

# Multiple Interacting Instantiations of Neuronal Dynamics (MIIND): a Library for Rapid Prototyping of Models in Cognitive Neuroscience

Marc de Kamps and Volker Baier

**Abstract**—Modelling the neuronal substrate of higher cognitive functions is very time consuming. Apart from a few exceptions, the models are only available in print. The source code behind these models is hardly ever published, which means that the model has to be re-implemented if it is to be extended or to be examined in detail. This prevents the creation of really complex models, since it is impossible to build on existing models. Many models share quite a number of characteristics, however, which can be captured by high level object oriented programming languages. Factoring out these characteristics leads to low level components which can be reused in a surprisingly versatile manner. In this paper we present a number of such components, which collectively are called MIIND. We will give examples of their use and will indicate how they could be extended.

## I. INTRODUCTION

Modelling the neuronal substrate of higher cognitive functions has always been difficult, if only because we don't know much about it. The situation is considerably worse than in computational neuroscience, where at least sound hypotheses can be formulated about the basic units: the neurons. It is no coincidence that in this area packages like NEURON and GENESIS have gained a foothold, which reduce the programming load on the modeller considerably. In the modelling of higher level cognitive functions, however, there are no accepted standard techniques. In this paper, let us restrict "cognitive modelling" to mean the modelling of primate behaviour in terms of a neural model. This is still a fairly inclusive definition: it may refer to for example Connectionist models to explain aspects of language production[1] or to models which try to interpret neurophysiologic data from experiments in visual attention [2] or to detailed models of visual cortex [3], [4] and covers a vast body of literature. So great is the variation in modelling approaches and objectives that often they seem to have little in common. The point that this paper tries to make is that there *are* commonalities in all of these studies, which can be expressed in programming code and that it is *essential* to do so if we are to make progress in understanding cognition.

Accurate modelling of the neural substrate of cognitive processes is at least as difficult as modelling in neuroscience, because in order to describe behaviour in neuronal terms, we

cannot extrapolate from individual neuron or small network models. We need the *emergent* properties of hundreds of thousands of neurons, as a description of *behaviour* in terms of individual neurons is out of the question. At this moment the level of detail of the experimental data on this level (cortical column and higher) is clearly more limited than in neuroscience, where *in vitro* and *in vivo* experiments have yielded detailed and reliable information which have constrained models.

This may be about to change. In the last decade we have seen a radical increase in the sophistication of non-invasive technique such as fMRI, EEG, PET and others. These techniques are able to identify global cortical networks and to reveal the interactions between entire brain areas. In conjunction with multi-electrode array data they will provide new experimental constraints for modelling cognitive processes, but it must be realised that the experimental signal is caused by the collective response of thousands or even millions of neurons. This is an area where the *emergent* properties of groups of neurons will become visible, an area which is not well covered by computational neuroscience at the time. This is also an area which goes well beyond Connectionism, which so far hasn't been overly concerned with the biological plausibility of its networks.

Clearly, models which on the one hand describe cognitive phenomena at a sufficiently high level to allow an explanation in common sense language, but on the other hand are realistic in their biological details, involve several levels of understanding. They must span several spatial and temporal scales. Good models also are likely to be created by heterogeneous research groups, each with their own expertise on parts of the model. The challenge then is to offer them a *framework*, where specialists can embed their specific knowledge into a larger context. If this framework is flexible, it will become possible to strike a balance between detailed local simulations when necessary, or coarse parameterized versions of the same brain area when sufficient. An example of the former case might be the influence of a particular neurotransmitter on working memory, whereas an example of the latter might be a statement like 'we know that prefrontal cortex provides the bias for competition in this area'. Importantly, such a framework is not just about the efficiency of simulation. It is also about sharing information and expertise between research groups and individuals. Different research groups can work on different parts of the model. Modelling cognition then in a way becomes the effective integration of

Marc de Kamps is with the Biosystems Group of the School of Computing, University of Leeds, Leeds, LS29JT, UK (phone: +44 113 343 5322; fax: +44 113 343 5457; email: dekamps@comp.leeds.ac.uk). Volker Baier is with Neuro-Cognitive Psychology, University of Munich, Leopoldstrasse 13, Munich, Germany (phone: +49 (89) 2180 4801; email: V.B@campus.lmu.de)

such specialized into a coherent framework.

