Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Salles Chevitarese

Neuronal circuit specification language and tools for modelling the virtual fly brain

TESE DE DOUTORADO

Thesis presented to the Programa de Pós-graduação em Engenharia Elétrica of the Departamento de Engenharia Elétrica, PUC-Rio as partial fullfilment of the requirement for the degree of Doutor em Engenharia Elétrica.

Advisor : Prof. Marley Maria Bernardes Rebuzzi Vellasco

Co–Advisor: Prof. Dilza de Mattos Szwarcman

Rio de Janeiro
February 2015.

Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Salles Chevitarese

Neuronal circuit specification language and tools for modelling the virtual fly brain

Thesis presented to the Programa de Pós-graduação em Engenharia Elétrica of the Departamento de Engenharia Elétrica, PUC-Rio as partial fullfilment of the requirement for the degree of Doutor em Engenharia Elétrica. Approved by the following commission:

Prof. Marley Maria Bernardes Rebuzzi Vellasco
Advisor
Departamento de Engenharia Elétrica — PUC–Rio

Prof. Dilza de Mattos Szwarcman
Co–Advisor
Departamento de Engenharia Elétrica — PUC–Rio

Prof. Leonardo Alfredo Forero Mendoza
PUC-Rio

Prof. Noemi de La Rocque Rodriguez
PUC-Rio

Prof. Alexandre Loureiro Madureira
LNCC

Prof. Antônio Pereira Júnior
UFRN

Prof. Karla Tereza Figueiredo Leite
UEZO

Prof. José Eugenio Leal
Coordenador do Centro Técnico Científico da PUC–Rio

Rio de Janeiro — February 27th, 2015.

Daniel Salles Chevitarese
graduated in Computating Engineering at PUC-Rio and finished his MA in Electrical Engineering at the same institution in 2010. Since then I am a PhD student at PUC-Rio and a Visiting Scholar at Bionet Group in Electrical Engineering Department of Columbia University. In addition, I am working as consultant in some research and development projects of ICA in Electrical Engineering Department of PUC-Rio. I have experience in modeling and development of decision support systems / business intelligence, working mainly in Research and Development projects in energy and energy distribution areas.

Bibliographic data

Salles Chevitarese, Daniel

Neuronal circuit specification language and tools for modelling the virtual fly brain / Daniel Salles Chevitarese; advisor: Marley Maria Bernardes Rebuzzi Vellasco; co–advisor: Dilza de Mattos Szwarcman. — Rio de Janeiro : PUC–Rio, Departamento de Engenharia Elétrica, 2015..

v., 110 f: il. ; 29,7 cm

1. Tese de Doutorado - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica.

Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Neurociência. 3. Drosophila Melanogaster. 4. Modelagem. 5. XML. 6. Padronização. 7. API. 8. Python. 9. CUDA. 10. NeuroML. I. Bernardes Rebuzzi Vellasco, Marley Maria. II. de Mattos Szwarcman, Dilza. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. IV. Título.

CDD: 510

Acknowledgments

To God.

To my advisers Marley Maria Vellasco and Dilza Szwarcman for the support, the everyday kindness and the incentive for the realisation of this work.

To CAPES and PUC–Rio, for the financial support, without which this work would not have been realized.

To my family, who suffered the most of my expatriation. To my daughter who endured months without contact with her father.

To my colleagues of the PUC–Rio and Columbia University, who have me loved both places.

To professor Aurel A. Lazar who offered me the opportunity of this cooperation.

To the people of the Electrical Engineering department for the constant help.

Abstract

Salles Chevitarese, Daniel; Bernardes Rebuzzi Vellasco, Marley Maria; de Mattos Szwarcman, Dilza. **Neuronal circuit specification language and tools for modelling the virtual fly brain**. Rio de Janeiro, 2015.. 110p. PhD Thesis — Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

The brain of the fruit fly Drosophila Melanogaster is an attractive model system for studying the logic of neural circuit function because it implements complex sensory-driven behavior with a nervous system comprising a number of neural components that is five orders of magnitude smaller than that of vertebrates. Analysis of the fly's connectome, or neural connectivity map, using the extensive toolbox of genetic manipulation techniques developed for Drosophila, has revealed that its brain comprises about 40 distinct modular subdivisions called Local Processing Units (LPUs), each of which is characterized by a unique internal information processing circuitry. LPUs can be regarded as the functional building blocks of the fly, since almost all identified LPUs have been found to correspond to anatomical regions of the fly brain associated with specific functional subsystems such as sensation and locomotion. We can therefore emulate the entire fly brain by integrating its constituent LPUs. Although our knowledge of the internal circuitry of many LPUs is far from complete, analyses of those LPUs comprised by the fly's olfactory and vision systems suggest the existence of repeated canonical sub-circuits that are integral to the information processing functions provided by each LPU. The development of plausible LPU models therefore requires the ability to specify and instantiate sub-circuits without explicit reference to their constituent neurons and internal connections. To this end, this work presents a framework to model and specify the circuit of the brain, providing a neural circuit specification language called CircuitML, a Python API to better handler CircuitML files and an optimized connector to neurokernel for the simulation of those LPUs on GPU. CircuitML has been designed as an extension to NeuroML (NML), which is an XML-based neural model description language that provides constructs for defining sub-circuits that comprise neural primitives. Sub-circuits are endowed with interface ports that enable their connection to other sub-circuits via neural connectivity patterns.

