An Introduction to NineML

Mike Hull <m.j.hull@sms.ed.ac.uk>

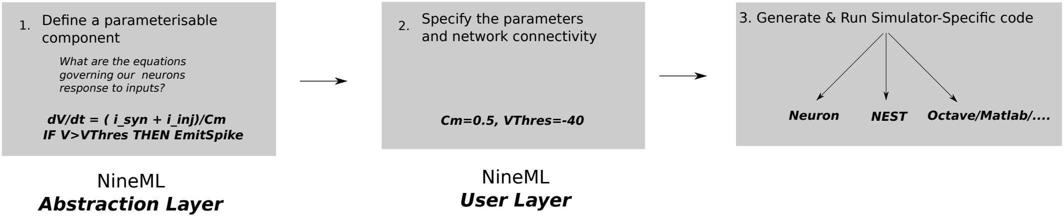
Overview

Introduction

- 9ML is a set of concepts underpinning a language for multiscale modelling in neuroscience
- The architecture, conceptual design & specification are the result of an INCF taskforce
- (Maps to an XML format)
- Multiple implementations of the concepts:
    - Python
    - Chicken Scheme
    - LEMS/NeuroML support/intersection
- Python lib9ml is a Python API to create and manipulate 9ML models

Layers

- 9ML is composed of 2 layers:
  - The Abstraction Layer
  - The User Layer



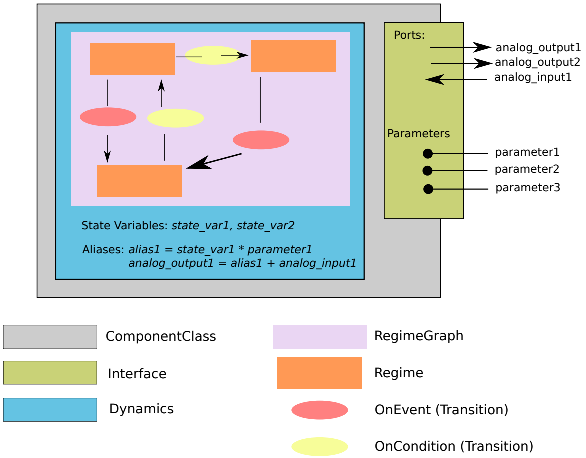
Object Model

Lots of Terms:

- Component
- Interface, Dynamics
- Regime, Transition
- StateVariable, TimeDerivative, StateAssignment
- OnEvent, OnCondition
- EventPort, AnalogPort, send/recv/reduce port
- Parameter
- Alias
- ...

Object Model Overview

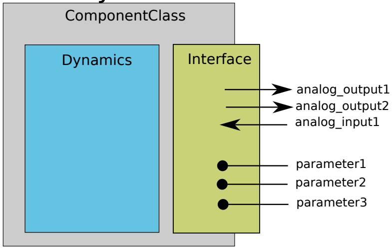
Upper Bound on Complexity...



ComponentClass

- A component represents some sort of dynamic object in a simulation
- A component could represent a channel, a neuron or a synapse

A 9ML component has *internal* dynamics and an *external* interface



Interface

"*how does the component communicate to the outside world?*"

The interface to a component is composed of

- Parameters - Set once at the start of a simulation (*compile-time*)
- Ports - Used to communication between components during a simulation (*run-time*)

Parameters

- Parameters allow us to define templated components in the abstraction layer
- This means we do not need to repeat the same code for every variation of the model
- For example, for an integrate-and-fire neuron: *Reset-Voltage* and *Firing-Threshold*
- Parameters are set at the start of a simulation and remain constant throughout
- [Parameter values are specified later by the User Layer]

Ports

- We are normally interested in not one object; but how a group of components respond when connected together
- Ports allow components to communicate between each other during a simulation
- There are 2 types:
    - AnalogPorts - Communicate continuous values
    - EventPorts - Communicate discrete events
- There are 3 port-modes; *incoming* (*recv/reduce*) or *outgoing* (*send*), i.e. receiving or transmitting information
- *outgoing* ports are connected to *incoming* ports

Event Ports

- EventPorts send discrete events at a particular instant in time.
- For example:
    - a *neuron* component could send an event when it fires an action-potential
    - then a synapse *component* could cause a post-synaptic current in another neuron in response to receiving that event
- A *send* EventPort can be connected to any number of *recv* EventPorts
- Similarly, a *recv* EventPort can be connected to any number of *send* EventPorts

AnalogPorts

- AnalogPorts transmit continuous values
- For example:
  - A current-clamp component would have a *send* AnalogPort for the current it produces.
  - A neuron component could connect a *recv* AnalogPort for receiving this external current
- A *send* AnalogPort can connect to any number of *recv* AnalogPorts
- BUT a *recv* AnalogPort can only be connected to one *send* AnalogPort.

