



2008 Special Issue

The state of MIIND

Marc de Kamps^{a,*}, Volker Baier^b, Johannes Drever^c, Melanie Dietz^c, Lorenz Mösenlechner^d, Frank van der Velde^e

^a Biosystems Group, School of Computing, University of Leeds, LS2 9JT Leeds, United Kingdom

^b Neuro-Cognitive Psychology, Ludwig-Maximilians Universität München, Leopoldstrasse 13, München, Germany

^c Robotics and Embedded Systems, Institut für Informatik, Technische Universität München, Boltzmannstrasse 3, D-85748 Garching bei München, Germany

^d Image Understanding & Knowledge-Based Systems, Institut für Informatik, Technische Universität München, Boltzmannstrasse 3, D-85748 Garching bei München, Germany

^e Leiden Institute for Brain and Cognition, Cognitive Psychology, Leiden University, Wassenaarseweg 52 2333 AK Leiden, The Netherlands

ARTICLE INFO

Article history:

Received 31 October 2007

Received in revised form

9 June 2008

Accepted 28 July 2008

Keywords:

Neuronal modelling

C++

Framework

Population density methods

Python

ABSTRACT

MIIND (Multiple Interacting Instantiations of Neural Dynamics) is a highly modular multi-level C++ framework, that aims to shorten the development time for models in Cognitive Neuroscience (CNS). It offers reusable code modules (libraries of classes and functions) aimed at solving problems that occur repeatedly in modelling, but tries not to impose a specific modelling philosophy or methodology. At the lowest level, it offers support for the implementation of sparse networks. For example, the library SparseImplementationLib supports sparse random networks and the library LayerMappingLib can be used for sparse regular networks of filter-like operators. The library DynamicLib, which builds on top of the library SparseImplementationLib, offers a generic framework for simulating network processes. Presently, several specific network process implementations are provided in MIIND: the Wilson–Cowan and Ornstein–Uhlenbeck type, and population density techniques for leaky-integrate-and-fire neurons driven by Poisson input. A design principle of MIIND is to support detailing: the refinement of an originally simple model into a form where more biological detail is included. Another design principle is extensibility: the reuse of an existing model in a larger, more extended one. One of the main uses of MIIND so far has been the instantiation of neural models of visual attention. Recently, we have added a library for implementing biologically-inspired models of artificial vision, such as HMAX and recent successors. In the long run we hope to be able to apply suitably adapted neuronal mechanisms of attention to these artificial models.

© 2008 Elsevier Ltd. All rights reserved.

1. The Philosophy of MIIND

1.1. The need for interoperability of models

Looking back on the developments in cognitive neuroscience over the last decade, the progress of experimental techniques is striking. Functional Magnetic Imaging Resonance (fMRI) machines have become commonplace and many of them are now purely dedicated to cognitive research. Techniques, such as EEG, PET, MEG and others have also matured and are increasingly used in conjunction with fMRI. Technological advances have allowed the development of novel ways of connecting electronic readout to neurons, leading to bigger multi-electrode arrays (MEAs) (Navarro et al., 2005; Nicolelis et al., 2003) or to neuron-on-silico chips (Fromherz, 2003; Schoen & Fromherz, 2007). This process seems to reinforce itself: the more we know about specific parts of the brain, the better we can plan the next experiment.

Enormous progress has also been made in the understanding of single neurons and neuronal systems. Researchers are beginning to explore large-scale models of biologically detailed networks (e.g., Djurfeldt et al. (2008)). Relatively simple neuronal models now exist that model experimental data accurately, but which are much simpler than Hodgkin–Huxley type models (Brette & Gerstner, 2005). The BlueBrain project (e.g., Markram (2006)) aims to model an entire cortical column in realistic detail. The effects of synaptic plasticity rules can now be investigated in great detail both on the neuronal (e.g., Gerstner and Kistler (2002)) and the network levels (Morrison, Aertsen, & Diesmann, 2007). But where experimental (cognitive) neuroscience has reached a breakthrough at the system level, and allows us to observe the global flow of neural activity in the active brain, computational modelling is yet to follow suit here. In general, progress in computational neuroscience has not informed models of high level cognition. In terms of understanding high level cognition, mental illnesses and endowing artifacts with human-like intelligence we have not reached a breakthrough at the system level.

What is the cause for this? Although it is certainly true that means for modelling and theory building are not commensurate

* Corresponding author.