In this paper, we present MIIND (Multiple Interacting Instantiations of Neuronal Dynamics), which might be considered as a case study for building such a framework. In MIIND we combine our experience in computational neuroscience and cognitive modelling. MIIND started out as a Connectionist model, but the necessity to include more realistic descriptions of neuronal activity in some of our work raised the questions of how to integrate them in our large-scale models. The code has been used in a large number of models [5], [6], [2], [7], [8], [9], [10], [11] and in some cases the models that we published are available as test suites. We believe that the approach taken here may be extended to include other areas of expertise as well such as visualization, anatomical and functional connection data, etc. The code of MIIND is available on SourceForge (<http://miind.sf.net>).

## II. THE PHILOSOPHY OF MIIND

The philosophy of MIIND is to factor out the commonalities in cognitive models as early as possible and to make them available as separate components. The idea is that such components can be reused easily because they offer simple solutions for well defined problems, which occur over and over again. Because the scope of the components is limited, they can be used to solve specific problems without forcing a modelling philosophy on users or forcing them to adopt a complex framework in order to start modelling. It is also hoped that because they are well limited in scope, components will mature and attain stable user interfaces quickly. Although we are aware of other packages in Cognitive Neuroscience, in particular PDP++ [12] which has a lot to offer, we feel that they tend to constrain the modelling philosophy of the user.

## III. THE REPRESENTATION OF SPARSE NETWORKS

### A. Basic idea

At first sight the issue of representing networks in programming code seems rather trivial. After all a network is just a set of nodes and a set of directed links, which can be conveniently represented by an adjacency matrix. Although such a representation is simple indeed, in many cases it is wasteful and would limit the size of the networks that can be realized in a simulation. Most biological networks are very sparse, which means that most entries in the adjacency matrix are zero. A more efficient method is to store with each node a list of links to the nodes that connect to it. In C++ this leads to a very specific data structure for a node, which is explained in Figure 1. In MIIND the concrete implementation of such a node is called a SparseNode. A network can then be represented by a collection of SparseNodes, which is called a SparseImplementation. For a single link in the network we need two numbers in a node to represent a link: one which indicates to which other node this node is connected and one which indicates the 'strength' of this connection. If we would have used an adjacency matrix to represent the network, we would only have had to use

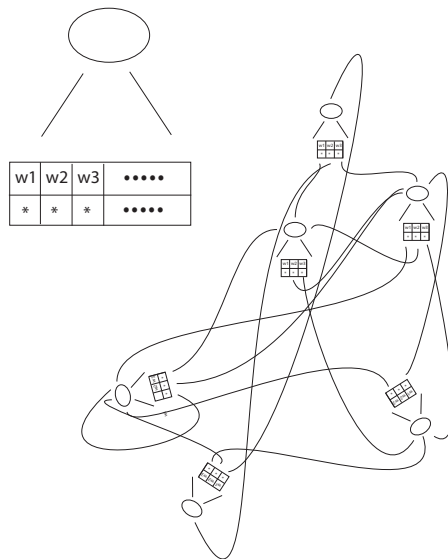


Fig. 1. A C++ data structure for representing sparse networks. Each node contains a list. The list contains pointer-weight pairs. The pointer points to a node with which this one is connected and the weight represents the strength of the connection. A collection of such nodes represents a network.