Keywords

Neuroscience; Drosophila Melanogaster; Model Specification; XML; Standardization; API; Python; CUDA; NeuroML;

Resumo

Salles Chevitarese, Daniel; Bernardes Rebuzzi Vellasco, Marley Maria; de Mattos Szwarcman, Dilza. **Linguagem de Especificação de Circuito Neuronal e Ferramentas para Modelagem do Cérebro Virtual da Mosca da Fruta**. Rio de Janeiro, 2015.. 110p. Tese de Doutorado — Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

O cérebro da Drosophila é um sistema atrativo para o estudo da lógica do circuito neural, porque implementa o comportamento sensorial complexo com um sistema nervoso que compreende um número de componentes neurais que é de cinco ordens de grandeza menor do que o de vertebrados. A análise do conectoma da mosca, revelou que o seu cérebro compreende cerca de 40 subdivisões distintas chamadas unidades de processamento local (LPUs), cada uma das quais é caracterizada por circuitos de processamento únicos. As LPUs podem ser consideradas os blocos de construção funcionais da cérebro, uma vez que quase todas LPUs identificadas correspondem a regiões anatómicas do cérebro associadas com subsistemas funcionais específicos tais como a sensação e locomoção. Podemos, portanto, emular todo o cérebro da mosca, integrando suas LPUs constituintes. Embora o nosso conhecimento do circuito interno de muitas LPUs está longe de ser completa, análises dessas LPUs compostas pelos sistemas olfativos e visuais da mosca sugerem a existência de repetidos sub-circuitos que são essenciais para as funções de processamento de informações fornecidas por cada LPU. O desenvolvimento de modelos LPU plausíveis, portanto, requer a habilidade de especificar e instanciar sub-circuitos, sem referência explícita a seus neurônios constituintes ou ligações internas. Para este fim, este trabalho apresenta um arcabouço para modelar e especificar circuitos do cérebro, proporcionando uma linguagem de especificação neural chamada CircuitML, uma API Python para melhor manipular arquivos CircuitML e um conector otimizado para neurokernel para a simulação desses LPUs em GPU. A CircuitML foi concebida como uma extensão para NeuroML (NML), que é uma linguagem para de descrição de redes neurais biológicas baseada em XML que fornece primitivas para a definição de sub-circuitos neurais. Sub-circuitos são dotados de portas de interface que permitem a sua ligação a outros sub-circuitos através de padrões de conectividade neural.

Palavras–chave

Neurociência; Drosophila Melanogaster; Modelagem; XML; Padronização; API; Python; CUDA; NeuroML;

Contents

List of Figures

List of Tables

Pequenos passos podem não fazer muita diferença numa jornada curta, mas na longa jornada da vida são capazes de colocar você num lugar completamente diferente.

James Hunter, *The servant.*

1
Introduction

The brain is the most amazing system that men are aware of, and understanding this incredible machine is one of the most challenging and fascinating tasks the mankind has ever had. Some say that it is not even feasible, but there are many efforts in different fields, with different techniques and subjects, to accomplish this task. For example, with the convergence of modern cognitive psychology and the brain sciences, scientists started to appreciate that all mental functions are divisible into sub functions (KANDEL et al., 2000). However, in a system of the size of a human brain - actually, any mammal brain - it is almost impossible to demonstrate which components of a mental operation are represented by a particular pathway or brain region. One difficulty is that our cognitive experience consists of instantaneous, smooth operations (KANDEL et al., 2000), each of which is composed of many independent components, where every task requires coordination of several distinct brain areas.

One strategy that may help scientists and has been investigated since last century, is to create a simpler model, for example a model of an insect brain, specifically of the fruit fly Drosophila Melanogaster that is widely used in laboratory research (ARMSTRONG et al., 2009). Such organism started to be studied in 1915 in a review by Sturtevant (STURTEVANT, 1915). Since then, the research of the fruit fly transformed this tiny insect into one of the most powerful genetic model organisms (ARMSTRONG et al., 2009), which uncovers many developmental principles, genetic regulation and cell signaling. One example of genetic techniques for manipulation of the fly's neural circuitry is the GAL4 driver system (DUFFY, 2002; RISTER et al., 2007; SONG et al., 2012; WARDILL et al., 2012; MAISAK et al., 2013). Principles, such as developmental processes, genetic regulation and cell signaling, are conserved across species and can explain some inherited diseases (ARMSTRONG et al., 2009).

