INCF Task Force for a Standard Language in Neural Network Models

NineML (9ML) version 0.9 General Description

Proposal v.9 ID:AG-YLF-20100709.01

Maintained by Anatoli Gorchetchnikov

July 9, 2010

Abstract

This draft intends to describe our effort to develop a standardized language for Neural Network modeling using a set of reference models. The aim of this first implementation is to design a benchmark for the language using a simple model and then adapt the language to more complex networks.

This work currently summarizes the different ideas and propositions elaborated during five Task Force meetings through an empirical approach and will serve as a basis for further work.

Changes from version 1:

- added footnotes for some issues raised in Stockholm June meeting
- cleaned the `definition` tag syntax according to Stockholm decisions
- added brief description of interactions between user layer and simulation software that does not support abstraction layer
- removed random number generator nodes as decided in Stockholm
- added more examples of random distributions in 1, 2 and 3D
- added random projection when a probability of projection between any cell in source and any cell in target is given
- added space, region, and layout nodes as a first draft of geometrical concepts (based on NetworkML proposal by Padraig and Robert as well as Stockholm discussions)
- removed recurrent projections that can be described on the population level
- cut out the previous approach to cell positioning within population

Contents

1 Introduction

With the increase of the Computational Neuroscience community and the number of work related to neural network modeling, the INCF identified a need for standards and guidelines, which will allow optimized model sharing, insure model reproducibility throughout the different software and is crucial for the scientific good practice. To create such a standard for Neural Network modeling, the INCF created a program aiming at developing a common standardized descriptive language for neural models.

The development of such a language is the first mandate of this Task Force. This language is developed using the modern descriptive technology including XML and possibly UML and should allow an unambiguous definition of the model of network both conceptually and mathematically and insure complete software independence. In addition this language is designed to be self-consistent and highly flexible, allowing addition of new models and mathematical descriptions without modification of the previous structure and organization of the language. This standard is a two layered language: a user description layer which represents the conceptual description of the model in biological terms and a core semantics (abstraction layer) which will define the mathematics behind each of the concepts defined at the user layer level.

User description layer is intended to be primarily machine readable. XML syntax defined in this document is designed with considerations of ease of parsing, verification, and automatic model construction in mind. This does not prevent the advanced user from editing the user layer XML descriptions by hand, but the primary means for creation of these descriptions are the software tools that will convert the GUI-based representations of objects and properties into a valid XML code.

The roles of two layers are delineated clearly by implementing an object oriented type of coding, the mathematical formalism being described in the core semantics (abstraction layer) and referenced in the user layer. This approach helps to complete the self-consistency requirement as any term used in the user layer is automatically linked to its formal definition in the core semantics. In this manuscript we focus on the organization of the user layer of the language, provide the syntax of linking with the core semantics, and describe in mathematical terms what the corresponding core semantic objects are.

In order to achieve the ultimate goals of the task force, meaning creating a first version of the language and propose it to the community of modelers, we decided to constrain our approach by selecting few seminal papers to design a working version of the language. The first specification presented here is based on several models selected by the Task Force and aim to be further tested by extending the network repertoire to more complex networks. This insures that the language presents a logical organization, which makes it robust to new additions and further modifications.

This manuscript so far only covers modifications of two simple forms of spiking neurons: parametrizations of the leaky integrate and fire, two types of postsynaptic responses: the alpha synapse and the delta synapse, only one simple network architecture, four variations of a random connectivity pattern, and one plasticity rule. Throughout the process of verification of this first implementation we will then expand the number of models to all the common forms of spiking neurons and synapses, widely used in the literature, as well as different types of network architecture.

In order to achieve this purpose, we agreed on the need of ontology of the neural network, requiring the compilation of all or most of the publication on neural network modeling. This work currently in progress will also lead to a better definition of terminology the standard should use. We agreed that the use of 'layer' or 'population' is confusing and we chose to use neutral terms such as 'set' or 'group'. Further refinement of these terms can be based on the current state-of-art of the neural network modeling. In the same time, the current version of the language does not cover much of the initialization and run-time specification. NineML is designed to provide network description rather than simulation description,

although there are cases when initialization of certain variables is indeed part of the model rather than of a simulation setup. Drawing the line between these two will happen during the development process using an extended library of neural networks.

The first part of this document presents descriptions of single elements (cells, synapses, inputs) with examples of user layer code, core semantics behind this code, and discussion of important issues. The second part deals with the grouping of these individual entities while the third part addresses the proposition for the connectivity description. For these last two parts the core semantics should deal with the geometrical representation of cell arrangement as well as coordinate systems in the feature space. These two parts of core semantics are not currently implemented and thus are omitted from the current version of the manuscript.

Appendices to this document present first implementations of the seminal models using the syntax defined in this specification. The aim of these implementations is to design a benchmark for the language, starting with simple models and progressively adapt the language to more complex networks. Formal grammar definition shall also be provided in the appendix.

2 Nodes and Properties

The basic building block of NineML is called node[1]. In the user layer description node is a reference to an object defined in the abstraction layer. Abstraction layer defines the mathematics of the node and is referred in the user layer by `definition` tag with type set to "NineML":

```
<definition type="NineML">http://www.NineML.org/neurons/IaF_tau.xml</definition>
```

For the ease of extensibility of the language the standard allows to put references to user space definitions. For example user defines some neuron through syntax used by abstraction layer in a separate file and then refers to it in the `definition` similar to how references to NineML database definitions are made:

```
<definition type="NineML">file://home/someuser/neurons/My_IaF.xml</definition>
```

Also for the purpose of extensibility the initial version of the standard allows external object definitions. Since these are completely outside of NineML and implementation dependent, the `definition` tag has an attribute `type` that has to be recognized by the parser of a specific simulator. No further assumptions about the contents of such definitions are made.

```
<definition type="ndf">
  ...
</definition>
```

The procedure shall be established to ensure that these external definitions are translated into NineML definitions that can be used by multiple simulators as the time progresses.

Each of the nodes has a user-given ID or `name` and a set of properties that is defined in the above definition and instantiated in the user layer. Since these properties are unique for every definition, the tags that are used for them are predefined by NineML for abstraction layer definitions and by 3rd party for external definitions[2]. The abstraction layer specification or the documentation for the external definition provides the mapping between mathematical description of the object and the corresponding tags. Here some of these mappings for NineML abstraction layer definitions are also provided.

User can add short notes to each node description. These notes are intended to provide a specific reference to the research paper page where the node is described and similar kind of information. These notes are not intended to duplicate the core semantics documentation or any other documentation, thus they shall not provide mathematical description and other details of the node implementation. Note can contain text or link to an Internet resource.

As a result, a complete description of a node in the user layer consists of three parts: reference to a definition, a list of properties, and an optional note:

```
<node name="E_ext">
  <definition type="NineML">http://www.NineML.org/neurons/Poisson.xml</definition>
  <properties>
    <rate>...</rate>
  </properties>
```

[1] Alternative terms: component, instantiation, entity

[2] Issue raised during June Stockholm meeting whether this shall be the case

```
  <note><text>Poisson input (page 185)</text></note>
</node>
```

The page number in the note above as well as in other note examples in this document refer to [Brunel, 2000] unless stated otherwise.

Hugo: Use identical syntax for name and for other properties, e.g:

```
<node>
  <definition type="NineML">http://www.NineML.org/neurons/Poisson.xml</definition>
  <properties>
    <name>E_ext</name>
    <rate>...</rate>
    <random>...</random>
  </properties>
  <note><text>Poisson input (page 185)</text></note>
</node>
```

To reduce the size of the resulting network description user can refer to already described node by user-given name instead of providing link to abstraction layer. In this case the properties of the node that have to be redefined are stated explicitly, the properties that are inherited from the original description are omitted:

```
<node name="E_ext2">
  <reference>E_ext</reference>
  <properties>
    <rate>...</rate>
  </properties>
</node>
```

In this case the node follows the same description as the one shown in the previous listings except for different `rate`. Note that the reference is done by the user-dened ID (`name`).

If the simulator only supports user layer, then the simulator developers can create mappings directly between the reference to a definition in the user layer description of the node and the intrinsic simulator code that implements the same mathematics.

2.1 Property Description

NineML uses predefined tags for properties. These tags are provided by the abstraction layer or external definition of the parent node. The user can set the value of the property as well as (where applicable) the units of measurements. There is no need to provide a user-given ID to a property because it can always be uniquely identified by the parent node ID and its own tag name. User can add a note to a property in a way similar to adding notes to nodes described above.

There are two kinds of properties: values of the first kind are given to the model by the user and stay fixed, values of a second kind are computed within the model during simulation. For all practical reasons the syntax of the user layer descriptions is identical for both kinds of properties. For example

```
<rate>
  <value>10</value>
  <unit>Hz</unit>
</rate>
```

sets a parameter value to a fixed number, while

```
<voltage>
  <value>-60</value>
  <unit>mV</unit>
</voltage>
```

sets the initial value of the variable. Please note that rate above still uses Hz as units because the syntax for spikes/sec is not formalized yet.

Issue: Syntax shall be defined to provide compound descriptions of both values and units, for example when relative conductance is defined in mS/cm^2 or value is computed from some other values. See section 3.1 for an example of possible syntax.

