ASL/NSL: A MULTI-LEVEL COMPUTATIONAL MODEL FOR DISTRIBUTED NEURAL SIMULATION*

Alfredo Weitzenfeld
Departmento Académico de Computación
Instituto Tecnológico Autónomo de México
Río Hondo #1, San Angel Tizapán, CP 01000
México DF, MEXICO
alfredo@lamport.rhon.itam.mx

Sebastián Gutiérrez
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
seguti@ics.uci.edu

Keywords: Neurosciences, Distributed processors, Continuous, Differential equation solvers, Hierarchical

ABSTRACT

As neural systems become large and complex, sophisticated tools are needed to support effective model development and efficient simulation processing. Initially, during model development, rich graphical interfaces linked to powerful programming languages and component libraries are the primary requirement. Later, during model simulation, processing efficiency is the primary concern. Workstations and personal computers are quite effective during model development, while parallel and distributed computation become necessary during simulation processing. We give first an overview of modeling and simulation in NSL/ASL together with an example. We then discuss current and future work with the system in the development and simulation of modular neural systems executed in a single computer or distributed computer network.

INTRODUCTION

In this paper we present an overview of the ASL/NSL computational model for the simulation of neural systems. This work is motivated by the quest to simulate animal-like behavior as faithfully as possible based on existing ethological, physiological and anatomical neural data. At the higher level these models are described in terms of *schema* (Arbib 1992) modules corresponding to behavior agents (ethology) simulated with ASL (*Abstract Schema Language*) (Weitzenfeld 1993). At the lower level, neural networks are described in terms of neural modules (physiology and anatomy) simulated with the Neural Simulation Language NSL (Weitzenfeld and Arbib 1994). Many models have been developed and simulated with ASL/NSL (see Weitzenfeld *et al.* 2000a for examples of models involving perception, visuomotor coordination, motor control as well as technological neural applications).

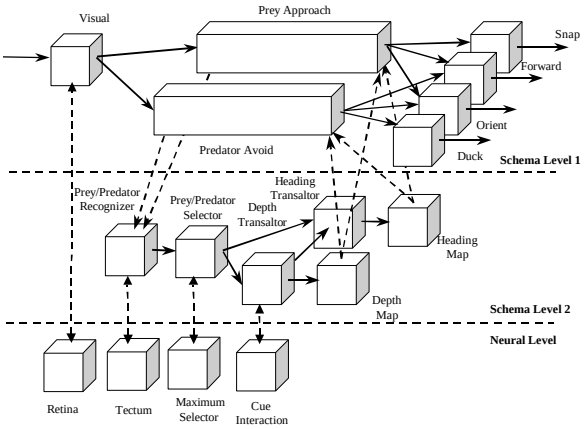

Figure 1. Schema hierarchy for the toad's prey acquisition and predator avoidance models. The top two levels correspond to schema levels (1 and 2) and the lower level corresponds to neural modules.

For example, in Figure 1 we show a simplified diagram of the toad's prey acquisitions and prey avoidance model (Cobas and Arbib 1992), (schema level 1) involving *perceptual* and *motor* schemas. The external visual input is processed to generate

* We thank the NSF-CONACyT collaboration grant (#IRI-9522999 in the US and #546500-5-C018-A in Mexico), the CONACyT REDII grant in Mexico, as well as the "Asociación Mexicana de Cultura, A.C.".

appropriate motor actions: forward, orient, snap and duck. Schemas at this level are decomposed and delegated to the next level down (schema level 2) where schemas perform more specific tasks. Prey approach and predator avoid schemas delegate their tasks to a *schema assemblage* composed of a prey/predator recognizer, a prey/predator selector, depth and heading translators and maps. Next level down, different neural modules: *Retina, Tectum, Maximum Selector* and *Cue Interaction*, if available, implement the actual neural network processing.

SCHEMAS AND NEURAL NETWORKS

To better understand the underlying computational model, ASL/NSL defines a tree-like schema or module hierarchy as shown in Figure 2.

