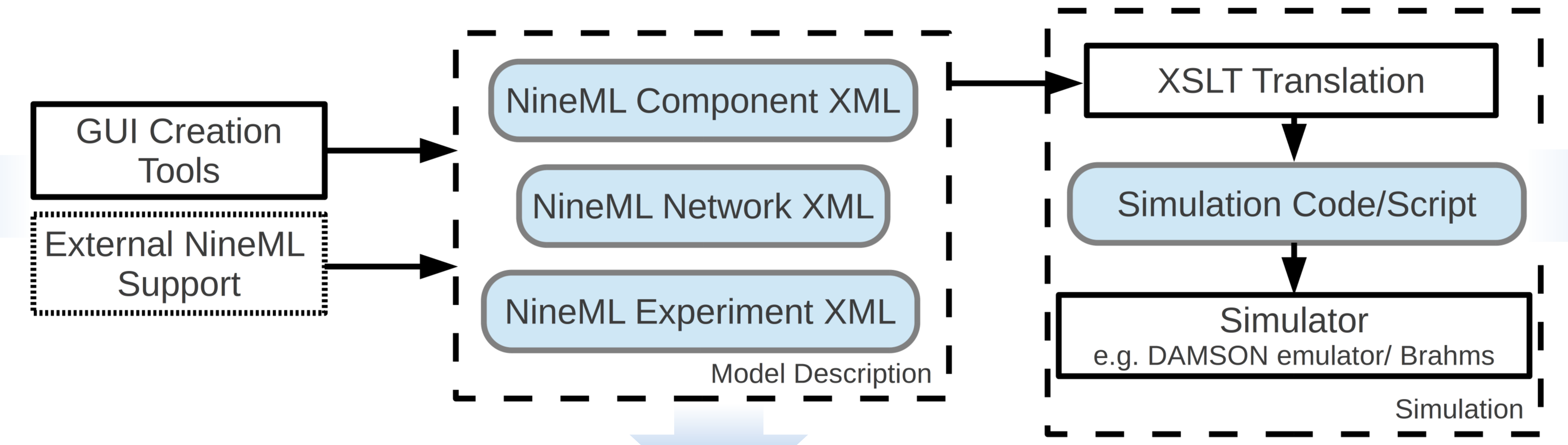
A toolchain for creation of spiking neural networks utilising NineML - a tool independent XML model description format

Cope A.J. and Richmond P. – University of Sheffield (contact a.cope@sheffield.ac.uk)

Introduction

There is a need for flexible, simulator independent methods of describing neural models. While NeuroML seeks to provide a comprehensive description format its support for Spiking Neural Networks (SNNs) is still in development. We therefore present a toolchain based around the NineML [4] specification (which we have modified and completed – allowing code generation from the model description). The toolchain we present supports the possibility of multiple creation tools, here we present a GUI as an example, and code generation for multiple simulators and hardware platforms – all based around a single independent model description format.



References:

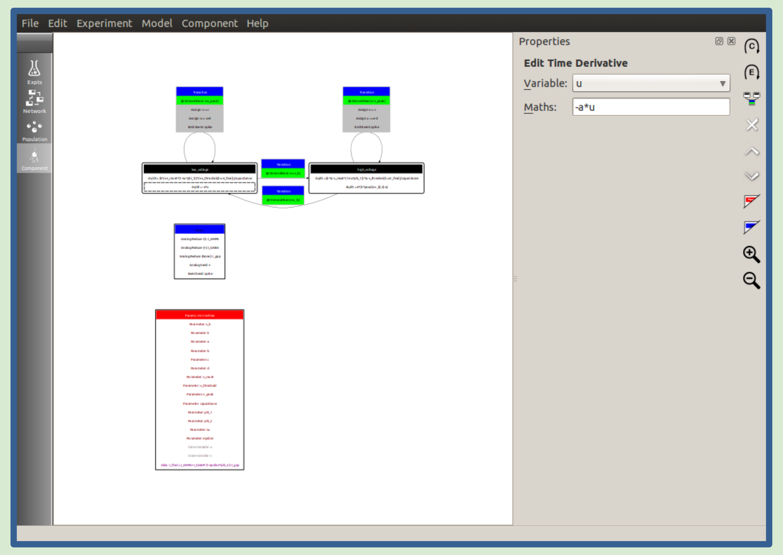
1. BRAHMS: Mitchinson B, Chan T-S, Chambers J, Pearson M, Humphries M, Fox C, Gurney K, Prescott TJ, **BRAHMS: Novel middleware for integrated systems computation**, ***Advanced Engineering Informatics***, 24(1):49-61, January 2010.
2. PyNN: Davison AP, Brüderle D, Eppler JM, Kremkow J, Muller E, Pecevski DA, Perrinet L and Yger P (2008) PyNN: a common interface for neuronal network simulators. Front. Neuroinform. doi:10.3389/neuro.11.011.2008
3. Benchmark: Brette et al (2007) Simulation of networks of spiking neurons: A review of tools and strategies. J. Comp. Neuro. doi:10.1007/s10827-007-0038-6
4. NineML: http://software.incf.org/software/nineml/wiki/introduction-to-nineml

The University Of Sheffield.