As NineML does not provide any default values for properties, it is a job of the user to provide initial values for all defined properties. To ensure the integrity of the model NineML requires all initial values to be set in the user layer description. For batch simulations and other modifications any of the values given in the user layer can be overwritten by a simulation setup description, but this is outside of the scope of the current version of NineML.

Properties described above have fixed values for all instances of the respective node and are determined at description level. In addition to that some properties may vary between different instances based on some random distribution, spatial coordinate, geometry, and other values that are determined at the time each instance is created. These can be termed instantiation level properties and are discussed in the following subsections.

2.1.1 Properties with Values Drawn from Random Distributions

These are the simplest out of instantiation level properties because they do not depend on some other properties outside of their parent node. A reference to a random distribution which is the same for all instances of the node is sufficient to set the value for this type of properties:

```
<voltage>
  <value>
    <reference>uniform_55-65</reference>
  </value>
  <unit>mV</unit>
</voltage>
```

Here the property value refers to a definition by reference to an object described elsewhere in the network that defines a uniform distribution as discussed in section 2.2.

2.1.2 Properties with Values Dependent on Geometry of the Network

In the simlest case the property of this type refers to a spatial coordinate of the particular instance and calculates its value from this coordinate. Here the reference to the coordinate system node shall be given. This reference should match one of the coordinate systems that this instance is positioned on. Also some function has to be described that allows the transformation of one, two, or three dimensional coordinate value into a single property value. This function can be defined using an inline abstraction layer definition, reference to abstraction layer database or MathML definition.

More complex cases can stem from here if coordinate itself is insufficient to define the property, for examle the distance between this instance and some other instance possibly in another population determines the property value. NineML shall insure that both populations are layed out on the same coordinate systems, reference to the system used shall be provided as above (but is necessary only if multiple systems used for population).

These are preliminary thoughs, until specific examples will be derived no syntax can be determined.

2.2 Random Distribution Nodes

This set of nodes allows to define random distributions with corresponding parameters.

2.2.1 1D Uniform Distribution

Provides 1D uniform distribution.

Core Semantics Definition This distribution generates random numbers uniformly distributed between `lowerBound` and `upperBound`.

User Layer Description

```
<node name="uniform_-55-65mV">
  <definition type="NineML">http://www.NineML.org/distributions/Uniform1D.xml</definition>
  <properties>
    <lowerBound>
      <value>-65</value>
      <unit>mV</unit>
```

```
    </lowerBound>
    <upperBound>
      <value>-55</value>
      <unit>mV</unit>
    </upperBound>
  </properties>
</node>
```

2.2.2 2D Uniform Distribution

Provides 2D uniform distribution.

Core Semantics Definition This distribution generates random numbers uniformly distributed within a 2D rectangle defined by `lowerBoundX`, `lowerBoundY` and `upperBoundX`, `upperBoundY`.

User Layer Description

```
<node name="2D uniform">
  <definition type="NineML">http://www.NineML.org/distributions/Uniform2D.xml</definition>
  <properties>
    <lowerBoundX>
      <value>-65</value>
      <unit>mV</unit>
    </lowerBoundX>
    <upperBoundX>
      <value>-55</value>
      <unit>mV</unit>
    </upperBoundX>
    <lowerBoundY>
      <value>-65</value>
      <unit>mV</unit>
    </lowerBoundY>
    <upperBoundY>
      <value>-55</value>
      <unit>mV</unit>
    </upperBoundY>
  </properties>
</node>
```

2.2.3 3D Uniform Distribution

Provides 3D uniform distribution.

Core Semantics Definition This distribution generates random numbers uniformly distributed within a 3D parallelepiped defined by `lowerBoundX`, `lowerBoundY`, `lowerBoundZ` and `upperBoundX`, `upperBoundY`, `upperBoundZ`.

User Layer Description

```
<node name="2D uniform">
  <definition type="NineML">http://www.NineML.org/distributions/Uniform3D.xml</definition>
  <properties>
    <lowerBoundX>
      <value>-65</value>
      <unit>mV</unit>
    </lowerBoundX>
    <upperBoundX>
      <value>-55</value>
      <unit>mV</unit>
    </upperBoundX>
```

```
    <lowerBoundY>
      <value>-65</value>
      <unit>mV</unit>
    </lowerBoundY>
    <upperBoundY>
      <value>-55</value>
      <unit>mV</unit>
    </upperBoundY>
    <lowerBoundZ>
      <value>-65</value>
      <unit>mV</unit>
    </lowerBoundZ>
    <upperBoundZ>
      <value>-55</value>
      <unit>mV</unit>
    </upperBoundZ>
  </properties>
</node>
```

2.2.4 1D Normal Distribution

Provides 1D normal (Gaussian) distribution.

Core Semantics Definition This distribution generates random numbers according to a 1D normal distribution with mean `mean` and standard deviation `stdDev`.

User Layer Description

```
<node name="normal_1.3-0.25">
  <definition type="NineML">http://www.NineML.org/distributions/Normal1D.xml</definition>
  <properties>
    <mean>
      <value>1.3</value>
      <unit>mV</unit>
    </mean>
    <stdDev>
      <value>o.25</value>
      <unit>mV</unit>
    </stdDev>
  </properties>
</node>
```

2.2.5 2D Normal Distribution

Provides 2D normal (Gaussian) distribution.

Core Semantics Definition This distribution generates random numbers according to a 2D normal distribution with mean `meanX`, `meanY` and standard deviations `stdDevX`, `stdDevY`.

User Layer Description

```
<node name="2D normal">
  <definition type="NineML">http://www.NineML.org/distributions/Normal2D.xml</definition>
  <properties>
    <meanX>
      <value>1.3</value>
      <unit>mV</unit>
    </meanX>
    <stdDevX>
      <value>o.25</value>
```

```
      <unit>mV</unit>
    </stdDevX>
    <meanY>
      <value>1.3</value>
      <unit>mV</unit>
    </meanY>
    <stdDevY>
      <value>o.25</value>
      <unit>mV</unit>
    </stdDevY>
  </properties>
</node>
```

2.2.6 3D Normal Distribution

Provides 3D normal (Gaussian) distribution.

Core Semantics Definition This distribution generates random numbers according to a 3D normal distribution with mean `meanX`, `meanY`, `meanZ` and standard deviations `stdDevX`, `stdDevY`, `stdDevZ`.

User Layer Description

```
<node name="normal_1.3-0.25">
  <definition type="NineML">http://www.NineML.org/distributions/Normal3D.xml</definition>
  <properties>
    <meanX>
      <value>1.3</value>
      <unit>mV</unit>
    </meanX>
    <stdDevX>
      <value>o.25</value>
      <unit>mV</unit>
    </stdDevX>
    <meanY>
      <value>1.3</value>
      <unit>mV</unit>
    </meanY>
    <stdDevY>
      <value>o.25</value>
      <unit>mV</unit>
    </stdDevY>
    <meanZ>
      <value>1.3</value>
      <unit>mV</unit>
    </meanZ>
    <stdDevZ>
      <value>o.25</value>
      <unit>mV</unit>
    </stdDevZ>
  </properties>
</node>
```

2.2.7 More subtypes here

2.3 Current Nodes

Current nodes describe the dynamics of currents that affect neuronal behavior. Note that these currents do not include synaptic currents that are grouped separately under the post-synaptic response nodes (section 2.6).

Issue: Some post-synaptic responses are currents, others are not. Currently this document describes post-synaptic responses together independently of their nature. Shall it stay this way or shall post-synaptic currents be moved here and other post-synaptic responses left in a separate section?

2.3.1 Injected Current

Currents that are injected in the cell by experimenters during some protocols.

Core Semantics Definition This current does not have internal dynamics and has a single property `magnitude`. How to manipulate this current externally (input protocol) is under discussion. It appears that onset and offset times are important for reproduceability of the results, but on the other hand they belong more to the simulation description rather then model description.

User Layer Description

```
<node name="Inject">
  <definition type="NineML">http://www.NineML.org/currents/Injection.xml</definition>
  <properties>
    <magnitude>
      <value>10</value>
      <unit>uA</unit>
    </magnitude>
  </properties>
</node>
```

2.3.2 More subtypes here

2.4 Spike-Producing Nodes

2.4.1 Simple Poisson Spike-Producing Node

An approximation of the neuron used as a source of input in many models (e.g. [Brunel, 2000]).

Core Semantics Definition Poisson generator is described as homogeneous Poisson process where the waiting time till each next spike is distributed exponentially

$$P(T_{next}) = -\nu t \tag{1}$$

and has a single parameter ν in the equation (1) that corresponds to user layer tag `rate`.

User Layer Description

```
<node name="E_ext">
  <definition type="NineML">http://www.NineML.org/neurons/Poisson.xml</definition>
  <properties>
    <rate>
      <value>10</value>
      <unit>Hz</unit>
    </rate>
  </properties>
  <note><text>Poisson input (page 185)</text></note>
</node>
```

This code refers to the core semantic definition of the Poisson spiker, sets the parameter `rate`.