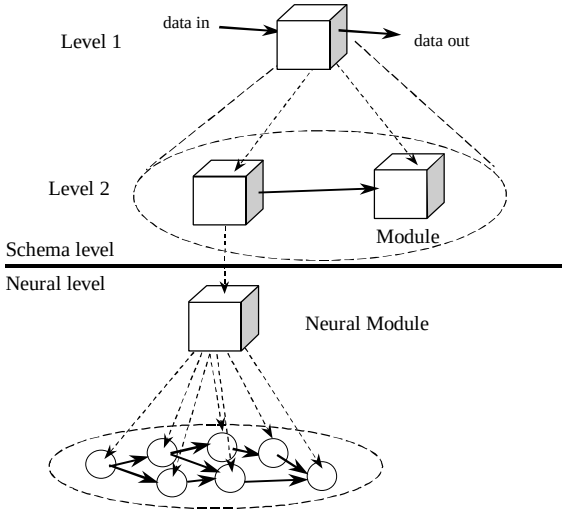

Figure 2. The ASL/NSL computational model is based on hierarchical interconnected modules. A schema module at a higher level (level 1) is decomposed (dashed lines) into additional interconnected (solid arrow) schema submodules (level 2). At the lowest level, neural modules are implemented by neural networks (circular objects).

Starting by the root module (known as the model), modules are further decomposed into additional *submodules*, having no limit on how many levels this may reach. At the same abstraction level, modules are interconnected (solid arrows), while at different levels modules have their task delegated (dashed arrows). Networks of submodules – *module assemblages* – are seen in their entirety in terms of a single higher-level module and may be implemented independently from each other in both top-down and bottom-up fashion, an important benefit of modular design. At the higher abstraction levels, the detailed module implementation is left unspecified, only specifying the module's interface and what is to be achieved. At the lowest level, schemas are implemented by neural modules.

As a computational unit, every module incorporates its own local structure and control mechanisms. Every module defines an external interface made of a set of unidirectional input and output *ports* supporting data passing between modules together with a set of public methods that can be externally invoked from other modules. Communication between modules is in the form of asynchronous message passing for both data and methods. Internally, communication is hierarchically managed through anonymous data port reading and writing. Externally, communication is managed through dynamic port *connections* (solid arrows) - links between output ports in one module to input ports in another module - and *relabelings* (dashed arrows) - module ports at one level in the hierarchy are linked to similar input or output ports at a different level. The hierarchical port management approach enables the development of neural architectures where modules may be designed and implemented independently and without prior knowledge of the complete model or their final execution environment, encouraging component reusability and permitting module execution in a distributed fashion (Weitzenfeld *et al*. 2000b).

For example, consider the *depth perception* problem where a three dimensional scene is presented to the two eyes. The depth perception model developed by House (1989) uses two systems to build a depth map, one driven by *disparity* cues - difference in retina projection - while the other is driven by *accommodation* cues - receiving information about focal length. The corresponding simplified ASL/NSL model consists of a *Stereo s* root module and three interconnected submodules: *Retina r, Depth m* (accommodation) and *s* (disparity), as shown in Figure 3.

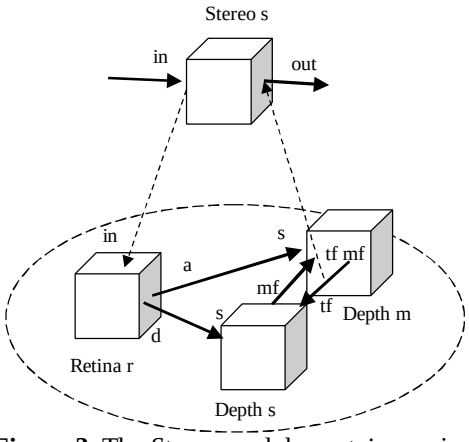

Figure 3. The *Stereo* module contains an *in* input port and an *out* output port. It is further decomposed into a *Retina* module containing an input port *in* and two output ports, *d* and *a*, for disparity and accommodation, respectively. The *Depth* module consists of an input port *s*, receiving data from the *Retina*, a second input port *tf*, receiving input from the other *Depth* module, and an output port *mf*.

NEURAL SIMULATION

Neural networks define the lowest level in the computational model. In general, neural simulation varies depending on whether it relates to artificial o biological systems. Artificial neural networks often require two-stage simulation process, (1) an initial training phase and (2) a subsequent processing or running phase. On the other hand, biological neural networks usually require a single running phase, in which behavior and learning may be intertwined. Initially, during the model development, the simulation process involves interactively specifying aspects of the model, such as network input and parameters values, as well as simulation control and visualization. As the model becomes more stable, efficiency becomes the primary concern of the simulation process and concurrency play an important role, not only as a way to increase processing performance but also to model neurons more faithfully (Weitzenfeld and Arbib 1991). Yet, large and complex models may require hours or even days of processing time and distributed computing may play a key role in speeding up computation.