Example Creation Tools - Graphical User Interface

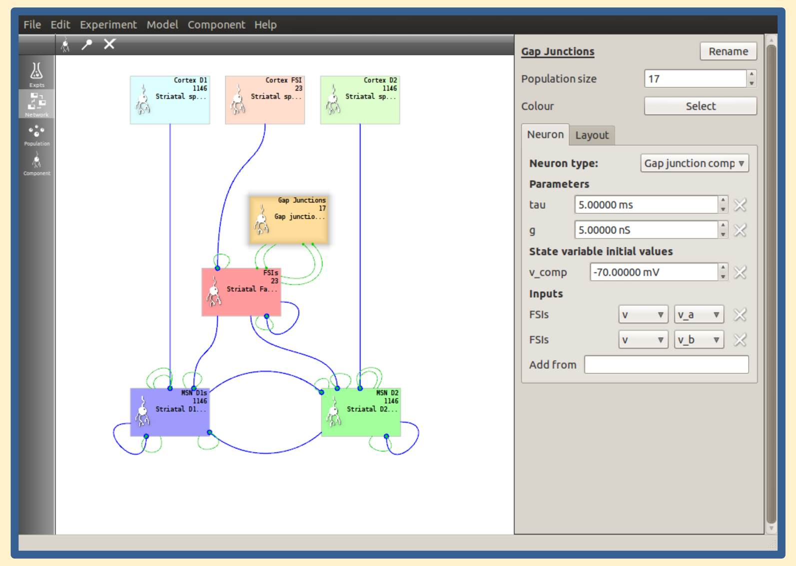
Component creation

- Graphviz automatic layout
- Direct mapping to underlying 9ML
- Point and click operation – no programming required!
- Diagrams can be exported in standard format for inclusion in publications (.eps, .jpg, .png)
- Save to 9ML
- Cross platform (developed using Qt tools) – Windows / Linux / OSX



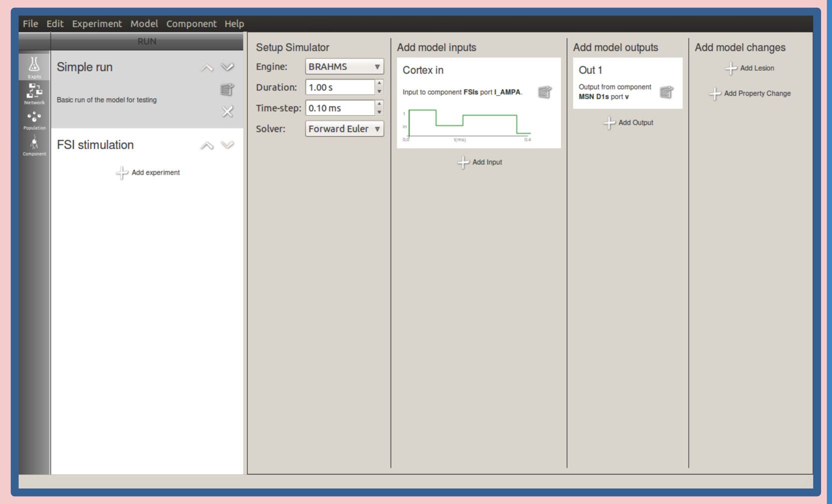
Network creation

- Point and click operation
- Direct mapping to 9ML Network Layer
- Easy layout (drag-drop, snap to grid)
- Colours for easy identification and grouping
- Context sensitive parameterisation
- Import connectivity / parameter data from comma separated files

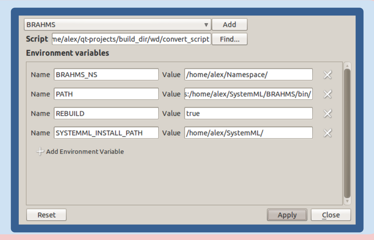


Experiments

- Provides an interface for the organisation of multiple experimental paradigms
- Utilises data from current model to ease experiment specification
- Future work will allow the automatic graphing of data from the model ouputs