2.4.2 Leaky Integrate-and-Fire Defined Through Membrane Time Constant

Core Semantics Definition Proposed type name

`IaF_tau`

represents IaF node with default dynamics described as

$$\tau \frac{dV}{dt} = -V \tag{2}$$

If voltage $V > \theta$ the neuron emits a spike, resets the voltage to V_r, and enters a refractory state for t_r. Correspondence between properties and tag names:

V `voltage`

t_r `refractoryTime`

V_r `resetPotential`

θ `threshold`

τ `membraneConstant`

Equation (2) has to be extended to allow addition of all currents the neuron might have. Thus the place holders for these currents are provided:

$$\tau \frac{dV}{dt} = -V + \frac{\tau}{C} \sum_i I_i \tag{3}$$

which adds a property

C `membraneCapacitance`

and an arbitrary number of currents.

> **Issue:** Something has to be done with this property to distinguish it from properties above that have to be always present. This one is only necessary if currents are present, maybe bind it to currents block?

Delta synapses that operate in terms of voltage steps do not produce currents, so they have to be added separately. Again, place holders are provided in the equation

$$V = V + \sum_j V_j \tag{4}$$

User Layer Description

```
<node name="E">
  <definition type="NineML">http://www.NineML.org/neurons/IaF_tau.xml</definition>
  <properties>
    <voltage>...</voltage>
    <threshold>...</threshold>
    <membraneConstant>...</membraneConstant>
    <refractoryTime>...</refractoryTime>
    <resetPotential>...</resetPotential>
    <membraneCapacitance>...</membraneCapacitance>
  </properties>
  <currents>
    <current>
      <definition type="NineML">http://www.NineML.org/currents/Injection.xml</definition>
      <properties>
        <magnitude>
          <value>20</value>
          <unit>uA</unit>
        </magnitude>
      </properties>
    </current>
    <current>
      <reference>Inject</reference>
    </current>
    ...
  </currents>
</node>
```

This code refers to a description of current `"Inject"` in section 2.3.1 as well as defines an additional injection current. Both types of descriptions are allowed by NineML, but for the current used in multiple neurons the reference to a current described separately is more compact and preferred.

Please note that description of the neuronal node above defines a prototypical neuron which can be reused multiple times within the network. As a consequence, multiple connectivity patterns can be applied to this neuron, which results in a different set of synaptic inputs. Therefore, no description of these inputs shall be provided at the node level, they shall be described at the projection level. Simulation software shall take care of complete construction of each neuron by analyzing both levels of description.

Issue: It is probably better to describe injected currents in a fashion similar to other inputs, so that the neuronal description only includes internal currents specific for this cell type. The description above will be corrected as soon as example descriptions of internal currents will be complete.

2.4.3 Leaky Integrate-and-Fire Defined Through Leakage Conductance

Core Semantics Definition Proposed type name

`IaF_gL`

represents IaF node with default dynamics described as

$$C_m \frac{dV}{dt} = -g_L V \tag{5}$$

If voltage $V > \theta$ the neuron emits a spike, resets the voltage to V_r, and enters a refractory state for t_r. Correspondence between properties and tag names:

V	`voltage`
C_m	`membraneCapacitance`
t_r	`refractoryTime`
V_r	`resetPotential`
θ	`threshold`
g_L	`leakageConductance`

Equation (5) has to be extended to allow addition of all currents the neuron might have. Thus the place holders for these currents are provided:

$$C_m \frac{dV}{dt} = g_L(E_L - V) + \sum_i I_i \tag{6}$$

Issue: I added E_L in this equation but no tag so far is provided for this property.

Delta synapses that operate in terms of voltage steps do not produce currents, so they have to be added separately. Again, place holders are provided, but this time they have to operate not on the derivative of voltage but on voltage itself as in the equation

$$V = V + \sum_j V_j \tag{7}$$

User Layer Description

```
<node name="E">
  <definition type="NineML">http://www.NineML.org/neurons/IaF_gL.xml</definition>
  <properties>
    <voltage>...</voltage>
    <threshold>...</threshold>
    <leakageConductance>...</leakageConductance>
    <refractoryTime>...</refractoryTime>
    <resetPotential>...</resetPotential>
    <membraneCapacitance>...</membraneCapacitance>
  </properties>
  <currents>
```

```
    <current>
      <reference>Inject</reference>
    </current>
    ...
  </currents>
</node>
```

See discussion in the section 2.4.2 for the limitations on the list of currents described within the neuronal node description.

2.4.4 More subtypes here

2.5 Plasticity Nodes

Plasticity nodes handle the synaptic weight and its possible modification. Note that synaptic weight is a dimensionless quantity and as such does not include units.

2.5.1 Static Synapse

Often the synapses in various models are held at fixed weights. This is achieved with static synapse descriptions.

Core Semantics Definition

User Layer Description

```
<node name="Excitatory">
  <definition type="NineML">http://www.NineML.org/weights/Static.xml</definition>
  <properties>
    <weight>
      <value>45.61</value>
    </weight>
  </properties>
</node>
```

2.5.2 Power Law STDP Synapse

Issue: Add discussion of spike sampling regimes (all, nearest N, etc).

Core Semantics Definition The definition follows [Morrison et al., 2007]:

$$\begin{aligned} \Delta w_+ &= \lambda w_0^{1-\mu} w^{\mu} e^{-\frac{|\Delta t|}{\tau_+}} \quad if \quad \Delta t > 0 \\ \Delta w_- &= -\lambda \alpha w e^{-\frac{|\Delta t|}{\tau_-}} \quad if \quad \Delta t < 0 \end{aligned} \tag{8}$$

Issue: Add a bit on how it works, what makes it different from additive and multiplicative rules.

Correspondence between properties and tag names:

w	`weight`
w_0	`referenceWeight`
τ_+	`potTimeConst`
τ_-	`depTimeConst`
μ	`power`
λ	`learningRate`
α	`asymmetry`

User Layer Description

```
<node name="Plastic">
  <definition type="NineML">http://www.NineML.org/synapses/PowerAA_STDP.xml</definition>
  <properties>
    <potTimeConst>
      <value>20</value>
      <unit>ms</unit>
    </potTimeConst>
    <depTimeConst>
      <value>20</value>
      <unit>ms</unit>
    </depTimeConst>
    <power>
      <value>0.4</value>
      <unit>dimensionless</unit>
    </power>
    <learningRate>
      <value>0.1</value>
      <unit>dimensionless</unit>
    </learningRate>
    <asymmetry>
      <value>0.11</value>
      <unit>dimensionless</unit>
    </asymmetry>
    <weight>
      <value>45.61</value>
      <unit>dimensionless</unit>
    </weight>
    <referenceWeight>
      <value>17</value>
      <unit>dimensionless</unit>
    </referenceWeight>
  </properties>
</node>
```

2.5.3 More subtypes here

2.6 Post-synaptic Response (PSR) Nodes

Post-synaptic response nodes define the effect imposed on the post-synaptic cell dynamics by triggering the synaptic input. This definition includes only the shape of this effect and does not include the exact magnitude of the effect. The magnitude is described separately through plasticity rules and synaptic weights in section 2.5. Note though that the units of the effect are defined here and synaptic weights are left dimensionless. This is done to allow same plasticity rules to operate on different types of postsynaptic responses.

> **Issue:** Shall the post-synaptic delay be a property of PSR nodes or of connectivity nodes?

NineML defines the following post-synaptic response nodes:

2.6.1 Delta PSR

This is an approximation of synaptic function by a voltage step.

Core Semantics Definition

User Layer Description

```
<node name="Delta PSR">
  <definition type="NineML">http://www.NineML.org/synapses/Delta.xml</definition>
  <properties>
    <step>
      <value>1</value>
      <unit>mV</unit>
    </step>
  </properties>
</node>
```

Issue: The actual magnitude of the step here is given by a synaptic weight, but the units are set inside delta synapse, since weight is dimensionless. As such the `value` tag above does not have any meaning, or does it? To take it further, a lot of PSRs describe variable that changes between 0 and 1...

2.6.2 Alpha Current PSR

This is an approximation of synaptic current by an alpha function.

Core Semantics Definition This current described as

$$I = \frac{e}{\tau_\alpha} t e^{-\frac{t}{\tau_\alpha}} \tag{9}$$

where t is the time since presynaptic spike and τ_α is a time constant of the synaptic response. Correspondence between parameter and tag name is

τ_α `timeConstant`

t `time`

User Layer Description

```
<node name="PSC">
  <definition type="NineML">http://www.NineML.org/synapses/AlphaI.xml</definition>
  <properties>
    <time>
      <value>0</value>
      <unit>ms</unit>
    </time>
    <timeConstant>
      <value>0.33</value>
      <unit>ms</unit>
    </timeConstant>
  </properties>
</node>
```

2.6.3 Exponential Conductance PSR

This is an approximation of synaptic conductance by an exponential function.

Core Semantics Definition This conductance described as instantaneously increasing to 1.0 when the spike arrives and decaying according to

$$g = e^{-\frac{t}{\tau_\alpha}} \tag{10}$$

where t is the time since presynaptic spike and τ_α is a time constant of the synaptic response. Correspondence between parameter and tag name is

τ_α `timeConstant`

t `time`

g `conductance`

An additional tag `reversalPotential` represents reversal potential E_R of the synapse that affects the response according to

$$I = g(E_R - V_m) \tag{11}$$

User Layer Description

```
<node name="ExcPSR">
  <definition type="NineML">http://www.NineML.org/synapses/ExponentialG.xml</definition>
  <properties>
    <time>
      <value>0</value>
      <unit>ms</unit>
    </time>
    <conductance>
      <value>0</value>
      <unit>nS</unit>
    </conductance>
    <timeConstant>
      <value>5.0</value>
      <unit>ms</unit>
    </timeConstant>
    <reversalPotential>
      <value>0.0</value>
      <unit>mV</unit>
    </reversalPotential>
  </properties>
</node>
```

2.6.4 More subtypes here

2.7 Connectivity Nodes

Connectivity nodes based on the geometry of source and/or target groups shall include descriptions of the edge handling. All connectivity nodes below have a property `delay` that sets the axonal delay for the corresponding set of projections. Connectivity node definitions in the abstraction layer are likely to be based on Connection Set Algebra, but this does not affect User Layer instantiations.