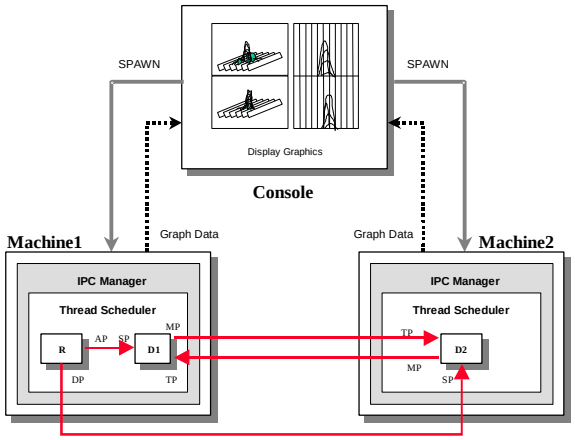

Figure 4. The distributed architecture of ASL/NSL is conceptualized as a set of autonomous (processing) nodes. Each node may contain one or more schema modules that communicate via asynchronous message passing. The nodes do not share a common address space and they are connected by unidirectional communication links along which messages are transmitted. A communication link is not necessarily a direct link, It can be any network path connecting these schemas

Sequential and Distributed Processing

ASL/NSL provides an architecture to support both sequential and distributed (as well as parallel) simulations. In the sequential simulation, neural dynamics are executed one equation at the time where the output from one neuron is immediately fed as input to the next one. In NSL the same procedure is applied to modules, where each module would get executed in a predetermined order sending newly generated output port values immediately to their corresponding interconnected input ports. This approach simplifies simulation results in that the order of computation depends directly from the order of computation. To further simulate brain-like concurrent behavior while doing sequential processing, NSL interleaves module execution while buffering output from one module to the next one, making processing order unimportant. In the distributed simulation, all simulation control as well as graphical display take place in the *console* in one of the machines responsible for distributing processing among local and/or remote machines (see

Figure 4). The actual distribution is specified by the user or automatically generated by the system depending on the available machines in the network.

Since neural models are processing and communication intensive, a distributed simulation requires monitoring and resource management capabilities in order to improve overall simulation performance. Particularly, we are interested in (1) identify processing bottlenecks, such as unnecessary serializations between schemas or excessive model fragmentation, (2) accomplish a cost-effective use of the resources in the system and (3) optimize communication performance.

Reflective meta-level architectures add introspective capabilities to computational systems by providing dynamic adaptability to internal and external processing constraints (Smith 1982). Here, the computation activities are separated into base and meta level activities, representing the application execution and the structures used to control its execution respectively. This separation of concerns allow us to monitor and reason about aspects of communication and resource management without affecting the normal behavior of the underlying application execution. In fact, meta-level facilities allows to provide more adequate control of modular neural networks (Smieja and Muhlenbein 1992), as well as improve the computational efficiency of the network (Boers and Kuiper 1993) in a semantically clean manner.

Meta-Level Architecture

In order to represent the resource and communication activities going on behind the scenes without interfere with the on-going computation (called also perturbation), the meta-level architecture is design as follows: (1) Each schema module has a meta-schema, which represents the meta-level capabilities of the base-level schema module, (2) The communication between the schema and its meta-schema is synchronous, in order to guarantee the integrity and semantics of the neural simulation that could be violated by some interleaving execution between the meta and base level, (3) Each meta-schema is composed by a computational and communication model, (4) The computational model gathers resource information of the schema it represents, (5) The communication model monitors the incoming/outgoing communication of its schema to identify possibly communication bottlenecks and evaluate different distribution strategies of the neural model through postmortem visualization of the communication primitives.

The computational model gathers the information by accessing the internal structures of the kernel in a modular way. Thus, it may be visualized as an plug-in component tailored to an specific architecture, which traces the following information for a particular schema module: (1) The node workload, (2) The statistic information of the node, such as number of processes, memory and virtual memory available and (3) performance statistics of the schema module such as real cpu time (cpu-ticks), amount of memory allocated, number of threads, priority assigned and total size of the schema. On the other side, the communication model uses a client-server logging event model for communication primitives, in order to minimize the degree of perturbation, while measure: (1) Ports communication throughput, (2) Message queue size, (3) Number of messages remaining in the queue and (4) Message properties, such as size of the message, type of the message, who is the sender, who is the receiver.