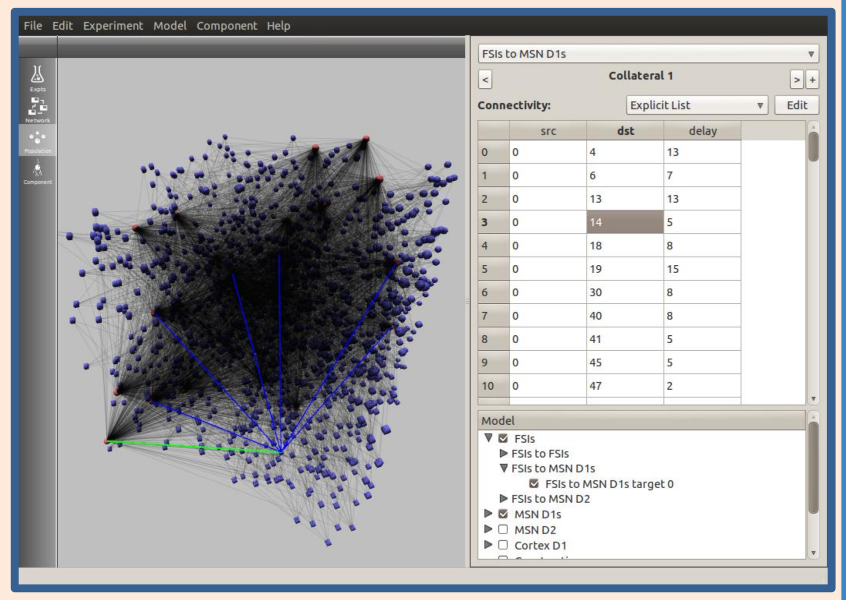


- User extensible support for simulators



Visualisation

- Powerful real time OpenGL visualisation of populations and connectivity
- User extensible 3D layout engine for specifying neuron positions procedurally
- Context sensitive connection highlighting in real time



Model Description Format – NineML (9ML)

Component layer

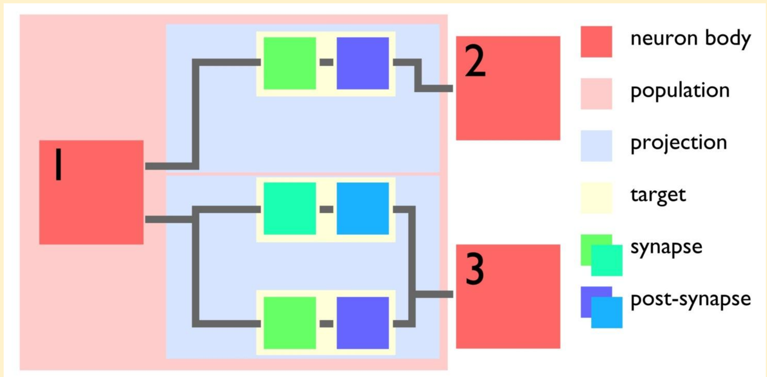
- Describes reusable cell level "components" which provide building blocks for a network description
- Uses Regimes, OnConditions and OnEvents to achieve state like dynamics based upon the existing 9ML format
- Ports (Analogue and Event) are used as input and output channels for communication

```xml
<NineML
  xmlns="http://www.shef.ac.uk/NineMLComponentLayer">
 <ComponentClass type="neuron_body" name="IF">
  <Dynamics initial_regime="integrating">
   <Regime name="integrating">
    <TimeDerivative variable="v">
     <MathInline>((i_offset + I_Syn) /cm)+(v_rest-v)/tau_m</MathInline>
    </TimeDerivative>
    <OnCondition target_regime="refractory">
     <StateAssignment variable="v"><MathInline>v_reset</MathInline></StateAssignment>
     <StateAssignment variable="t_spike"><MathInline>t</MathInline></StateAssignment>
     <EventOut port="spike"/>
     <Trigger><MathInline>v &gt; v_thresh</MathInline></Trigger>
    </OnCondition>
   </Regime>
   <Regime name="refractory">
    <OnCondition target_regime="integrating">
     <Trigger><MathInline>t &gt; (t_spike+tau_refractory)</MathInline></Trigger>
    </OnCondition>
   </Regime>
   <StateVariable name="v"/>
   <StateVariable name="t_spike"/>
  </Dynamics>
  <AnalogReducePort reduce_op="+" name="I_Syn"/>
  <EventSendPort name="spike"/>
  <Parameter dimension="pF" name="cm"/>
  <Parameter dimension="pA" name="i_offset"/>
  <Parameter dimension="mV" name="v_thresh"/>
  <Parameter dimension="mV" name="v_rest"/>
  <Parameter dimension="mV" name="v_reset"/>
  <Parameter dimension="ms" name="tau_m"/>
  <Parameter dimension="ms" name="tau_refractory"/>
 </ComponentClass>
</NineML>
```

```xml
<NineML
  xmlns="http://www.shef.ac.uk/NineMLNetworkLayer">
  <Population>
    <Neuron name="Excitatory" size="3200" url="Integrate_and_fire.xml">
      <Property name="v">
        <UniformDistribution minimum="-60" maximum="-50"/>
      </Property>
      <Property name="c_m">
        <FixedValue value="0.2"/>
      </Property>
      <Property name="v_threshold">
        <FixedValue value="-50"/>
      </Property>
      <Property name="v_rest">
        <FixedValue value="-49"/>
      </Property>
      <Property name="v_reset">
        <FixedValue value="-60"/>
      </Property>
      <Property name="tau_m">
        <FixedValue value="0.02"/>
      </Property>
      <Property name="tau_refractory">
        <FixedValue value="0.005"/>
      </Property>
    </Neuron>
    <Projection dst_population="Inhibitory">
      <Target>
      <FixedProbabilityConnection probability="0.02"
        seed="123">
        <Delay>
          <FixedValue value="0.1"/>
        </Delay>
      </FixedProbabilityConnection>
      <Synapse name="E_to_I_syn"
               url="../Componenets/fixed_weight.xml"
               input_src_port="spike"
               input_dst_port="spike">
      </Synapse>
      <PostSynapse name="E_to_I_psp"
                   url="../Componenets/Current_exp_synapse.xml"
                   input_src_port="PSP"
                   input_dst_port="PSP"
                   output_src_port="I_Syn"
                   output_dst_port="I_Syn">
      </PostSynapse>
    </Target>
    </Projection>
  </Population>
</NineML>
```