2.7.1 One-to-One Projection

A simplest way to organize the connectivity. When connecting two populations of equal sizes one projection per neuron in both populations is created. In the simplest case of indexed populations the projection goes from neuron i in source population to neuron i in target population.

> **Issue:** Add here descriptions of addressing with shifts, between populations of different sizes, maybe geometry based.

Core Semantics Definition

User Layer Description

```
<node name="input">
  <definition type="NineML">http://www.NineML.org/connections/One-to-one.xml</definition>
  <properties>
    <delay>
      <value>2</value>
      <unit>ms</unit>
    </delay>
  </properties>
</node>
```

2.7.2 Pair-based Random Projection

Core Semantics Definition Random projection defined based on the fixed probability of projection for each possible pair of cells (one from source set, one from target set).

User Layer Description

```
<node name="random">
  <definition type="NineML">http://www.NineML.org/connections/RandomPair.xml</definition>
  <properties>
    <probability>
      <value>0.1</value>
    </probability>
    <delay>
      <value>2</value>
      <unit>ms</unit>
    </delay>
  </properties>
</node>
```

2.7.3 Number-based Random Projection

Core Semantics Definition Random projection defined based on the fixed number of sources/destinations. If the `direction` is convergent the parameter `number` defines the fixed number of sources that project to each destination cell. If the `direction` is divergent the parameter `number` defines the fixed number of destinations each source cell is projecting to.

> **Issue:** The discussion was on the mailing list about the exact terminology for convergent/divergent, but I am not sure we concluded the issue.

User Layer Description

```
<node name="random fan-in">
  <definition type="NineML">http://www.NineML.org/connections/NBRP.xml</definition>
  <properties>
    <number>
      <value>1000</value>
    </number>
    <direction>convergent</direction>
    <delay>
      <value>2</value>
      <unit>ms</unit>
    </delay>
  </properties>
</node>
```

2.7.4 Fraction-based Random Projection

Core Semantics Definition Random projection defined based on the fraction of sources/destinations. If the `direction` is convergent the parameter `fraction` defines the fraction of all source cells (e.g 10%) that project to each destination cell. If the `direction` is divergent the parameter `fraction` defines the fraction of destination cells each source cell is projecting to.

User Layer Description

```
<node name="random fan-in">
  <definition type="NineML">http://www.NineML.org/connections/FBRP.xml</definition>
  <properties>
    <fraction>
      <value>0.1</value>
    </fraction>
    <direction>convergent</direction>
    <delay>
      <value>2</value>
      <unit>ms</unit>
    </delay>
```

```
  </properties>
</node>
```

2.7.5 More subtypes here

2.8 Space Nodes

Space nodes define the coordinate systems used to set up the geometry of the network. These nodes can be Cartesian, polar, and maybe some other coordinate systems. More than one space nodes can be defined in the same network, for example Cartesian 3D for cortical cells and polar 2D for retinal space.

> **Issue:** In case we need rotated, translated, or scaled versions of the same coordinate system, shall we define a new space node?

> **Issue:** In case user combines two models how shall the space be handled?

2.8.1 1D Cartesian Space

Sets a coordinate line in 1D space.

Core Semantics Definition

User Layer Description

```
<node name="my 1D space">
  <definition type="NineML">http://www.NineML.org/spaces/Cartesian1D.xml</definition>
  <properties>
    ...
  </properties>
</node>
```

> **Issue:** Could not think of any properties yet. Later we might add rotation/translation/scale here...

2.8.2 2D Cartesian Space

Sets a coordinate plane in 2D space.

Core Semantics Definition

User Layer Description

```
<node name="my 2D space">
  <definition type="NineML">http://www.NineML.org/spaces/Cartesian2D.xml</definition>
  <properties>
    ...
  </properties>
</node>
```

2.8.3 3D Cartesian Space

Sets a coordinate axes in 3D space.

Core Semantics Definition

User Layer Description

```
<node name="my 3D space">
  <definition type="NineML">http://www.NineML.org/spaces/Cartesian3D.xml</definition>
  <properties>
    ...
  </properties>
</node>
```

2.8.4 2D Polar Space

Sets a polar coordinate plane in 2D space.

Core Semantics Definition

User Layer Description

```
<node name="my 2D polar space">
  <definition type="NineML">http://www.NineML.org/spaces/Polar2D.xml</definition>
  <properties>
    ...
  </properties>
</node>
```

2.8.5 3D Polar Space

Sets a polar coordinate system in 3D space.

Core Semantics Definition

User Layer Description

```
<node name="my 3D polar space">
  <definition type="NineML">http://www.NineML.org/spaces/Polar3D.xml</definition>
  <properties>
    ...
  </properties>
</node>
```

2.8.6 more subtypes here

2.9 Region Nodes

Region nodes define a finite region given a coodrinate system (space). A reference to a corresponding coordinate system as well as boundaries of a region constitute the properties of a region node.

2.9.1 Line Segment

Sets a line segment in 1D space.

Core Semantics Definition Properties include reference to coordinate system `space`, and boundaries `lowerBound` and `upperBound`

User Layer Description

```
<node name="my line">
  <definition type="NineML">http://www.NineML.org/regions/Line.xml</definition>
  <properties>
    <space>my 1D space</space>
    <lowerBound>
      <value>100</value>
    </lowerBound>
    <upperBound>
      <value>500</value>
    </upperBound>
  </properties>
</node>
```

2.9.2 Axes Aligned Rectangle

Sets a rectangle parallel to axes in 2D space.

Core Semantics Definition Properties include reference to coordinate system `space`, and boundaries `lowerBoundX`, `lowerBoundY`, `upperBoundX` and `upperBoundY`.

User Layer Description

```
<node name="my square">
  <definition type="NineML">http://www.NineML.org/regions/AARectangle.xml</definition>
  <properties>
    <space>my 2D space</space>
    <lowerBoundX>
      <value>100</value>
    </lowerBoundX>
    <upperBoundX>
      <value>500</value>
    </upperBoundX>
    <lowerBoundY>
      <value>100</value>
    </lowerBoundY>
    <upperBoundY>
      <value>500</value>
    </upperBoundY>
  </properties>
</node>
```

2.9.3 Axes Aligned Parallelepiped

Sets a parallelepiped alligned with the axes in 3D space.

Core Semantics Definition Properties include reference to coordinate system `space`, and boundaries `lowerBoundX`, `lowerBoundY`, `lowerBoundZ`, `upperBoundX`, `upperBoundY` and `upperBoundZ`.

User Layer Description

```
<node name="my block">
  <definition type="NineML">http://www.NineML.org/regions/AAPPD.xml</definition>
  <properties>
    <space>my 3D space</space>
    <lowerBoundX>
      <value>100</value>
    </lowerBoundX>
    <upperBoundX>
      <value>500</value>
    </upperBoundX>
    <lowerBoundY>
      <value>100</value>
    </lowerBoundY>
    <upperBoundY>
      <value>500</value>
    </upperBoundY>
    <lowerBoundZ>
      <value>100</value>
    </lowerBoundZ>
    <upperBoundZ>
      <value>500</value>
    </upperBoundZ>
  </properties>
</node>
```

Issue: The last two only work if space is Cartesian, they will fail for polar spaces, but it is not shown anywhere...

2.9.4 Circle

Sets a circle with center and radius in 2D space.

Core Semantics Definition Properties include reference to coordinate system `space`, coordinates of the center `centerX` `centerY`, and `radius`.

User Layer Description

```
<node name="fovea">
  <definition type="NineML">http://www.NineML.org/regions/Circle.xml</definition>
  <properties>
    <space>my 2D polar space</space>
    <centerX>
      <value>0</value>
    </centerX>
    <centerY>
      <value>0</value>
    </centerY>
    <radius>
      <value>500</value>
    </radius>
  </properties>
</node>
```

2.9.5 Sphere

Sets a sphere with center and radius in 3D space.

Core Semantics Definition Properties include reference to coordinate system `space`, coordinates of the center `centerX`, `centerY`, `centerZ`, and `radius`.

User Layer Description

```
<node name="nucleus">
  <definition type="NineML">http://www.NineML.org/regions/Sphere.xml</definition>
  <properties>
    <space>my 3D space</space>
    <centerX>
      <value>100</value>
    </centerX>
    <centerY>
      <value>100</value>
    </centerY>
    <centerZ>
      <value>100</value>
    </centerZ>
    <radius>
      <value>500</value>
    </radius>
  </properties>
</node>
```

2.9.6 more subtypes here

2.10 Layout Nodes

Layout nodes define how neurons or groups of neurons are positioned within a given region of space. Note that both discrede grids and continuous mappings can operate on the same region, so it is possible to create a 3D grid of neurons for one population as well randomly distributed set of neurons from other population within the same region.