The client-server logging event model works similar to PGPVM (Brad Topol *et al.* 1994), but without synchronization barriers. That is, there is a centralized server which communicates with the underlying network in order to satisfy communication service requirements while providing distributed buffered trace in PICL (*Portable Instrumented Communication Library*) format (Geist *et al.* 1991) as follows (see Figure 5):

1. The message send request is intercepted by the meta-schema
2. The message properties are registered and a log event for the transmission is generated and saved in a local buffer
3. The transmission request is sent to the communication server (eSvr), which in turn sent it to the underlying network
4. The underlying network sent the message and return and acknowledge to eSvr
5. eSvr return the control to the meta-schema, which resume the base level application

6. At the end of the simulation, each meta-schema sent the collected information to eSvr
7. eSvr consolidate the information and save it in the trace database
8. A post processing of the information is required in order to create a trace file, which will be the input of the postmortem visualization program ParaGraph (Heath and Etheridge 1991)

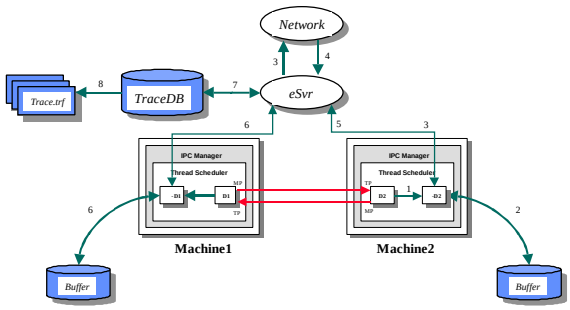

Figure 5. A simple distributed neural model composed of two schema modules **D1** and **D2**, in which communication is restricted to their port interconnections

SIMULATION RESULTS

In this section we present the results of the depth perception distributed simulation, realized with the parameters shown in Table 1.

System endTime	*System delta*	*Differential delta*	*Differential integration*
2.0	0.05	0.05	Euler

Table 1. Simulation and approximation method parameters.

Model	*CPU (MHz)*	*Disk (GB)*	*RAM (MB)*	*Operating System*
Ultra-1	200	1	32	Solaris 2.6
Ultra-10	333	1	128	Solaris 2.6
Sparc-5	170	0.5	32	Solaris 2.6

Table 2. Hardware and operating system parameters.

The underlying network infrastructure is a shared Ethernet 10 baseT and the hardware specifications shown in Table 2.

One of the most interesting parts of the simulation was the frequency and volume of communication. As we can see in Figure 6, the communication between *Depth1* and *NslConsole* (ranked node 0) or *Retina* (node 1) is frequency intensive, while communication between *Depth1* and *Depth2* (node 3) is volume intensive

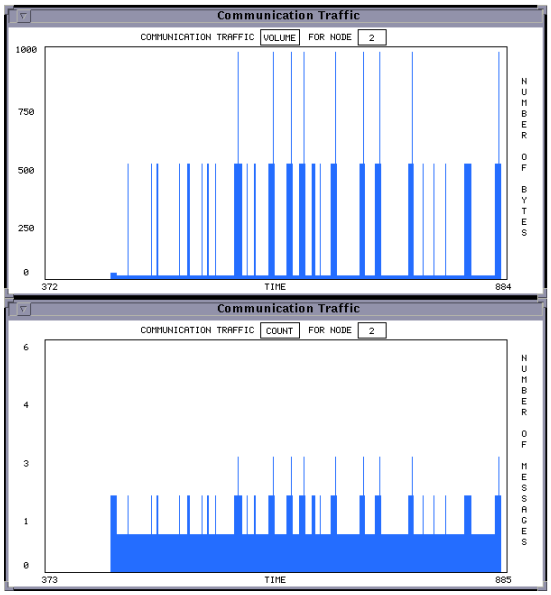

Figure 6. Communication frequency and volume of *Depth1*

The *lamport* diagram of space-time, mainly knowledge as the happens before relation, shown in Figure 7 displays the order of the events during the simulation

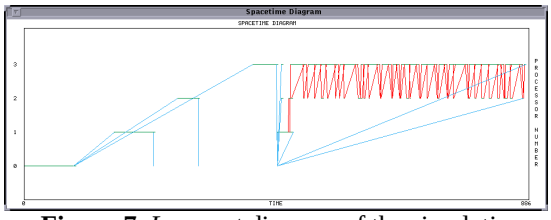

Figure 7. *Lamport* diagram of the simulation with node 0 as the Nsl Console, node 1 as *Retina* and nodes 2 and 3 as *Depth1* and *Depth2* respectively.