one single number, i.e. the relevant entry of the matrix to indicate the strength of the connection. The deciding advantage of SparseNodes, however, is that links which don't exist are not represented, contrary to the adjacency matrix representation. This means that as soon as a network has less than half of the number of possible combinations realized, the SparseImplementation is more efficient in terms of memory use. Since biological networks are typically much sparser than that, SparseImplementation is an obvious choice for a data structure. A SparseNode is characterized by a state variable, which is called Activity. Non-zero elements of the adjacency matrix are typically called Weights or Efficacies. A SparseNode is able to compute InnerProducts over its Weights and the Activities of the Nodes to which it is connected. The default form of InnerProduct is the standard scalar product  $\sum_j w_j E_j$ , where  $j$  labels the input nodes of a given node,  $E_j$  is the activity of that node and  $w_j$  is the weight of the link from input node  $j$  to the node, but others are possible as well (see e.g. Equations 4 and 5).

Each Node contains a list. The list contains pointer-weight pairs called Connections. The pointer points to a node that connects to the parent Node of the Connection and the weight represents the strength of the connection. A collection of such nodes represents a network. A Node provides very simple access to its Connections. A Node can generate a begin iterator [13], which points to the first Connection of a Node and an end iterator, which points *one past* the last Connection. In this manner there is a simple way to implement loops over a Node's inputs in the same style as C++'s Standard Template Library [14]. Since the iterator directly accesses all Connections subsequently, the loop is very efficient, it only iterates over Connections which exist.

An important aspect of the `SparseNode` representation is that both the `Activities` and the `Efficacies` are template parameters, i.e. they can have every desired type. This gives an amazing flexibility, which may be new to researchers who aren't familiar with C++. In short, the fact that `Activities` are template parameters means that the type of the `Activity` need not be single floating point number. It may be a *struct* (or a record for people familiar with PASCAL or FORTRAN), a type which may store more than one single point value. It may even be something bizarre like a string, or indeed any user defined class. The same holds for the `Weights`. Below, we will examine an example of a connection between two neuronal populations, which is not only determined by an `Efficacy` but also by an effective Number of connections (see the example of Section VI-B). One may also envision situations where a connection isn't only characterized by efficacy, but also by delay time. In all such cases it's useful to use the same code for network representations and only vary the type of the connection. Heterogeneous networks can also easily be created. There are two requirements that such a network has to fulfil:

- A Node has to be able to 'make sense' of the `Efficacies` that it receives: it has to be able to calculate an `InnerProduct` from the `Efficacies` that it receives and the `Activities` of the Nodes which connect to it.
- Simpler types of connections must be embedded in the most complicated type of the network.

The possibility of making heterogeneous networks is very important. It means that parts of the network which have to be solved using CPU intensive methods can receive more CPU power than more trivial parts of the network. This is also make it easy for different models, possibly from different research groups, to be accommodated in the framework.

#### IV. CONNECTIONIST NETWORKS - SPATIAL RELATIONS

It is easy to build Connectionist networks using this library. Sparse Connectionist networks can be easily represented by adopting a squashing function for each node. By providing methods to store patterns into specific parts of the `SparseImplementation` and to read out other parts and specifying an `Order` in which the nodes are updated, one has a conventional (Connectionist) Artificial Neural Network (ANN). It turns out to be beneficial to make a special `LayeredSparseImplementation`. This implementation keeps track of which Node is in which Layer, how many Nodes a given Layer contains, etc. Importantly, the `LayeredSparseImplementation` also provides iterators to each Layer. Iterators are typically used to point to the begin Node of each Layer and *one past* the end Node of each Layer. These iterators provide a simple uniform access method to loop over a single Layer.

As is clear from Figure 1, a Node only knows about the Nodes that connect to it, its 'predecessors'. For evaluating neural networks this is often enough, but in some cases such as in the Backpropagation algorithm, activity in the network has to be reversed. In this case a Node needs to

know about its 'predecessors' and its 'successors'. A special kind of a `SparseNode`, a `ReversibleSparseNode` generates a second list of so-called `ReverseConnections` from the original `Connections` lists of all Nodes. The `ReversibleSparseNode` also provides iterators. A moderately complex algorithm like Backpropagation [15], [16] can now be programmed in a very simple way. The iterators know where a Layer ends, or where a list of `Connections` ends, so the loops over layers become very simple. The hard work: 'at which Node should the loop over a Layer stop?' has already been done by the provider of the iterators. This is a well known technique to simplify code which contains many nested loops [14], but it also makes algorithms independent of the underlying representation: should we later decide to use an `AdjacencyMatrix` representation after all, there will be some work to provide the corresponding iterators, but the algorithm can work on them just as well.