Declaring the existence of a population implicitly introduces a notion of indexing for its elements. In general, a layout is simply a mapping from these indices to generalized positions. For spatial layouts it should reference an existing `region` which in turn will reference some coordinate system.

2.10.1 Unstructured Layout

A pseudo-layout for populations where space does not matter at all (like in [Brunel, 2000]).

> **Issue:** Any other layout will break if we will not define space and region, but this one does not refer to region, so it will survive. Does it mean that we can simply omit describing space and region in the networks like Brunel's?

Core Semantics Definition This layout has no properties.

User Layer Description

```
<node name="unstructured">
  <definition type="NineML">http://www.NineML.org/layouts/Unstructured.xml</definition>
  <properties/>
</node>
```

2.10.2 1D Grid Layout

Sets a uniformly spaced layout in 1D region of space.

Core Semantics Definition Since there is only one dimension, the cell number and the region fully define the spacing between cells, thus `region` is the only necessary property. We might consider the case when region and spacing are given and the cell count is calculated in the population based on this information.

User Layer Description

```
<node name="my uniform1D">
  <definition type="NineML">http://www.NineML.org/layouts/Grid1D.xml</definition>
  <properties>
    <region>my 1D region</region>
  </properties>
</node>
```

2.10.3 2D Grid Layout

Sets a uniformly spaced layout in 2D region of space.

Core Semantics Definition Here the cell count has to be a product of `xCount` and `yCount`.

User Layer Description

```
<node name="my uniform2D">
  <definition type="NineML">http://www.NineML.org/layouts/Grid2D.xml</definition>
  <properties>
    <region>my 2D region</region>
    <xCount>
      <value>50</value>
    </xCount>
    <yCount>
      <value>25</value>
    </yCount>
  </properties>
</node>
```

2.10.4 3D Grid Layout

Sets a uniformly spaced layout in 3D region of space.

Core Semantics Definition Here the cell count has to be a product of `xCount`, `yCount`, and `zCount`.

User Layer Description

```
<node name="my uniform3D">
  <definition type="NineML">http://www.NineML.org/layouts/Grid3D.xml</definition>
  <properties>
    <region>my 3D region</region>
    <xCount>
      <value>50</value>
    </xCount>
    <yCount>
      <value>25</value>
    </yCount>
    <zCount>
      <value>10</value>
    </zCount>
  </properties>
</node>
```

2.10.5 1D Random Uniformly Distributed Layout

Sets a uniformly distributed random layout in 1D region of space.

Core Semantics Definition

User Layer Description

```
<node name="my uniform1D">
  <definition type="NineML">http://www.NineML.org/layouts/Uniform1D.xml</definition>
  <properties>
    <region>my 1D region</region>
  </properties>
</node>
```

2.10.6 2D Random Uniformly Distributed Layout

Sets a uniformly distributed random layout in 2D region of space.

Core Semantics Definition

User Layer Description

```
<node name="my uniform2D">
  <definition type="NineML">http://www.NineML.org/layouts/Uniform2D.xml</definition>
  <properties>
    <region>my 2D region</region>
  </properties>
</node>
```

2.10.7 3D Random Uniformly Distributed Layout

Sets a uniformly distributed random layout in 3D region of space.

Core Semantics Definition

User Layer Description

```
<node name="my uniform3D">
  <definition type="NineML">http://www.NineML.org/layouts/Uniform3D.xml</definition>
  <properties>
    <region>my 3D region</region>
  </properties>
</node>
```

2.10.8 more subtypes here

3 Grouping

One of the issues the Task Force faced was to define standardized terms to describe network and network topology. It appeared that the use of words such as layer, population, and such is confusing. The term 'population' is still used below to designate a collection of identical elements because it appears the most common use of it within the Task Force.

Grouping can apply not only to cells but also to synapses, currents, and all other nodes defined in the previous sections. Technically, the grouping of objects is of two major types: **constructive** grouping used to **create** objects and **access** grouping used to define the subgroups of **already created** objects for the purposes of connectivity or monitoring.

To eliminate access groups that fully replicate corresponding constructive groups any constructive group can be used for the access purposes. Both types of grouping allow hierarchical build of larger groups from smaller groups. Rather than having a deep hierarchy, however, the structure is kept flat by referencing. A group does not contain the description of the elements that are repeated, but only refers to their original description.

The rules of construction provided by constructive groups shall not be executed immediately on occurrence of such a group within the description. The process of construction shall proceed from top-most group in the hierarchy down to component groups. This allows the flexibility of defining one constructive group in the description and reusing it multiple times within the network. This also allows the inclusion of developed networks in the larger models as sub-components. The top-most group in the hierarchy is not marked anywhere in the description, it is rather selected by the user through the software interface. This way the user can simulate individual components of the network for debugging and tuning purposes without modifying the rest of the model.

3.1 Constructive Group (`group`)

This is a two-layered structure including of a set of populations on the lower level and connectivity between these populations together with access groups (section 3.3) on the upper level. Each population describes a collection of *identical* elements. Both the number of populations and the number of elements within population can be set to one. Processing of a constructive group by a NineML-compliant software shall lead to creation of instances for all elements, define the spatial layout of the group, and provide internal connectivity if necessary. Constructive groups use tag `group` in user layer descriptions.

A group contains a user-given ID (`name`). The scoping rules for these names are described in section 5.1.

An user layer description example of a three population group with 90000 cells of the first type (`E` marked by `prototype` tag here), 22500 cells of the second type (`I`), and 112500 cells of the third type (`E_ext`):

```
<group name="Network">
  <population name="Exc">
    <prototype><reference>E</reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>90000</number>
  </population>
  <population name="Inh">
    <prototype><reference>I</reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>22500</number>
  </population>
```

```
  <population name="Ext">
    <prototype><reference>E_ext</reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>population[@name="Exc"].number+population[@name="Inh"].number</number>
  </population>
  <projection>...</projection>
  ...
  <projection>...</projection>
</group>
```

In all three populations `prototype` tag refers to a user-given ID (`name`) of the prototypical node. Note also that `prototype`s in different populations can use the same prototypical node, for example in the code above another population consisting of a number of inhibitory cells of type `I` can be added if the user wants to define two populations of interneurons with different connectivity or with different positional distribution.

The line

```
<number>population[@name="Exc"].number+population[@name="Inh"].number</number>
```

is a first attempt on syntax that allows definition of properties through values of other properties.

A collection of `projection` entries describes internal connectivity within this group according to the rules shown in section 4.

Possibility for standard extension: technically, if the prototypes refer to definitions of different compartments, `number`s are set to one and projections are done based on diffusion currents, this group can implement a multi-compartmental cell or a part of such a cell.

3.2 Hierarchy of Constructive Groups

In the previous example group used cells as prototypes, but this is not required. To allow the arbitrary complexity of the model, prototypes in constructive groups can be other constructive groups. Please note that these have to be groups, not populations within a group. An example of a constructive group with 1000 networks of the same type defined above:

```
<group name="Mega Network">
  <population name="Sub Net">
    <prototype><reference>Network</reference></reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>1000</number>
  </population>
  <projection>...</projection>
</group>
```

This example will create total of 90 millions of cells of type `E` and so forth. The set of `projection` descriptions describes internal for this group connectivity according to the rules shown in section 4. Note, that this connectivity is between `Network`s, since within `Network` projections were defined in the description of `Network`.

3.3 Access Group (`set`)

Access group does not define how the cells are created, but rather how they shall be selected for the purpose of monitoring of the the activity or building the connectivity. The tag used by NineML for this kind of group is `set`. Any constructive group declaration also naturally defines a set of all the cells it contains plus one set per each of the populations. Constructive group user-given IDs as well as population user-given IDs can be used anywhere set IDs can, without explicitly defining a set for each constructive group or its population. Declaring a set does not add any objects or nodes to a model specification: it just forms a new, possibly empty, set from those nodes that have already been declared.

3.3.1 Selection by Logical Combinations of Criteria

Within each select statement logical operations are possible. For completeness all three (AND, OR, NOT) shall be allowed. In order to make tags more intuitive to non-mathematical users the suggested tags are

`any` for logical OR operation, where match of any of the given criteria is sufficient to select an element;

`all` for logical AND operation, where match of all given criteria is necessary to select an element;

`not` for logical NOT operation, where only elements that do not match the criteria are selected.

3.3.2 Selection by a User-given ID

Here is an example of a set where the only matching criteria are user-given IDs (`name`) that can be found while descending the tree within the parent group of the set description. It only selects internal cells and excludes external input cells:

```
<group name="Network">
  ...
  <set name="IaF neurons">
    <select>
      <any>
        <equal>
          <value>population[@name]</value>
          <value>Pyramidal Neurons</value>
        </equal>
        <equal>
          <value>population[@name]</value>
          <value>Interneurons</value>
        </equal>
      </any>
    </select>
  </set>
  ...
</group>
```

This set can be used both to record activity of integrate-and-fire cells ignoring the Poisson spikers and select the targets of external input to this network.