Reduce Ports

In many cases, we may want to connect an *incoming* port to several *send* ports.

For example, if we have neuron, we do not want to define a *recv* port on that component for each possible external input current.

Instead, we use *reduce* ports; which is similar to a *recv* port, except that it can be connected to multiple *send* ports and we define how the input from those should be combined.

We currently support, + for summing inputs.

This allows us to write decoupled components; since we are not interested in what or how many things we connect to; only that input-currents for example, can be summed.

Dynamics

"*how does the component component work internally??*"

The dynamics block defines the behaviour of the component; in response to the interface. A dynamics block is composed of:

- StateVariables
- A RegimeTransition graph composed of *Regimes* & *Transitions*

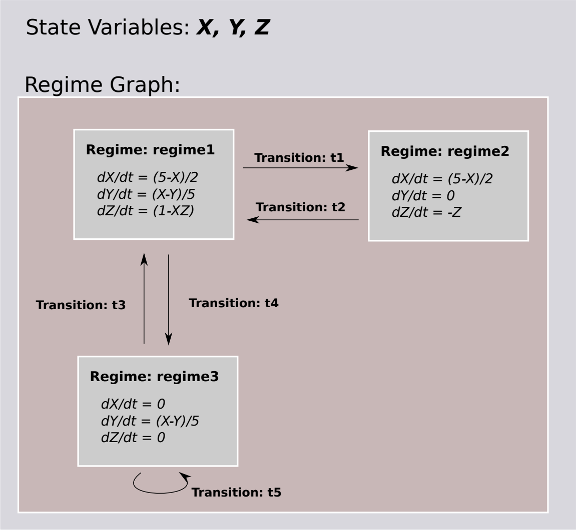
State Variables, Regimes & Transitions

- The dynamics of a component is defined by a set of state-variables;
- StateVariables can change either discontinuously or continuously as a function of time.
- The changes happen in two ways:
    - through TimeDerivatives, which define the continuous evolution over time. e.g. dg/dt = -g/gtau
    - through StateAssignments, which make discrete changes to a StateVariable’s value. e.g. g = g + 5

RegimeTransition Graph

- A component can have different TimeDerivatives for the StateVariables at different times.
- At any given time, a component will be in a certain Regime
- The TimeDerivatives controlling the StateVariables are specified by the current *Regime*.
- [The state-space of a neuron is defined by its StateVariables and its current Regime.]

RegimeTransition Graph



Transitions

Components move between Regimes via Transitions. There are 2 ways of triggering a Transition:

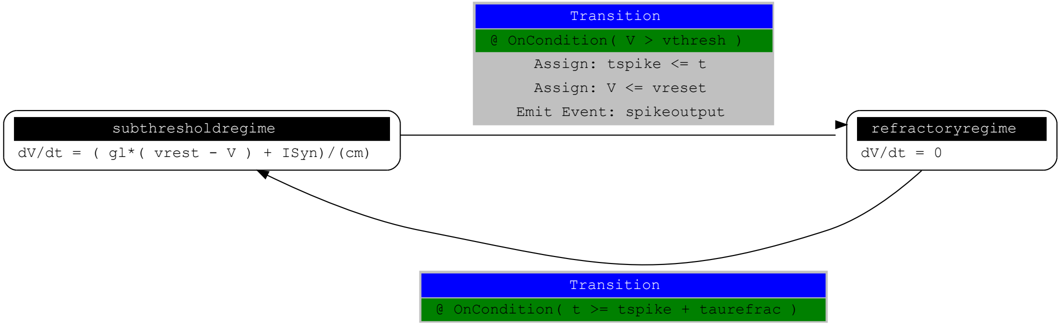
- By a condition of the state variables, for example *V > VThresh*.
- By an InputEvent on a port.

When a Transition is triggered; three things can happen:

- A change of Regime. e.g. if the component is in *regime3*, and the trigger for *t3* is satisfied, then the component will move into *regime1*
- StateAssignments can take place
- The component can send OutputEvents

During a transition, multiple StateAssignments and OutputEvents can occur.