E-mail address: dekamps@comp.leeds.ac.uk (M. de Kamps).

with the enormous investments in experimental equipment, in the opinion of the authors this is not the main reason. To include a realistic level of detail in models of high level cognition a lot of work will be necessary, which will involve a large number of people and expertise from various disciplines. However, in modelling we seem to have hit something of a complexity wall: most modelling is done on a project-by-project basis, in relatively small groups. Typically, the results of modelling are published, but publishing the implementation of the model, be it in the form of source code or an executable program (or in other forms: e.g. CORBA/COM objects), is rare, although with the recent arrival of ModelDB (<http://senselab.med.yale.edu/modeldb>) this may slowly change. This means that the level of complexity of the model in general will not exceed that what can be achieved by a small research group (1–5 people) over a limited period of time (2–5 years). This is not a problem if one believes that the brain can be understood at a relatively high level of abstraction, and with relatively simple models, which only need to be scaled up to brain sized dimensions.

While it is not possible to disprove such an optimistic view, because of our lack of success in endowing artifacts with human- or animal-like intelligence, many researchers now adopt the view that we must invest more effort in understanding the brain and behaviour as a biological system (O'Reilly, 2006; Webb, 2001). This is a novel development in some disciplines, for example in Connectionism the need for biological realism used to be de-emphasized or even opposed. It has proven difficult to disentangle the neuronal level from the genetic level and the behavioural level from the neuronal level and modelling any higher level cognitive function involves a large number of brain areas, which are interconnected in complicated ways. The human brain will turn out to be extremely difficult to model. Some scientists argue that the way to go is to build brain- or biologically-inspired technological applications (Brooks, 1991; Pfeifer & Scheier, 1999; Webb, 2001), and learn from this how the brain functions. This is a valid approach, and will undoubtedly yield useful insights, but modelling the brain is also crucial to obtain insight into the causes of mental disorders and to improve existing human-brain interfaces systematically.

In this light, the complexity barrier imposed by small research groups and the limited amounts of time are a disaster. In order to make progress, it is necessary to construct models that can be reused by other research groups, which would reduce the enormous level of duplication which is currently going on. Ideally, models should be extensible in two directions: it should be possible to 'detail' them, replacing a certain coarse level of modelling by a finer grained one, and it should be possible to build complex models, using the current ones as building blocks. Not only would such a reuse of modelling greatly reduce duplication of programming efforts and allow the construction of complex models by small research groups, it would also greatly help in the transfer of expertise between the different disciplines involved in brain science.

Technically, it is extremely challenging to create such models, and the question is, what happens when they are there. Who maintains them? Who validates them? Who evaluates their usefulness? It is clear that coordination of some kind is necessary and it seems that with the creation of the International Neuroinformatics Coordinating Facility (INCF; <http://www.incf.org>), this is potentially forthcoming. However, even a coordination agency needs something to start with and it is clear that the first step must come from researchers in the field, since the problems sketched here surface most clearly when one tries to extend models and tries to incorporate work from other researchers. MIIND is an example of such efforts: starting as a relatively simple Artificial Neural Network (ANN) model for object-based attention (van der Velde & de

Kamps, 2001), which was subsequently expanded into the Closed Loop Attention Model (CLAM) (van der Velde, de Kamps & van der Voort van der Kleij, 2004), we extended it to include a neural black-board architecture (van der Velde & de Kamps, 2006). We experienced first, that some of the programming work we had to do was quite repetitive and second, that we could not easily include our earlier work into the later, more sophisticated models. We also found that in these later models, which were characterised by a much more complex spatial organization, it was sometimes necessary to include more realistic descriptions of neuronal dynamics. Furthermore, we found it necessary to be able to integrate our models with those of others, in other words to construct models that are interoperable (see Cannon et al. (2007) for a recent review of interoperability of neural simulators). For example, we are interested to see if the mechanisms described in van der Velde and de Kamps (2001) can be applied to recent models of object recognition in the ventral stream, such as HMAX and more recent variations (Riesenhuber & Poggio, 1999; Serre, Wolf, Bileschi, Riesenhuber, & Poggio, 2007). Finally, we have observed that models of others are often very similar in mathematical structure (Hamker, 2005; Lanyon & Denham, 2004; Usher & Niebur, 1996), even when expressing ideas or models quite different from our own.

We call our ideas on interoperability and extensibility of models "The Philosophy of MIIND" (de Kamps & Baier, 2007). They are realized in C++ code, but can be considered demonstrators, or realizations of design patterns (Gamma, Helm, Johnson, & Vlissides, 1994). Some of our ideas are not necessarily related to a C++ implementation and could, from a user's perspective also be realized on a GRID architecture.

1.2. The Philosophy of MIIND

The 'Philosophy of MIIND' is to isolate repetitive code tasks that occur during modelling as early as possible and to detach the problems that occur as much as possible from their modelling context. An example is `SparseImplementationLib`, which is a library for the representation of sparse random networks. Although initially created as an implementation for ANNs, it contains no explicit references to ANNs and is in fact quite useful in any problem that deals with sparse random networks. The idea is that a library has a limited, well defined task and that if the functionality of the library increases, it should be split.