> **Issue:** Note though that in all recent examples `select` is the only statement in `set` so we might just drop the `select` tag:

```
<set name="IaF neurons">
  <any>
    <equal>
      <value>population[@name]</value>
      <value>Pyramidal Neurons</value>
    </equal>
    <equal>
      <value>population[@name]</value>
      <value>Interneurons</value>
    </equal>
  </any>
</set>
```

Multiple syntactic constructs were discussed so far with the final choice between them left for later discussions. When all matches are exhausted, the union of each such binding is constructed and returned. Note that the way `IaF Neurons` is defined it consists of 112500 cells.

3.3.3 Selection by a Cell Property

This selection only makes sense for cells that do have certain property defined in their parameter list. If later the model will be changed and a different neuronal node will be used as a prototype, this select statement will become invalid. The user shall be notified of this inconsistency by a simulation software and resolve the problem manually. Furthermore, if the property in question has units of measurements attached to it, the selection shall also include units.

> **Issue:** Given that these sets are easily invalidated by changing the underlying definitions maybe we shall forbid them.

An example syntax for selecting cells in two populations based on their membrane potential is:

```
<select>
  <any>
    <all>
      <equal>
        <value>population[@name]</value>
        <value>Pyramidal Neurons</value>
      </equal>
      <less than>
        <property>population.prototype.voltage.value</property>
        <value>15</value>
        <unit>mV</unit>
      </less than>
    </all>
    <all>
      <equal>
        <value>population[@name]</value>
        <value>Interneurons</value>
      </equal>
      <less than>
        <property>population.prototype.voltage.value</property>
        <value>15</value>
        <unit>mV</unit>
      </less than>
    </all>
  </any>
</select>
```

Note that this particular set will have a variable structure that changes on every step of the simulation.

Issue: Shall NineML allow this kind of sets?

Similar syntax

```
<select>
  <all>
    <equal>
      <value>population[@name]</value>
      <value>Pyramidal Neurons</value>
    </equal>
    <less than>
      <property>population.prototype.threshold.value</property>
      <value>-35</value>
      <unit>mV</unit>
    </less than>
  </all>
</select>
```

will lead to a set fully defined at the start of the simulation and constant for the same values of threshold parameter.

Note that for properties that are assigned values when nodes are instantiated sets only can be created after all cells in the population are instantiated. Same is true for the selection by geometry.

3.3.4 Selection by Geometry

The main issue with this type of selection is that unlike properties, positions are never explicitly mentioned in the description. As a result we cannot access them using the same synatax as above. Below I show an example sysntax how to extract coordinate in some space for comparison and leave empty the part where we supposed to extract cell coordinate.

```
<select>
  <less than>
```

```
    <distance>
      <space>my 2D space</space>
      <from>
        <posX>155</posX>
        <posY>275</posY>
      </from>
      <to><position>...</position></to>
    </distance>
    <value>150</value>
    <unit>um</unit>
  </less than>
</select>
```

There should also be some mechanism that allows to verify that indeed the cells that we are looking at are layed out on the given coordinate system.

> **Issue:** For topographic projections the position of the source within its group (population) partially determines the position of the target within a different group (population). In this case we need a bit more complicated coordinate mapping then a simple reduction to a uniform coordinate space, because we might map between populations that are layed out in different coordinate systems (e.g. retinal and cortical).

4 Projections

A `projection` tag holds a description of the connectivity between two groups of cells. Along with a user-given ID (`name`) unique inside the hosting group, each projection description contains `source` and `target` elements, each of which can reference either individual cell or a set. In the latter case each `projection` tag leads to creation of a set of projections rather than a single projection.

Projection description also includes references to a plasticity node (`plasticity`) that controls the synaptic weight and post-synaptic response node (`response`) that controls the influence of the input through this projection on the post-synaptic cell dynamics. Both appear in the description of the projection rather than neuron because the same type of neuron can use different synapses when instantiated in different populations, similarly the same postsynaptic response can be used in multiple projections with different plasticity rules. Instead of providing the user with all possible pre-wired combinations, NineML allows user to combine the plasticity→response→neuron chain from a set of small standard components.

Finally, a connectivity pattern or rule shall be set for a projection through reference to a connectivity node. This is done with tag `rule`.

Example description of the projection

```
<projection name="Inhibitory">
  <source><reference>Inhibitory cells</reference></source>
  <target><reference>Input receivers</reference></target>
  <response><reference>PSR</reference></response>
  <plasticity><reference>Plastic</reference></plasticity>
  <rule><reference>random fan-in</reference></rule>
</projection>
```

5 Structure of a User Layer Description

In order to simplify the descriptions themselves and the mechanisms for combining multiple components of the model the user layer description is consisting of multiple files that define various components and contains the syntax to import external files. The proposed syntax is

```
<import><url>...</url></import>
```

The URL in the above statement can be a reference to a local file

```
<url>file://home/someuser/somedirectory/someneurons.xml</url>
```

as well as a reference to some Internet database of models

```
<url>http://www.modelDB.org/somepath/someneurons.xml</url>
```

Issue: Different technical approaches were proposed in Antwerp, such as using XPath or Xinclude, but we decided that these technical issues should be investigated before the next meeting. They have not been investigated yet.

5.1 Scope of User Layer Descriptions

In order to avoid the collisions of user-given IDs as well as to provide the user with the ability to redefine some of the imported components the following set of rules is suggested:

- Every identifier is considered unique only within its parent
- To access the identifier a full path from the point of access to this identifier shall be provided, e.g. accessing a threshold parameter of a pyramidal cell in a cortical layer II from a top level network that includes all six layers of cortex will look similar to

  ```
  component[@name="layerII"].component[@name="pyramidal"].reference.threshold
  ```

  Here the first level accesses the group "layer II", the second level accesses the group "pyramidal" within the "layer II" group, the third level accesses the reference to a cell description and the last level selects threshold from this description. The exact syntax remains to be defined.
- Any imported construct can be redefined within the importing description by adding a new definition of the same object. For example if the file `pyramidal.xml` contains the description of a pyramidal cell named `''Pyramidal Cell''` that the user wants to reuse with a different parameter value the following syntax shall be used:

  ```
  <import><url>http://www.modelDB.org/somepath/pyramidal.xml</url></import>
  <node name="Pyramidal Cell">
    <threshold>
      <value>-35</value>
    </threshold>
  </node>
  ```

  In this example all properties of the cell are imported from external file and the value of the threshold is redefined in the local description. Note that the above example is only valid if a node named `''Pyramidal Cell''` was described in the imported file.
- If multiple redefinitions of the same entity are found, then the most recent one will be used. The most recent is determined by the order of parsing of the top level model description file. Here the rule shall be enforced that when multiple files are included all nodes are parsed before all groups. This will allow a smooth transition from networks with simple nodes and groups to networks where some nodes and some groups are redefined. A good example is a transition from Brunel's model A to model B in the appendix.
- If several different subfields are redefined, then all of these new definitions will be used. For example if in addition to the above code the user will provide

  ```
  <node name="Pyramidal Cell">
    <threshold>
      <unit>mV</unit>
    </threshold>
  </node>
  ```

  then both the value and the unit of the threshold will be set to follow local description.

Please note that having the code above is not the optimal way, and the better way to achieve the same effect is to provide all redefinitions in a unique location:

```
<import><url>http://www.modelDB.org/somepath/pyramidal.xml</url></import>
<node name="Pyramidal Cell">
  <threshold>
    <value>-35</value>
    <unit>mV</unit>
  </threshold>
</node>
```

The compliant simulation software shall understand and interpret both descriptions identically. Simulation software developers can (but at this point of time are not obliged to) optimize the NineML code that is output by their software by grouping all redefinitions of components of the same node into a single node.

5.2 Replacing Components of a User Layer Description

Issue: Add here discussion with Hugo about renaming/changing model/prototype without the necessity to modify analysis scripts.

6 Appendix: Specific Models Descriptions

In a first attempt, it appears that the more interesting approach is to start jointly the implementation of the user layer and the abstraction layer based on reference models. The user layer will be benchmarked on different models. In parallel, the abstraction layer mapping of these networks should be designed. Current version of this document only provides user layer descriptions.

6.1 [Brunel, 2000] Model A Description

The parameters used below Brunel gives in his Sec. 6, p. 196, which are the only simulations for which he shows spiking output.

So far there are several bits, we need to figure out how to frame them in a set of complete files.

