\frac{time}{t_{1/2}}}$$

With $t_{1/2}(time) = lambda_0 \times e^{-lambda \times temp}$

The virus decay process is modelled as time and temperature dependent exponential decay. Here $t_{1/2}$ is the temperature dependent half-life of PRRSV in minimum essential medium. $lamba_0$ =243.54 (h) and lambda = 0.109 (per °C) are PRRSV-specific parameters. 243.54 hours is the half-life of PRRSV at 0 °C and 0.109/°C is the decay rate of PRRSV half-life for temperatures between 0 °C and 34 °C. For calculations "time" must be given in hours and "temp" in °C.

V_init is the initial Virus concentration in meat. The model has three variables and two parameters with following default values.

- Input:

Variables	*V_init=10 log10 TCID50/kg*
	temp=20 °C
	Time = 60 h
Parameter	*lambda0= 243.54 h*
	lambda = 0.109/°C

- Output

Variables	*V_end*

5.2. Dose response example model

This example defines a parameterized dose-response model which gives the probability of infection after ingestion of PRRS virus by pigs for a given dose.

$$P(Infect|Dose) = 1 - \left(1 + \frac{Dose}{beta}\right)^{-alpha}$$

Here *alpha=0.3* and *beta=14400* (*log10 TCID50*) are PRRS and pig specific parameters. The equation represents a beta-poisson response function where the alpha describes the slope of the response curve and the value of beta shifts the curve along the abscissa (Dose axis).

This model has one variable and two parameters with the following default values:

- Input:

Variables	Dose = 4 $\log_{10}$ $TCID_{50}$
Parameter	Alpha = 0.3
	Beta = 14400 $\log_{10}$ $TCID_{50}$

- Output

Variables	P_Infect_Dose

Example metadata:

Name	Dose Response Model for Porcine Reproductive And Respiratory Syndrome Virus
FSK Type	Risk Characterization || dose response model
FSK Identifier	URI / URL of openFSMR or other model repository
Biological Agent(s)	Porcine Reproductive and Respiratory Syndrome Virus (PRRS)
Matrix	Pork (raw)
Process	Infection with PRRS by ingestion of 0.5 kg of Pork (raw)
Model topic and scope	This is a dose-response model which describes the probability that pigs get infected with PRRSV. The model is based on a beta-poisson response function. It is considered to hold for a PRRSV dose from 0 to 8 $\log_{10}$ TCID50/ml in residual blood serum in raw pork.
Stochastic-Modelling?	No
Dependent-Var(s) – Name	PInfectDose
Dependent-Var(s) -Unit	[] (no name)
Dependent-Var(s) - Data type	numeric
Dependent-Var(s) - Description	PInfectDose is the probability that a pig gets infected given that it ingests a certain dose of PRRS virus.
Independent-Var(s) – Name	Dose
Independent-Var(s) – Unit	$\log_{10}$ TCID50

Kommentar [miguel5]: Original type was Double but here I changed it to numeric to follow the defined data types.

Independent-Var(s) – Data type	numeric
Independent-Var(s) – Description	Dose: Concentration of PRRS virus in residual blood in pork when ingested
Parameter(s) – Name	Alpha || beta
Parameter(s) – Unit	[] (no name) || log10 TCID50
Parameter(s) – Data Type	[numeric, numeric]
Parameters – Description	Alpha = Parameter that describes the slope of the beta-poisson dose response function Beta = Parameter which shifts the beta-poisson dose response function
Creator	V.J. Brookes; Charles Sturt University – Wagga Wagga; vbrookes@csu.edu.au
Reference-Description	Brookes VJ, Hernández-Jover, Holyoake, Ward MP (2014) Import risk assessment incorporating a dose-response model: introduction of highly pathogenic porcine reproductive and respiratory syndrome into Australia via illegally imported raw pork. Prev. Vet. Med. 113, 565-579
Created	10.29.2013
Modified	
Rights	Public
Software	R / KNIME
FSK-Curation-Status	No

Table 15 Example metadata

Kommentar [miguel6]: Original type was Double but here I changed it to numeric to follow the defined data types.

Kommentar [miguel7]: Do the parameters have the same type?

References

Bergmann, F., Adams, R., Moodie, S., Cooper, J., Glont, M., Golebiewski, M., et al. (2014). *COMBINE archive and OMEX format: one file to share all information to reproduce a modeling project.* (B. Bioinformatics, Ed.) Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/25494900

Bergmann, F., Cooper, J., Le Novère, N., Nickerson, D., & Waltermath, D. (2015). Simulation Experiment Description Markup Language (SED-ML) Level 1 Version 2. *Journal of intergrative Bioinformatics*, 262.

Bergmann, F., Rodriguez, N., & Le Novère, N. (2015). *COMBINE Archive Specification Version 1.* Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/26528559

Cyganiak, R., Wood, D., Lanthaler, M., Klyne, G., Carroll, J. J., & and McBride, B. (2014). *RDF 1.1 concepts and abstract syntax.* Retrieved from https://www.w3.org/TR/rdf11-concepts/

Freed, N. a. (1996). *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.* Retrieved from www.iana.org/assignments/media-types/media-types.xhtml

Hucka, M., Bergmann, F. T., Hoops, S., Keatin, S., Sahle, S., Schaff, J., et al. (2010). *The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core.* Retrieved from http://identifiers.org/combine.specifications/sbml.level-3.version-1.core.release-1

O' Dada, J. (2011). *Numerical Markup Language.* Retrieved from http://code.google.com

OMG. (2009, February). *UML 2.2 Superstructure and Infrastructure.* Retrieved from http://www.omg.org/spec/uml/2.2

Waltemath, D., Adams, R., Beard, D., Bergmann, F., Bhalla, U., Britten, R., et al. (2011). *Minimum Information About a Simulation Experiment (MIASE).* Retrieved from http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1001122