REFERENCES

Arbib, M.A., 1992, "Schema Theory", *Encyclopedia of Artificial Intelligence*, 2nd Edition, Editor Stuart Shapiro, 2:1427-1443, Wiley.

Boers, E.J., Kuiper, H., 1993, "Biological Metaphors and the Design of Modular Artificial Neural

Networks", Master Thesis, Dept. of Computer Science and Experimental and Theoretical Psychology, Leiden University, Netherlands.
Cobas, A., and Arbib, M.A., 1992, "Prey-catching and Predator-avoidance in Frog and Toad: Defining the Schemas", *J. Theor. Biol* 157, 271-304.
Geist, G., Heath, M., Peyton, B., and Woley, P, 1991, "A User's Guide to PICL", Technical Report ORNL/TM-11616, Oak Ridge National Lab.
Heath, M., and Etheridge, J., 1991, "Visualizing the Performance of Parallel Programs", *IEEE Software*, 8(5):29-39.
House, D., 1989, "Depth perception in frogs and toads: A study of in neural computing", *Lecture notes in Biomathematics*, 80, Springer-Verlag.
Smieja, F.J., Muhlenbein, H., 1992, "Reflective Modular Neural Network Systems", Technical Report, German National Research Center for Computer Science, May.
Smith B, 1982, "Reflection and Semantics in a Procedural Languages", Technical Report 272, Laboratory of Computer Science, Massachusetts Institute of Technology.
Topol, B., Sunderam, V., and Alund, A., 1994, *PGPVM Performance Visualization Support for PVM*, Technical Report CSTR-940801, Emory University.
Weitzenfeld, A., 1993, "ASL: Hierarchy, Composition, Heterogeneity, and Multi-Granularity in Concurrent Object-Oriented Programming", *Proceedings of the Workshop on Neural Architectures and Distributed AI: From Schema Assemblages to Neural Networks*, USC, October 19-20.
Weitzenfeld A & Arbib M, 1991, "Concurrent Object Oriented Framework for the Simulation of Neural Networks", *Proceedings of ECOOP/OOPSLA'90 Workshop on Object-Based Concurrent Programming*, *OOPS Messenger*, 2(2):120-124.
Weitzenfeld, A., Arbib, M.A., 1994, "NSL, Neural Simulation Language", *Neural Networks Simulation Environments*, Editor J. Skrzypek, Kluwer.
Weitzenfeld, A., Arbib, M., Alexander, A., 2000a, *NSL - Neural Simulation Language: System and Applications*, MIT Press (to be published).
Weitzenfeld, A., Peguero, O., Gutierrez, S., 2000b, "NSL/ASL: Distributed Simulation of Modular Neural Networks", *Proc of MICAI 2000*, Acapulco, Mexico.

BIOGRAPHY

Alfredo Weitzenfeld is currently an Assistant Professor at the Computer Engineering Department at the Autonomous Technological Institute of Mexico (ITAM). He received his BS in Electrical Engineering from Israel Institute of Technology (TECHNION). He has an MS in Computer Engineering and a PhD in Computer Science both from the University of Southern California, where he was a Research Assistant Professor. In Mexico he is the co-founder of CANNES Laboratory for Brain Simulation and a member of the Mexican National Research System (SNI) as well as a member of the Doctoral Advising Committee at the National Autonomous University of Mexico (UNAM). He is currently supported by Mexico´s funding agency CONACyT. Previously he has received grants from NSF-CONACyT collaboration projects with a number of US academic institutions. His main research interests include simulation systems, software engineering, artificial intelligence, neural networks and robotics.

Sebastian Gutiérrez is currently pursuing a Ph.D. from the University of California, Irvine. He received his BS in Computer Engineering from the Autonomous Technological Institute of Mexico (ITAM). He worked as Research Assistant at CANNES Laboratory for Brain Simulation. His main research interests include semantic foundation for composition of communication services in open distributed systems, neural networks and clustering for scientific computing.