A set of nodes descriptions (proposed file name `BrunelNodesA.xml`, this file is imported by following files):

```
<?xml version='1.0' encoding='UTF-8'?>
<nineml xmlns="http://www.NineML.org/9ML/1.0" name="Brunel (2001) Model A Nodes">
  <node name="E">
    <definition type="NineML">
      http://www.NineML.org/neurons/IaF_tau.xml
    </definition>
    <properties>
      <membraneConstant>
        <value>20</value>
        <unit>ms</unit>
      </membraneConstant>
      <threshold>
        <value>20</value>
        <unit>mV</unit>
      </threshold>
      <refractoryTime>
        <value>2</value>
        <unit>ms</unit>
      </refractoryTime>
      <resetPotential>
        <value>10</value>
        <unit>mV</unit>
      </resetPotential>
    </properties>
  </node>
  <node name="I">
    <reference>E</reference>
  </node>
  <node name="E_ext">
    <definition type="NineML">
      http://www.NineML.org/neurons/Poisson.xml
    </definition>
    <properties>
      <rate>
```

```
        <value>unclear</value>
        <unit>Hz</unit>
      </rate>
    </properties>
  </node>
  <node name="Excitatory">
    <definition type="NineML">
      http://www.NineML.org/weights/Static.xml
    </definition>
    <properties>
      <weight>
        <value>0.2</value>
        <unit>dimensionless</unit>
      </weight>
    </properties>
  </node>
  <node name="Inhibitory">
    <reference>Excitatory</reference>
    <properties>
      <weight>
        <value>node@name="Excitatory".weight.value*(-1)</value>
        <unit>dimensionless</unit>
      </weight>
    </properties>
  </node>
  <node name="Delta PSR">
    <definition type="NineML">
      http://www.NineML.org/synapses/Delta.xml
    </definition>
  </node>
  <node name="excitation">
    <definition type="NineML">
      http://www.NineML.org/connections/NBRP.xml
    </definition>
    <properties>
      <number>
        <value>1000</value>
      </number>
      <direction>convergent</direction>
      <delay>
        <value>2</value>
        <unit>ms</unit>
      </delay>
    </properties>
  </node>
  <node name="inhibition">
    <reference>excitation</reference>
    <number>
      <value>250</value>
    </number>
  </node>
  <node name="unstructured">
    <definition type="NineML">
      http://www.NineML.org/layouts/Unstructured.xml
    </definition>
    <properties/>
  </node>
</nineml>
```

A network description imports file `BrunelNodesA.xml`:

```
<?xml version='1.0' encoding='UTF-8'?>
<nineml xmlns="http://www.NineML.org/9ML/1.0" name="Brunel (2001) Model A Network">
<import><url>file://./BrunelNodesA.xml</url></import>
<group name="Network">
  <population name="Exc">
    <prototype><reference>E</reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>10000</number>
  </population>
  <population name="Inh">
    <prototype><reference>I</reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>2500</number>
  </population>
  <population name="Ext">
    <prototype><reference>E_ext</reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>population[@name="Exc"].number+population[@name="Inh"].number</number>
  </population>
  <set name="All neurons">
    <select>
      <any>
        <equal>
          <value>population[@name]</value>
          <value>Exc</value>
        </equal>
        <equal>
          <value>population[@name]</value>
          <value>Inh</value>
        </equal>
      </any>
    </select>
  </set>
  <projection name="External" >
    <source><reference>E_ext</reference></source>
    <target><reference>All neurons</reference></target>
    <plasticity><reference>Excitatory</reference></plasticity>
    <response><reference>Delta PSR</reference></response>
    <rule><reference>input</reference></rule>
  </projection>
  <projection name="Excitation" >
    <source><reference>Exc</reference></source>
    <target><reference>All neurons</reference></target>
    <plasticity><reference>Excitatory</reference></plasticity>
    <response><reference>Delta PSR</reference></response>
    <rule><reference>excitation</reference></rule>
  </projection>
  <projection name="Inhibition" >
    <source><reference>Inh</reference></source>
    <target><reference>All Neurons</reference></target>
    <plasticity><reference>Inhibitory</reference></plasticity>
    <response><reference>Delta PSR</reference></response>
    <rule><reference>inhibition</reference></rule>
  </projection>
</group>
</nineml>
```

This network is saved as `BrunelNetworkA.xml` for reuse in model B.

6.2 [Brunel, 2000] Model B Description

This is an example of extending the previous model. Note that the first cell type overrides the description of the interneuron in Model A, the second adds a new cell type. A set of nodes descriptions (proposed file name `BrunelNodesB.xml`, this file imports `BrunelNodesA.xml`):

```
<?xml version='1.0' encoding='UTF-8'?>
<nineml xmlns="http://www.NineML.org/9ML/1.0" name="Brunel (2001) Model B Nodes">
<import><url>file://./BrunelNodesA.xml</url></import>
<nodes>
  <node name="I">
    <reference>E</reference>
    <properties>
      <membraneConstant>
        <value>unclear</value>
      </membraneConstant>
    </properties>
  </node>
  <node name="E_ext_I">
    <definition type="NineML">
      http://www.NineML.org/neurons/Poisson.xml
    </definition>
    <properties>
      <rate>
        <value>unclear</value>
        <unit>Hz</unit>
      </rate>
    </properties>
  </node>
</nodes>
</nineml>
```

A network description imports file `BrunelNodesB.xml` and file `BrunelNetworkA.xml`. This import leads to the file `BrunelNodesA.xml` being imported twice through recursive inputs from both files. The scoping rules of section 5.1 enforce that all nodes are parsed first, so the updated neuron descriptions are used even for creating groups defined in `BrunelNetworkA.xml`.

```
<?xml version='1.0' encoding='UTF-8'?>
<nineml xmlns="http://www.NineML.org/9ML/1.0" name="Brunel (2001) Model B Network">
<import><url>file://./BrunelNodesB.xml</url></import>
<import><url>file://./BrunelNetworkA.xml</url></import>
<group name="Network">
  <population name="Ext">
    <number>population[@name="Exc"].number</number>
  </population>
  <population name="Ext I">
    <prototype><reference>E_ext_I</reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>population[@name="Inh"].number</number>
  </population>
  <projection name="External" >
    <target><reference>Exc</reference></target>
  </projection>
  <projection name="External I" >
    <source><reference>E_ext_I</reference></source>
    <target><reference>Inh</reference></target>
    <plasticity><reference>Excitatory</reference></plasticity>
    <response><reference>Delta PSR</reference></response>
    <rule><reference>input</reference></rule>
  </projection>
</group>
```

```
</nineml>
```

This network reuses components `Exc` and `Inh` as defined in `BrunelNetworkA.xml` but with new description of an interneuron `I` done in `BrunelNodesB.xml`. It also reuses projections `Excitation` and `Inhibition` from `BrunelNetworkA.xml`. This network redefines the `number` for component `Ext` and the target for projection `External`. Finally it adds descriptions of component `Ext I` and projection `External I`.

Issue: This is awesomely compact but barely readable.

6.3 [Morrison et al., 2007] Model Description

A set of nodes descriptions (proposed file name `MorrisonNodes.xml`, this file imports `BrunelNodesA.xml` and redefines some parameters):

```
<?xml version='1.0' encoding='UTF-8'?>
<nineml xmlns="http://www.NineML.org/9ML/1.0" name="Morrison et al (2007) Nodes">
<import><url>file://./BrunelNodesA.xml</url></import>
<nodes>
  <node name="E">
    <properties>
      <membraneConstant>
        <value>10</value>
      </membraneConstant>
      <membraneCapacitance>
        <value>250</value>
        <unit>pF</unit>
      </membraneCapacitance>
      <refractoryTime>
        <value>0.5</value>
      </refractoryTime>
      <resetPotential>
        <value>0</value>
      </resetPotential>
    </properties>
  </node>
  <node name="I">
    <reference>E</reference>
  </node>
  <node name="E_ext">
    <properties>
      <rate>
        <value>20880</value>
      </rate>
    </properties>
  </node>
  <node name="Excitatory">
    <definition type="NineML">
      http://www.NineML.org/weights/Static.xml
    </definition>
    <properties>
      <weight>
        <value>45.61</value>
        <unit>dimensionless</unit>
      </weight>
    </properties>
  </node>
  <node name="Inhibitory">
    <reference>Excitatory</reference>
    <weight>
```

```
    <value>node@name="Excitatory".weight.value*(-5)</value>
  </weight>
</node>
<node name="Plastic">
  <definition type="NineML">
    http://www.NineML.org/synapses/PowerAA_STDP.xml</definition>
  <properties>
    <potTimeConst>
      <value>20</value>
      <unit>ms</unit>
    </potTimeConst>
    <depTimeConst>
      <value>20</value>
      <unit>ms</unit>
    </depTimeConst>
    <power>
      <value>0.4</value>
      <unit>dimensionless</unit>
    </power>
    <learningRate>
      <value>0.1</value>
      <unit>dimensionless</unit>
    </learningRate>
    <asymmetry>
      <value>0.11</value>
      <unit>dimensionless</unit>
    </asymmetry>
  </properties>
</node>
<node name="PSC">
  <definition type="NineML">
    http://www.NineML.org/synapses/AlphaI.xml
  </definition>
  <properties>
    <timeConstant>
      <value>0.33</value>
      <unit>ms</unit>
    </timeConstant>
  </properties>
</node>
<node name="input">
  <definition type="NineML">
    http://www.NineML.org/connections/One-to-one.xml
  </definition>
  <properties>
    <delay>
      <value>unclear</value>
      <unit>ms</unit>
    </delay>
  </properties>
</node>
<node name="excitation">
  <definition type="NineML">
    http://www.NineML.org/connections/NBRP.xml
  </definition>
  <properties>
    <number>
      <value>9000</value>
    </number>
```

```
      <direction>convergent</direction>
      <delay>
        <value>unclear</value>
        <unit>ms</unit>
      </delay>
    </properties>
  </node>
  <node name="inhibition">
    <reference>excitation</reference>
    <number>
      <value>2250</value>
    </number>
  </node>
</nodes>
</nineml>
```