Biological networks are often determined by spatial relations rather than abstract mathematical descriptions. Ventral stream in visual cortex, for example, is often modelled by a hierarchical multilayer feedforward network, where the nodes in each layer have a receptive field: only nodes in a previous layer which are close enough to a given node will be input to this node.

It is tedious and very error-prone work to programme these relations. It is an enormous boost to productivity if a user only needs to specify the parameters of a given spatial relation, rather than programme it from scratch. MIIND provides the `DenseOverlapLinkRelation`, which is described in Figure 2 and a number of other `LinkRelations`, which are a bit more specialized. It is easy to derive from `AbstractLinkRelation` if the existing `LinkRelation` types are not able to instantiate the desired architecture. They need to be programmed only once and if they are added to the framework, they will be available to other users.

`LinkRelations` make it possible to define highly organized spatial structures which can be defined by only a relatively small number of parameters and once they are defined, they can easily be reused.

#### V. A GENERIC REPRESENTATION OF SYSTEMS OF COUPLED EQUATIONS

Many biological processes actually can be represented by sparse networks. A description of the evolution of the process then usually entails the solution of a set of coupled (differential) equations. If the system is complicated it pays to separate the methods for constructing the network from the methods that implement the `EvolutionAlgorithm`. The modeller only has to concentrate on modelling processes at an individual Node and to specify how in general one Node influences another node. If this is done rigorously, it is possible to guarantee that the entire network will be evolved correctly, no matter how large it is.

To solve a system of equations, it is sufficient to start with an array of `DynamicNodes`, which are `SparseNodes` endowed with two extra attributes: a `NodeState` and an `AbstractAlgorithm`. The `AbstractAlgorithm` 'knows' on which type of

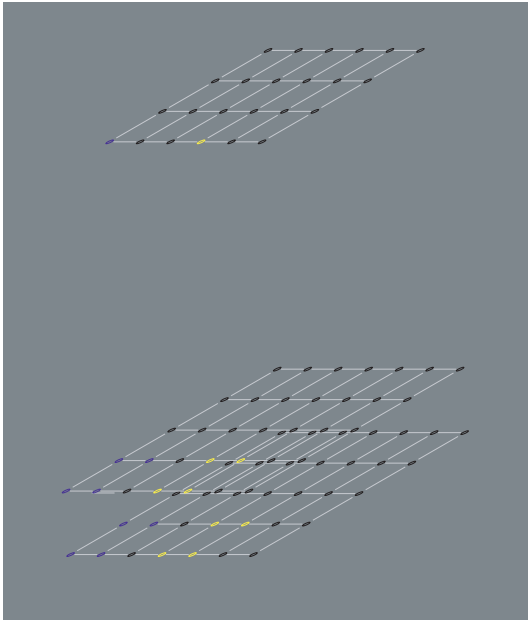


Fig. 2. Even a simple feedforward structure must be specified by a large number of parameters: the number of layers, the size of each layer in two dimensions, the size of the receptive field of a node in previous layers in two dimensions, the relative position of the receptive fields of two neighbouring nodes. In the figure the lighter coloured nodes in the lower two layers are in the receptive field of the neuron in the higher layer. The connections are not shown. Recurrent structures of cortical areas involving multiple such networks need a great deal more parameters.

NodeState it is operating and can generate the NodeState that is used by its parent Node. A DynamicNetworkImplementation now *is a* SparseImplementation augmented with an Evolve method. The Evolve method is a generic simulation loop. When the Evolve method is called, each DynamicNode is visited in turn and this calls upon its AbstractAlgorithm instance to update its NodeState.

