Example Neosim projections

March 16, 2000

1 Scope

This document includes examples of specifying projections.

2 Basic Projections - a vector of individual connections

A set of connections can be built from the script reader using the `connect()` function. This takes a Projection as its argument, which in its simplest form is a vector of connections to make. More sophisticated Projections can specified algorithmically. This allows rules for making connections to be based on the state of source and destination entities (e.g. their position and orientation).

```
/* Connect each neuron in a population to its neighbour */
Population popn = root.getChild( "Neurons" );
Vector connections = new Vector();
PortID op = new PortID(1);
PortID ip = new PortID(1);
Time delay = new Time(1.0);

for (int i=0; i<popn.getNumEnts(); i+=2) {
     EntityID srce  = popn.getEntityID(i);
     EntityID deste = popn.getEntityID(i+1);
     connections.addElement( new ConnectionSpec( srce, op, deste, ip, delay ) );
}
Projection p = new ProjectionImpl( connections );
connect( p );      // Actually make the connections
```

The default Neosim connection just includes source/destination entities and ports with a delay. To include extra information such as weights, an extended form of the connection spec is needed. The example below shows a connection specifcation which adds segment number, weight and location.

```
/** NEURON specific connections - adds weight, location and segment#
```

```
 */
class NeuronConnectionSpec extends ConnectionSpec {
    double weight, location;
    int segment;
    NeuronConnectionSpec(EntityID srce, PortID srcpid, EntityID deste, PortID destpid,
                         double weight, int segment, double location) {
        super( srce, srcpid, deste, destpid, delay );
        this.weight = weight;
        this.location = location;
        this.segment = segment;
    }

    /** Set any extra connection parameters at the destination end
     * e.g. a weight
     */
    public void setDstParams( Entity deste, ConnectionID destcid ) {
        NeuronEntity ne = new NeuronEntity(deste); // Cast to Neuron entity
        ne.setSynapseParams( destcid, weight, segment, location );
    }
}
```

This connection specification can be used for connecting to NeuronEntity entities, which have an extra user defined method for setting the extra parameters. The setSrcParams() method can also be overridden to allow for setting thresholds and any other parameters required at the source of the connection.

3 Generalised Projections - user supplied methods

In the previous section the connections to make were known a priory; they did not depend on the internal state of the source and destination entities. This will not be the case in general, and more complex projection schemes will depend on the relation between both source and destination entities.

The key to the ability to make any kind of connection is that the user can provide a *SourceMethod* to generate *ConnectionRequests* and a corresponding *DestMethod* to handle them.

3.1 Example all to all connection

To specify an all to all connection, the user provided *SourceMethod* code would look like:

```
class MySourceMethod extends SourceMethod {
  /**
   * The sendRequests method checks the source entity's details
   * and sends connection requests to individual destination
```

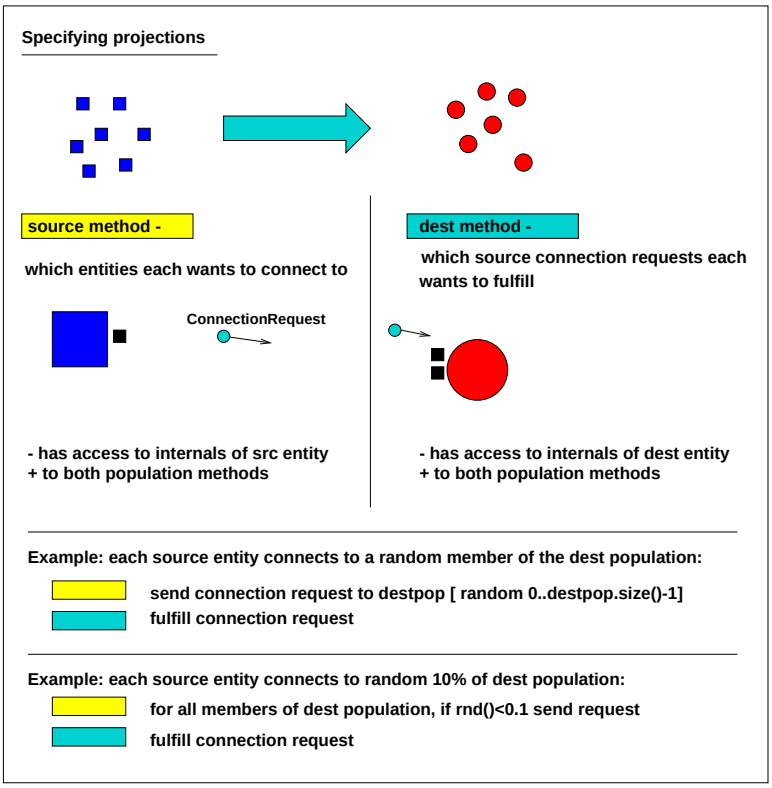

Figure 1: Specifying projections in a general way.

```
     * entities (using sendRequest) or to all members of a population
     * (using bcastRequest).
     */
    void sendRequests( Entity srce, Population destPop, DestMethod dm ) {
       srce.bcastRequest( destPop, new ConnectionRequestImpl( srce.getID(), srcPort, dm )
    }
  }
```

and the *DestMethod* handler would look like:-

```
  class MyDestMethod extends DestMethod {
    /** The considerRequest method checks the source entity's details
     * against the dest entity's details, and returns True or False
     * depending on whether or not the connection will be made.
     *   deste - the destination entity
     *   cr    - the connection request from the source entity
     *   returns true if a connection made, false otherwise.
     */
    boolean considerRequest( Entity deste, ConnectionRequest cr ) {
      Connection c = new ConnectionSpec( cr.getSrcEntityID(), cr.getSrcPortID(),
                                         deste.getID(), destPort, delay );
      this.deste = deste;
      makeConnection( c );
      return true;
    }
  }
```

This method always fulfils connection requests. A probabilistic version could sample some random number to make 10% of connections:-

```
  boolean considerRequest( Entity deste, ConnectionRequest cr ) {
    // ...
    if (rnd.sample() < 0.10) {
      makeConnection( c );
      return true;
    } else {
      return false;
    }
  }
```

3.2 Distance-based connections

If the decision about whether to make a connection depends on something like the relative position of the two entities, the connection request to send has to be extended to include the extra position information:-

```
  // the new ConnectionRequest...
  class PosnConnectionRequest extends ConnectionRequest {
```

```
  public Position srcpos;
  // ...
}

// the new SourceMethod...
class PositionSourceMethod extends SourceMethod {
  void sendRequests( Entity srce, Population destPop, DestMethod dm ) {
    srce.bcastRequest( destPop,
         new PosnConnectionRequest( srce.getID(), srcPort, dm,
                                    srce.getPosition()) );
  }
}

// the new DestMethod...
class PositionDestMethod extends DestMethod {
  boolean considerRequest( Entity deste, ConnectionRequest cr ) {
    PosnConnectionRequest pcr = (PosnConnectionRequest)cr;
    // calc distance b/w source and dest entity...
    double distance = dist( deste.getPosition(), pcr.srcpos );
    // decide to make the connection based on some function...
    if (distance < 100e-6) {
      makeConnection( c );
      return true;
    } else return false;
  }
}

// and actually make the connections...
connect( new GenProjection( new PositionSourceMethod( srcPop ),
                            new PositionDestMethod( destPop ) ) );
```

Although this method for defining projections may look long-winded, users should only need to write code like this if their connection scheme is highly specific. Common connection schemes such as are provided by the GENESIS volumeconnect functions can be included as standard - it is only if the user's model needs extra information about the source entity to decide on connections that extra source or destination methods have to be written.