RegimeTransition Graph for IAF Component

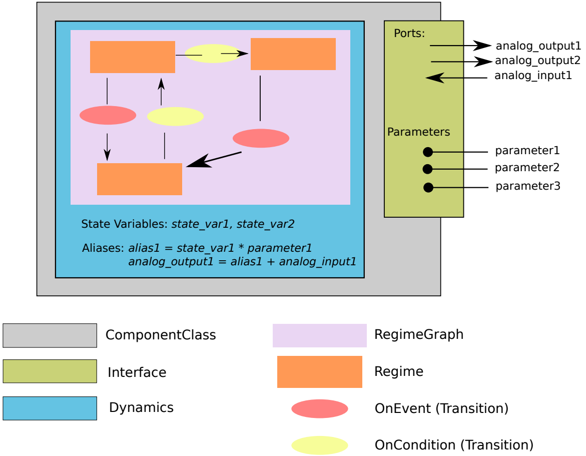


Aliases

- Aliases allow us to define local symbols in a ComponentClass:
  - Intermediate variables.
  - Reduce Duplication (E.g. Tau/Inf in Hodgekin-Huxley Channel).
  - Recording of more complex expressions.
  - Simplification of send AnalogPorts.
- Example:

```
minf = malpha / (malpha + mbeta)
mtau = 1.0 / (malpha + mbeta)
malpha = (A_alpha + B_alpha *V ) / (C_alpha + exp( (D_alpha +V)/E_alpha) )
mbeta = (A_beta + B_beta *V ) / (C_beta + exp( (D_beta +V)/E_beta) )
```

Object Model Recap



Python lib9ML

What can it do for you?

- Simulation & Tool developers ?

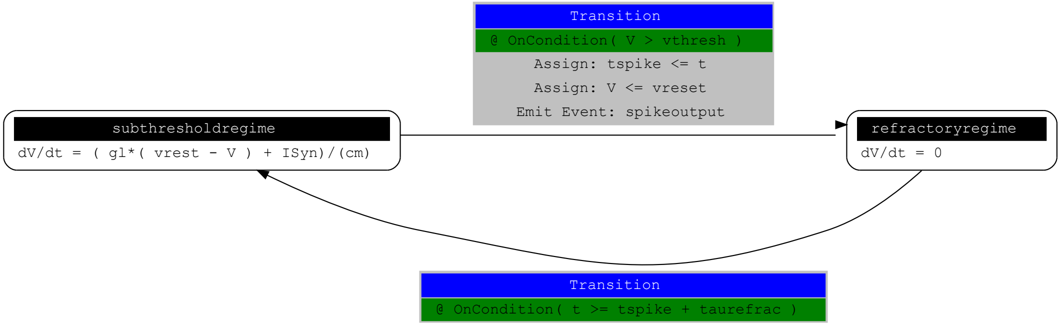
  Python lib9ml is an implementation of the 9ML Spec for:

  - Loading and saving models from/to XML to/from Python objects.
  - Object oriented API for traversing, querying and manipulating 9ML object model
  - A platform for code-generation
- Users/Modellers ?
  - Intuitive, human read-writable API for model creation using 9ML
  - NEST, NEURON and PyNN support (beta)
  - Early adoptors and testers welcome!

Python lib9ML Model Construction

- Python lib9ML provides a concise syntax for constructing models directly in python
- Automatic inference of StateVariables, Parameters and EventPorts
- Hierarchical components (Not standardised)
    - allow us to build a single component, out of several smaller components.
    - It allows us to separate their namespaces, producing simpler models.
    - This allows us to define subcomponents in a *reusable* way, (for example synapses)

Example Regime Transition Graph



Model Construction

```
import nineml.abstraction_layer as al
r1 = al.Regime(
               "dV/dt = ( gl*( vrest - V ) + ISyn)/(cm)",
               name = "subthresholdregime",
               transitions = al.On("V > vthresh",
                                   do=["tspike = t",
                                       "V = vreset",
                                       al.OutputEvent('spikeoutput')],
                                   to="refractoryregime"),)
r2 = al.Regime(
            "dV/dt = 0",
            name = "refractoryregime",
            transitions = [ al.On("t >= tspike + taurefrac",
                                  to="subthresholdregime") ], )

iaf = al.ComponentClass( name = "iaf",
                         regimes = [r1,r2],
                         analog_ports = [    al.SendPort("V"), al.ReducePort("ISyn", reduce_op="+")],
                        )
```

Manipulating the Object Model

```
# Writing/Reading the object-model to XML
XMLWriter( iaf, 'iaf.xml')
loaded_iaf = XMLReader('iaf.xml')

# Creating a dot file of the RegimeTransition Graph
DotWriter( iaf, 'iaf_rtg.dot')

# Run simulations using PyNN ...
...
```

Finally..

- Thanks for listening
- Any questions?