Those who are familiar with object oriented programming (OOP) will have guessed correctly that AbstractAlgorithm is abstract, in plain English: it is a stub, which is to be replaced by a concrete Algorithm. Several often used algorithms are already provided with MIIND (see the example sections).

## VI. SOME EXAMPLES

### A. Artificial Neural Networks and Wilson-Cowan dynamics

A simplified version of Wilson-Cowan dynamics is given by:

$$\frac{dE_i}{dt} = -E + S\left(\sum_i w_{ij} E_j\right), \quad (1)$$

where  $E_i$  is the firing rate of a neuron (population, node),  $S$  some sigmoid function and  $w_{ij}$  a weight matrix which represents the efficacy of the activity of node  $i$  on node  $j$ . Equation 1 describes the evolution of the firing rate of a population under the influence of external input from other populations  $E_j$  [17]. (One may also encounter the form  $\sum_j w_{ij} S(E_j)$ , see e.g. [18], [19]). These equations, sometimes with minor variations, are by far the most widely

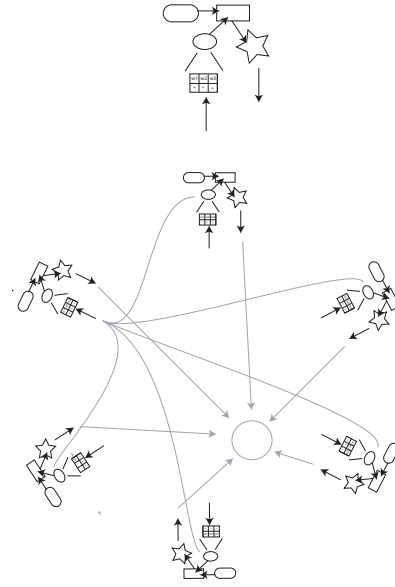


Fig. 3. A single DynamicNode is derived from a SparseNode. It has an AbstractAlgorithm (oval), which operates on a NodeState (rectangle). When prompted by the simulation loop, the AbstractReportHandler sends the current NodeState to a central file. DynamicNodes are almost autonomous. The central simulation loop determines which Node is in line to evolve its NodeState over a short time step, but the Nodes themselves collect input from other Nodes and deliver this to their own Algorithms which evolve the Node's NodeState. This setup is easy to parallelise.

used equations in large-scale models of cortical dynamics, see [2], [5], [18], [19], [20] for just a few examples. And they are not only used in simulations, but also, for example, in the interpretation of fMRI data [21]. Their main attractive features are: they are probably not very wrong when the neuronal dynamics is near equilibrium and fast transients are not important in the model [22], as long as their activity is interpreted as a population firing rate and not the membrane potential of a single neuron. They can easily be identified with ANNs since the equilibrium state of Equation 1 corresponds exactly to the update equation of an artificial neuron. Since they can be easily solved numerically at low computational cost, using several thousands of populations in a simulation is not a problem. They are also simple to program. The organization of the populations in a spatial connectivity structure, which approximately resembles the brain areas under study, is usually much more work.

The models mentioned above hardly differ from each other in terms of mathematical structure. Typically they entail the embedding of the populations in a spatial structure and solving the resulting system of Equations 1. The spatial structures in all of these models resemble each other strongly: mostly they consist of hierarchically organized two dimensional fields of populations with a highly regular internal connections. Most of these models could be easily implemented with the LinkRelations described above to define the spatial connectivity of the models and the WilsonCowanAlgorithm that is provided with MIIND to evolve the DynamicNetwork

of such populations.