A network description imports file `MorrisonNodes.xml`:

```
<?xml version='1.0' encoding='UTF-8'?>
<nineml xmlns="http://www.NineML.org/9ML/1.0" name="Morrison et al (2007) Network">
<import><url>file://./MorrisonNodes.xml</url></import>
<group name="Network">
  <population name="Exc">
    <prototype><reference>E</reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>90000</number>
  </population>
  <population name="Inh">
    <prototype><reference>I</reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>22500</number>
  </population>
  <population name="Ext">
    <prototype><reference>E_ext</reference></prototype>
    <layout><reference>unstructured</reference></layout>
    <number>population[@name="Exc"].number+population[@name="Inh"].number</number>
  </population>
  <set name="All neurons">
    <select>
      <any>
        <equal>
          <value>population[@name]</value>
          <value>Exc</value>
        </equal>
        <equal>
          <value>population[@name]</value>
          <value>Inh</value>
        </equal>
      </any>
    </select>
  </set>
  <projection name="External" >
    <source><reference>E_ext</reference></source>
    <target><reference>All neurons</reference></target>
    <plasticity><reference>Excitatory</reference></plasticity>
    <response><reference>PSC</reference></response>
    <rule><reference>input</reference></rule>
  </projection>
  <projection name="Exc-to-Inh" >
    <source><reference>Exc</reference></source>
```

```
    <target><reference>Inh</reference></target>
    <plasticity><reference>Excitatory</reference></plasticity>
    <response><reference>PSC</reference></response>
    <rule><reference>excitation</reference></rule>
  </projection>
  <projection name="Inhibitory" >
    <source><reference>Inh</reference></source>
    <target><reference>All Neurons</reference></target>
    <plasticity><reference>Inhibitory</reference></plasticity>
    <response><reference>PSC</reference></response>
    <rule><reference>inhibition</reference></rule>
  </projection>
  <projection name="Exc-to-Exc" >
    <source><reference>Exc</reference></source>
    <target><reference>Exc</reference></target>
    <plasticity><reference>Plastic</reference></plasticity>
    <response><reference>PSC</reference></response>
    <rule><reference>excitation</reference></rule>
  </projection>
</group>
</nineml>
```

6.4 [Vogels and Abbott, 2005] Model Description

This comes from our excercise in Stockholm and is only partially complete. A set of nodes descriptions (proposed file name `VogelsAbbotNodes.xml`. Note that here we used `onset` and `duration` inside `InputNeuron` description to mark the input protocol.

```
<?xml version='1.0' encoding='UTF-8'?>
<nineml xmlns="http://www.NineML.org/9ML/1.0" name="Vogels and Abbott (2005) Nodes">
  <node name="InputNeuron">
    <definition type="NineML">
      http://www.NineML.org/1.0/neurons/Poisson.xml
    </definition>
    <properties>
      <onset>
        <value>0.0</value>
        <unit>ms</unit>
      </onset>
      <duration>
        <value>50.0</value>
        <unit>ms</unit>
      </duration>
      <rate>
        <value>100.0</value>
        <unit>Hz</unit>
      </rate>
    </properties>
  </node>
  <node name="Neuron">
    <definition type="NineML">
      http://www.NineML.org/1.0/neurons/LeakyIAF.xml</definition>
    <properties>
      <voltage>
        <value>-65</value>
        <unit>mV</unit>
      </voltage>
      <membraneCapacitance>
        <value>0.2</value>
```

```
      <unit>nF</unit>
    </membraneCapacitance>
    <membraneTimeConstant>
      <value>20.0</value>
      <unit>ms</unit>
    </membraneTimeConstant>
    <refractoryTime>
      <value>5.0</value>
      <unit>ms</unit>
    </refractoryTime>
    <threshold>
      <value>-50.0</value>
      <unit>mV</unit>
    </threshold>
    <restingPotential>
      <value>-60.0</value>
      <unit>mV</unit>
    </restingPotential>
    <resetPotential>
      <value>-60.0</value>
      <unit>mV</unit>
    </resetPotential>
  </properties>
</node>
<node name="ExcPSR">
  <definition type="NineML">
    http://www.NineML.org/1.0/plasticitys/ExponentialG.xml
  </definition>
  <properties>
    <time>
      <value>0</value>
      <unit>ms</unit>
    </time>
    <conductance>
      <value>0</value>
      <unit>nS</unit>
    </conductance>
    <timeConstant>
      <value>5.0</value>
      <unit>ms</unit>
    </timeConstant>
    <reversalPotential>
      <value>0.0</value>
      <unit>mV</unit>
    </reversalPotential>
  </properties>
</node>
<node name="InhPSR">
  <reference>
    ExcPSR
  </reference>
  <properties>
    <timeConstant>
      <value>10.0</value>
      <unit>ms</unit>
    </timeConstant>
    <reversalPotential>
      <value>-80.0</value>
      <unit>mV</unit>
```

```
    </reversalPotential>
  </properties>
</node>
<node name="ExcProj">
  <definition type="NineML">
    http://www.NineML.org/1.0/weights/Static.xml
  </definition>
  <properties>
    <delay>
      <value>0.2</value>
      <unit>ms</unit>
    </delay>
    <weight>
      <value>4</value>
      <unit>dimensionless</unit>
    </weight>
  </properties>
</node>
<node name="InhProj">
  <definition type="NineML">
    http://www.NineML.org/1.0/weights/Static.xml
  </definition>
  <properties>
    <delay>
      <value>0.2</value>
      <unit>ms</unit>
    </delay>
    <weight>
      <value>51</value>
      <unit>dimensionless</unit>
    </weight>
  </properties>
</node>
<node name="InputProj">
  <definition type="NineML">
    http://www.NineML.org/1.0/weights/Static.xml
  </definition>
  <properties>
    <delay>
      <value>0.2</value>
      <unit>ms</unit>
    </delay>
    <weight>
      <value>100</value>
      <unit>dimensionless</unit>
    </weight>
  </properties>
</node>
<node name="connectRule">
  <definition type="NineML">
    http://www.NineML.org/1.0/connections/FixedProbability.xml
  </definition>
  <properties>
    <allow_self_connections>
      <value>True</value>
    </allow_self_connections>
    <p_connect>
      <value>0.02</value>
      <unit>dimensionless</unit>
```

```
      </p_connect>
    </properties>
  </node>
  <node name="inpConnectRule">
    <definition type="NineML">
      http://www.NineML.org/1.0/connections/FixedProbability.xml
    </definition>
    <properties>
      <allow_self_connections>
        <value>True</value>
      </allow_self_connections>
      <p_connect>
        <value>0.01</value>
        <unit>dimensionless</unit>
      </p_connect>
    </properties>
  </node>
  <node name="CommonLayout">
    <definition type="NineML">
      http://www.NineML.org/1.0/layouts/Unstructured.xml
    </definition>
    <properties/>
  </node>
</nineml>
```

A network description imports file `VogelsAbbotNodes.xml`:

```
<?xml version='1.0' encoding='UTF-8'?>
<nineml xmlns="http://www.NineML.org/9ML/1.0" name="Vogels and Abbott (2005) Network">
  <import><url>file://./VogelsAbbotNodes.xml</url></import>
  <group name="Network">
    <population name="ExcitatoryPop">
      <number>3200</number>
      <prototype><reference>Neuron</reference></prototype>
      <layout><reference>CommonLayout</reference></layout>
    </population>
    <population name="InhibitoryPop">
      <number>800</number>
      <prototype><reference>Neuron</reference></prototype>
      <layout><reference>CommonLayout</reference></layout>
    </population>
    <population name="InputPop">
      <number>20</number>
      <prototype><reference>InputNeuron</reference></prototype>
      <layout><reference>CommonLayout</reference></layout>
    </population>
    <set name="AllNeurons">
      <select>population[@name="ExcitatoryPop"]|population[@name="InhibitoryPop"]</select>
    <set>
    <projection name="InhibitoryPop-AllNeurons">
      <source><reference>InhibitoryPop</reference></source>
      <target><reference>AllNeurons</reference></target>
      <rule><reference>connectRule</reference></rule>
      <response><reference>InhPSR</reference></response>
      <plasticity><reference>InhProj</reference></plasticity>
    </projection>
    <projection name="InputPop-AllNeurons">
      <source><reference>InputPop</reference></source>
      <target><reference>AllNeurons</reference></target>
      <rule><reference>inpConnectRule</reference></rule>
```

```
      <response><reference>ExcPSR</reference></response>
      <plasticity><reference>InputProj</reference></plasticity>
    </projection>
    <projection name="ExcitatoryPop-AllNeurons">
      <source><reference>ExcitatoryPop</reference></source>
      <target><reference>AllNeurons</reference></target>
      <rule><reference>connectRule</reference></rule>
      <response><reference>ExcPSR</reference></response>
      <plasticity><reference>ExcProj</reference></plasticity>
    </projection>
  </group>
</nineml>
```

7 Appendix: Formal Grammar of User Layer Descriptions

In Antwerp we decided to add a systematic definition of user layer tags here. To be filled.

References

[Brunel, 2000] Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *Journal of Computational Neuroscience*, 8:183–208.

[Morrison et al., 2007] Morrison, A., Aertsen, A., and Diesmann, M. (2007). Spike timing dependent plasticity in balanced random networks. *Neural Computation*, 19:1437–1467.

[Vogels and Abbott, 2005] Vogels, T. P. and Abbott, L. F. (2005). Signal propagation and logic gating in networks of integrate-and-fire neurons. *Journal of Neuroscience*, 25:10786–10795.