Consider the example of a sparse network representation: most simulators have dealt with this problem, but their solution is usually not exposed to users and other developers. This is an opportunity for code reuse which is often missed. MIIND explicitly aims to expose its low level functionality. `SparseImplementation` (Section 3) can be used for any problem which involves sparse irregular networks. `DynamicLib` (Section 4.2) is a very generic solver for systems of equations, not necessarily restricted to neuronal simulations. Even if other developers decide not to adopt MIIND's implementation, they may get some ideas from the code. This is a major improvement over the situation where algorithms are published, but where their implementation is not publicly available.

MIIND's core functionality is implemented in C++, an object-oriented language which is suitable for high performance computing. Its object orientation is a major advantage over environments such as MATLAB, which rely on a functional programming paradigm. In our experience data structures in high level cognitive modelling become complicated and lead to a cumbersome implementation in MATLAB (van der Velde & de Kamps, 2001, 2006). The C++ template mechanism allows a static version of duck-typing (Koenig & Moo, 2005) which does not incur run time overheads. This means that central concepts can be expressed in a very abstract and powerful way that still leads to efficient code. In

Fig. 1. An overview of the MIIND libraries.

the population density methods that MIIND provides, this is of crucial importance. Also, a framework for equation solving must be efficient. This has motivated the choice for C++. At the same time the C++ syntax is obscure and complicated. Scripting languages such as Python offer a significant increase in productivity over C++ for code that is not time critical (estimated to be at least a factor 2, depending on the application (Prechelt, 2000)). At the moment we are equipping MIIND with a Python interface. `LayerMappingLib` already has such an interface, and a Python module will be built automatically when MIIND is compiled. The other libraries will receive a Python interface soon. MIIND uses `ROOT` (<http://root.cern.ch>) for visualization. `ROOT` is Open Source software and is developed by CERN to process and visualize the data that will come from the LHC project. It offers scripting in C++ (`CINT`) and Python (`PyRoot`).

MIIND is OpenSource, released on a modified BSD license and hosted on SourceForge (<http://miind.sf.net>). The only modification with respect to the original license is that if you use MIIND in the preparation of a scientific publication, you must cite the ‘currently valid reference’, which is listed on MIIND’s website.

2. An overview of MIIND

Most concepts in MIIND are relatively of high level and therefore not dependent on a particular programming language. Throughout this paper we will illustrate the concepts with simple code fragments of the ‘Hello World’ type. This code should be self-explanatory, even to people who do not have a background in C++. Only in Section 3 are some aspects of the C++ implementation discussed in more detail, but in a way that still should make sense to an audience with a general background in computer science. Throughout the text we will denote *libraries*, *class names* and *class methods* as shown here.

2.1. Sparse networks

An overview of MIIND can be found in Fig. 1. `SparseImplementationLib` is a library which offers generic support for the implementation of sparse random networks. Most biological networks are sparse: the number of nodes is typically large and the resources for the implementation of connections are typically limited. In the brain, for example, the 10^{11} nodes would lead to 10^{22} connections in a fully connected network. In practice, this number is closer to 10^{15} , so that an adjacency matrix representing

the connection structure of a brain would be very sparse indeed. In neural simulations and other biological applications, the networks are also characterised by irregularity: the formation of connections may be driven by random processes or the networks may display little symmetry. Simulations of large networks are often limited by the amount of memory that is available and the adoption of an implementation which makes efficient use of memory is crucial to large network simulations. `SparseImplementationLib` supports the creation of large sparse irregular networks with non-trivial connection structures and provides facilities for reading them from and writing them to disk. Due to the use of C++’s template mechanism, `SparseImplementationLib` is extremely versatile (we elaborate on the relative merits of this mechanism in Section 3) and some of the libraries that build on top of it (`DynamicLib` and `PopulistLib`) are demonstrators of the way in which it can be extended.

Although biological networks are often irregular, artificial models inspired by biology may display a high degree of symmetry. This is true, for example, for the Neocognitron (Fukushima, 1980) and also for more recent models of the ventral stream for object recognition (e.g. Riesenhuber and Poggio (1999) and Serre et al. (2007)). Often, these networks are hierarchical structures of filters which display translation invariance. A literal network implementation of such a filter hierarchy, where the filter operations are implemented by connections, would require a large number of connections. Since the networks are typically feedforward, there are alternative implementations that are more efficient. The effect of a filter-based network may be realized by storing a single filter and by applying that filter repeatedly to all locations in the previous layer. This reduces the large number of connections required to implement the whole parallel feature bank structure to a small set of filters that can be applied in rapid succession (or even in parallel given suitable parallelization). This is the idea behind `LayerMappingLib`.

2.2. Simulating large systems of coupled equations

`DynamicLib`, the generic simulator, builds on top of `SparseImplementationLib`. It models large systems of coupled equations as networks, which is best illustrated by an example: suppose this network is modelled by Wilson–Cowan (Wilson & Cowan, 1972)