### B. Coupled systems of Fokker-Planck equations

Wilson-Cowan equations are important and dominate the literature when it comes to large-scale cortical modelling, but they are unrealistic when phenomena are considered which are characterized by fast transients or oscillatory phenomena. Also the fact that the *variability* of the neuronal input is ignored and only the mean of the input is considered can be a substantial limitation in some cases. For these reasons, there is a considerable interest in the use of population density techniques to describe the response of groups of spiking neurons in response to spiking variable input [23]. In cortex the evolution of the population density for a group of Leaky-Integrate-and-Fire (LIF) neurons is assumed to be described reasonably well by a Fokker-Planck equation:

$$\frac{\partial \rho(v, t)}{\partial t} = \frac{\sigma^2}{2} \frac{\partial^2 \rho(v, t)}{\partial v^2} + \frac{\partial}{\partial v} [(v - \mu) \rho(v, t)] \quad (2)$$

Here  $v$  is the membrane potential of a neuron,  $\rho(v)dv$  is the fraction of the population which has its membrane potential in  $[v, v + dv)$ ,  $\mu$  the mean of the input contribution, defined below and  $\sigma$  the variance of the input contribution. The firing rate of a population can be computed from its population density:

$$\nu = -\frac{1}{2\sigma^2} \frac{\partial \rho(v_{th}, t)}{\partial v}, \quad (3)$$

where  $v_{th}$  is the threshold potential of neurons in the population.

It's assumed here that the input spike train to the population are Poisson distributed, which for high input rates becomes approximately Gaussian [24]. If we connect various neuronal populations with each other, these connections are characterized by a distribution of neuronal *efficacies*, but also by the *number* of connections. More specifically, it's assumed that the mean of population  $i$ 's input is given by Equation 4 and the variance of the input by Equation 5:

$$\mu_i = \tau_i \sum_j J_{ij} N_{ij} \nu_j, \quad (4)$$

$$\sigma_i = \sqrt{\sum_j \tau_i J_{ij}^2 N_{ij} \nu_j}, \quad (5)$$

where the  $J$  are the effective efficacies between two populations and the  $N$  the effective number of connections.

This sets the stage for a description of a circuit which consists of several populations: for each population equation 2 holds and the coupling is described by Equations 4 and 5, where the output rates which enter these Equations are calculated with Equation 3.

Modelling such systems constitute a considerable challenge: in the first place the implementation of algorithms for solving Fokker-Planck equations are non-trivial, in the second place they are CPU-intensive. Moreover, the links in the network which represents a coupled system are now determined by *two* numbers: not only the  $J_{ij}$ , but also the  $N_{ij}$ . The last problem can be solved easily: C++'s

template mechanism yields a compiler generated version of a DynamicNetwork where links are not determined by a single weight, but by a  $J - N$  pair. Recently de Kamps has provided stable and efficient algorithms for solving Equation 2 [7], [25] and since these algorithms are part of MIIND (PopulationAlgorithm), it is now possible to set up networks in a similar way as networks of Wilson-Cowan equations.

### C. Rapid prototyping - Simplified Dynamics

In an analysis of a potential mechanism for the implementation of short-term memory by spiking neurons Amit and Brunel [24] made considerable simplifications which allow the description of steady-state rates of the networks described above. They showed that solving the following closed system of equations

$$\nu_i = \phi_i(\mu_i, \sigma_i), \quad (6)$$

where:

$$\phi_i(\mu_i, \sigma_i) \equiv \left\{ \tau_{r,i} + \sqrt{\pi} \tau_i \int_{\frac{\mu_i - \theta_i}{\sigma_i}}^{\frac{v_{r,i} - \mu_i}{\sigma_i}} du [1 + \operatorname{erf}(u)] e^{u^2} \right\}^{-1} \quad (7)$$

and  $\mu_i$  and  $\sigma_i$  are given by Equations 4 and 5, respectively, gives steady-state solutions of the coupled Fokker-Planck equations. It is easy to find solutions to the system defined by Equation 6 by introducing a Wilson-Cowan-like pseudo-dynamics:

$$\frac{d\nu_i}{dt} = -\nu_i + \phi_i(\mu_i, \sigma_i) \quad (8)$$

This is computationally less intensive and more amenable to analytical analysis. *In principle, it is now possible to separate network structure from the algorithm that is used at the Nodes.* It is possible to find a network structure which has sensible steady-state behaviour first, using the grossly simplified dynamics of equation 8. If satisfactory network connections have been found, the dynamics of Equation 8 can simply be exchanged by that of Equation 2.