Network layer

- Creates parameterised instances (i.e. Populations or Groups) of components
- Multiple schemas allow different levels of simulator support
- High Level Schema specifies a network using Populations and Projections (with Synapse and PSP components) to describe connectivity
  - Connectivity described using an extendible set of Connectivity types
  - Re-mappings between Synapse and PSP automatically assumed
- Low Level Schema allows arbitrary Groups of component instances
  - Connectivity described using generic input connections
  - Can be used as inputs within Population components (including Synapse and PSP)
  - Extra flexibility allows modelling of larger range of neural phenomena e.g. gap junctions



XML Schema

- Syntactical validation of model specification including cross referencing of named items
  - Ensures well formatted model for code generation
- Uses a hybrid "Flat" and "Complex Type" schema design to ensure extendibility (i.e. new connectivity types can easily be added)

Experiment layer

- integration methods
- Describes "Lesions" to Projections and any parameter or state variable changes to Populations or Groups
- Defines Inputs to the network via component Ports
- Defines Outputs of the model for logging

```xml
<NineML
  xmlns="http://www.shef.ac.uk/NineMLExperimentLayer">
  <Experiment name="Benchmark">
    <Model network_layer_url="model.xml" />
    <Simulation duration="1" preferred_simulator="Brahms">
      <EulerIntegration dt="0.1" />
    </Simulation>
    <LogOutput name="Ispike" target="Inhibitory" port="spike"/>
    <LogOutput name="Espike" target="Excitatory" port="spike"/>
    <LogOutput name="VI" target="Inhibitory" port="V"/>
    <LogOutput name="VE" target="Excitatory" port="V"/>
  </Experiment>
</NineML>
```

Simulator Support

Models/Experiments mapped to simulators using XSLT transformation templates
Requires no additional library or software dependencies (XSLT processing governed by W3C)

Reference simulator (using BRAHMS middleware [1])

- Designed around the use of reusable components
- Supports full specification (all high and low level functionality)
- Supports parallelisation across cores with a machine, and across machines in a network
- Built in logging and profiling
- Minimal overhead

PyNN [2]

- Supports only High Level Network layer descriptions
- Supports only a fixed set of PyNN compatible component types (which match PyNN standardised neuron models)
- Supports current inputs via external current ports in components

Neuromorphic Hardware Support

- Uses the DAMSON event driven C-like language to provide a mapping through to a functional SpiNNaker emulator (with hardware support through code generation)
- Only fixed point numerical representation
- Requires an additional Network layer Splitter to allow multi-core simulation
  - Constrains sizes of Groups and Populations
  - Splits properties of parameters and state variables
  - Splits Projections and Inputs
  - Splitting beneficial to other multi-core simulators (including Reference Simulator)

Results

Consistent results obtained from mapping a standard benchmark model [3] through to three supported simulators.
Results are consistent with model specified directly in PyNN