### D. Incorporating Monte Carlo simulations

Both the population density formalism and Wilson-Cowan equations simulate the collective activity of a large population of LIF neurons. It is also possible to simply simulate a large group of LIF neurons directly and to encapsulate these simulations in an AbstractAlgorithm interface. At the moment we are working to create a MonteCarloLifAlgorithm. In a similar way, more detailed simulations by packages as NEURON or GENESIS could be incorporated, when the influence of particular ion channels or morphology must be investigated. The important assumption that MIIND makes is that the influence from one neuronal population on another can be specified by a relatively small number of parameters, which can be described as Weights.

## VII. DISCUSSION

We hope that, the philosophy of MIIND is visible. If one is only interested in representing sparse networks, then one

can use the SparseImplementation library. The DynamicNetwork combined with the WilsonCowanAlgorithm allows the solution of a large set of coupled Wilson-Cowan equations without further ado, but if the Nodes that represent the individual populations need to be embedded in a spatial structure, StructnetLib can be invoked and with the use of suitably chosen LinkRelation objects complicated spatial connection structures can be instantiated. What can be done with Wilson-Cowan dynamics can be done *mutatis mutandis* for networks of coupled Fokker-Planck equations. They too can be embedded in complex spatial hierarchies. First experiments with Fokker-Planck versions of models that were earlier cast in terms of the steady-state solutions of Equation 6 [8] have already yielded some surprises: violent oscillations occurred in Fokker-Planck systems before equilibrium was reached, which means that these models have been falsified to some extent. It turns out that the oscillations in some cases can be explained by theoretical analysis (see e.g. [26]). Although one might argue that the theoretical analysis should have been done first, in order to prevent such surprises, this is not always possible, certainly not in larger networks. The possibility to move between a superficial and a deeper level of simulation gives the modeller much more opportunity to experiment and verify his or her assumptions on several levels.

The modelling literature of cognitive neuroscience contains a large number of models that are strikingly similar when their cognitive motivations are ignored and they are examined exclusively from a mathematical point of view. Since most models are programmed from scratch, the community has wasted much effort on reimplementing code which already existed somewhere. We hope that MIIND offers modellers the software infrastructure they need to implement this kind of model and that they can concentrate more on modelling and less on implementation. We also hope that reuse and extension of existing models will become more commonplace than it is now.

MIIND is available on SourceForge (<http://miind.sf.net>). It is released under a modified BSD license<sup>1</sup>. On the website instructions for use of the software are available. MIIND is aimed at developers, i.e. experienced C++ programmers. The software is provided in the form of libraries. The Fokker-Planck algorithms still need work to make them perform reliably with network parameters that lie outside the range with which we have experimented. To make MIIND a truly parallel framework, which can run on large cluster machines, an MPI implementation of the central DynamicNetwork evolution loop is a high priority. Nevertheless, we believe that MIIND already has a lot to offer for modellers in Cognitive Neuroscience.

#### REFERENCES

[1] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

<sup>1</sup>The modification requests that the the package be cited when used in the preparation of scientific publications, but doesn't affect the right to freely use and modify the source code in any way.

[2] F. van der Velde and M. de Kamps, "From knowing what to knowing where: Modeling object-based attention with feedback disinhibition of activation," *Journal of Cognitive Neuroscience*, vol. 13(4), pp. 479–491, 2001.

[3] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature Neuroscience*, vol. 2, pp. 1019–1025, 1999.

[4] T. Serre, M. Kouh, C. Cadieu, U. Knoblich, G. Kreiman, and T. Poggio, "A theory of object recognition: Computations and circuits in the feedforward path of the ventral stream of primate visual cortex." AI Memo 2005-036, CBCL Memo 259, 2005.

[5] M. de Kamps and F. van der Velde, "From artificial neural networks to spiking populations of neurons and back again," *Neural Networks*, vol. 14, pp. 941–953, 2001.

[6] M. de Kamps and F. van der Velde, "Using a recurrent network to bind form, colour and position into a unified percept," *Neurocomputing*, vol. 38–40, pp. 523–528, 2001.

[7] M. de Kamps, "A simple and stable numerical solution for the population density equation," *Neural Computation*, vol. 15, pp. 2129–2146, 2003.

[8] M. de Kamps, "A model for delay activity without recurrent excitation," *Lecture Notes in Computer Science*, vol. 3696, pp. 229–234, 2005.

[9] F. van der Velde and M. de Kamps, "Neural blackboard architectures of combinatorial structures in cognition," *Behavioral and Brain Sciences*, vol. 29(1), pp. 37–70, 2006.

[10] V. Baier, "Motion perception with recurrent self-organizing maps based models," in *IJCNN 2005 Conference Proceedings*, IEEE, 2005.

[11] V. Baier, *Motion Perception and Prediction: A Subsymbolic Approach*. Electronic, Chair for Theoretical Computer Science and Foundations of Artificial Intelligence, TU München, November 2006.

[12] R. O'Reilly and J. W. Rudy, "Conjunctive representations in learning and memory. principles of learning in the neocortex and hippocampus," *Psychological Review*, vol. 108, pp. 311–345, 2001.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1994.

[14] S. Meyers, *Effective STL - 50 specific ways to improve your use of the Standard Template Library*. Reading, MA: Addison-Wesley, 2001.

[15] J. L. McClelland, D. E. Rumelhart, and the PDP research group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition (Volume 1,2)*. Cambridge (MA): The MIT Press, 1986.

[16] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences." PhD thesis, Harvard University, 1974.

[17] H. R. Wilson and J. D. Cowan, "Excitatory and inhibitory interactions in localized populations of model neurons," *Biophysical Journal*, vol. 12, pp. 1–23, 1972.

[18] M. Usher and E. Niebur, "Modeling the temporal dynamics of it neurons in visual search: A mechanism for selective top-down attention," *Journal of Cognitive Neuroscience*, vol. 8, pp. 311–327, 1996.

[19] L. J. Lanyon and S. L. Denham, "A model of active visual search with object-based attention guiding scan paths," *Neural Networks*, vol. 17, pp. 873–897, 2004.

[20] F. H. Hamker, "The reentry hypothesis: The putative interaction of the frontal eye field, ventrolateral prefrontal cortex, and areas v4, it for attention and eye movement," *Cerebral Cortex*, vol. 15, pp. 431–447, 2005.

[21] K. J. Friston, L. Harrison, and W. Penny, "Dynamic causal modelling," *Neuroimage*, vol. 19(4), pp. 1245–1254, 2003.

[22] W. Gerstner, "Population dynamics of spiking neurons: Fast transients, asynchronous states, and locking," *Neural Computation*, vol. 12, p. 4389, 2000.

[23] B. W. Knight, D. Manin, and L. Sirovich, "Dynamical models of interacting neuron populations in visual cortex," in *Symposium on Robotics and Cybernetics: Computational Engineering in Systems Applications* (E. C. Gerf, ed.), France: Cite Scientifique, 1996.

[24] D. J. Amit and N. Brunel, "Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex," *Cerebral Cortex*, vol. 7, pp. 237–252, 1997.

[25] M. de Kamps, "An analytic solution of the reentrant poisson master equation and its application in the simulation of large groups of spiking neurons," in *Proceedings WCCI2006 (IJCNN2006)*, (Vancouver, CA), 2006.

- [26] N. Brunel and V. Hakim, "Fast global oscillations in networks of integrate-and-fire neurons with low firing rates," *Neural Computation*, vol. 11, pp. 1621–1671, 1999.