In addition to the genetic toolkit, recent advances in experimental methods for precise recordings of the fly's neuronal responses to stimuli (KIM; LAZAR; YEVGENIY, 2010; KIM; LAZAR; SLUTSKIY, 2010; KIM; LAZAR;

SLUTSKIY, 2011a; KIM; LAZAR; SLUTSKIY, 2011b; WILSON, 2011), techniques for analyzing the fly's behavioral responses to stimuli (BUDICK; DICKINSON, 2006; KATSOV; CLANDININ, 2008; MAIMON; STRAW; DICKINSON, 2008; CHIAPPE et al., 2010), and progress in reconstruction of the fly connectome (CHKLOVSKII; VITALADEVUNI; SCHEFFER, 2010; TAKEMURA et al., 2013) by using identified neurons, which are stereotyped neurons that can be located in every fly (OLSEN; WILSON, 2008), facilitates shaping circuits, as they can be easily recognized among flies. One advantage on the engineering perspective is the reasonable compromise between tractability and richness, since flies have an interesting behavioral repertoire and possess, approximately, 150,000 neurons (CHIANG et al., 2011), which means five orders of magnitude smaller than that of vertebrates.

Although the fly does represent a very interesting model for studying, the limitations of this organism are important to mention. Besides its very complete and well-studied genetic toolkit, which can be defined as a set of methodologies, processes and tools that have the ability to modify the animal behavior by changing its genes, the small size of the brain makes electrophysiology in a live subject a challenging task. In addition, there is no clear answer to whether the fly is a good model for studying how neural activity sculpts neural circuits (OLSEN; WILSON, 2008) or if the "hard-wired" nature of the drosophila brain would limit our ability to generalize to bigger systems (OLSEN; WILSON, 2008). In the end, pondering the pros and cons, the knowledge about the fly genes, the sharing of biological principles among species and a balance between tractability and richness, lead many scientists to conclude that the brain of the fruit fly can be considered an attractive model system for studying the logic of neural circuit function.

Analyses of the fly's connectome (SPORNS; GIULIO; ROLF, 2005), which is the map of all neural elements and connections between them, have revealed that its brain comprises about 40 distinct modular subdivisions called Local Processing Units (LPUs) (CHIANG et al., 2011), each of which can be regarded as a functional building block of the fly brain. Almost all identified LPUs have been found to correspond to anatomical regions of the fly brain associated with specific functional subsystems, for example, sensation and locomotion. Such modularity makes easier the virtualization of the fly brain, proposed by Armstrong (ARMSTRONG et al., 2009), that may help scientists to better understand the animal brain. However, the first step on this virtualization process is to collect, save and share data.

Allowing scientists, from different fields and areas, to save and share data collected from electrophysiology, imaging or any other method that collects

neural information, is very hard, since they have different levels of familiarity with computer tools and programing languages. Also, the availability of so many tools and formats to save neural data makes information sharing painful, since, in order to do so, scientists must convert from one format to another. NeuroML project (GLEESON et al., 2010), which will be presented in more details on the next chapter, is the most prominent effort to standardize neural data among scientists. NeuroML is a meta-language, based on XML (Extensible Markup Language), that is very simple and easy to use, but very powerful by allowing detailed models and their components to be defined in a standalone form to be used across multiple simulators and to be archived in a standardized format (GLEESON et al., 2012). Although NeuroML already offers many advantages, it was intended to address the variety of neurological systems in a general manner. In addition, entities are grouped according to biological characteristics only based on biology, i.e., by chemical elements, by cell types, etc. It is missing a way to organize this massive data by its functionality.

1.1. Objectives

In order to arrange entities by their functionality, this work presents a layer over NeuroML that changes the perspective on brain specification, where neural networks are organized into LPUs, each of which is responsible for one particular task. Then, by combining LPUs, it is possible to perform the entire behavioral repertoire of one specie.

The main objective of this work is, therefore, to provide a methodology and a toolkit for specifying and simulating neural circuits and sub-circuits in a structured fashion, where the circuit functionality can be highlighted from the biological circuit.

1.2. Contributions

This work presents a toolkit that inherits all benefits from NeuroML and extends its functionality by providing a circuit perspective in a modular framework, where each module or block, is a functional entity. Since the brain is a big entangled system, where all behavior comes from serial and parallel processes (KANDEL et al., 2000), such approach may allow scientists to expose small circuits in the brain that have a specific functionality i.e., perform a specific process, and use those circuits repeatedly inside other processes, or together with them. The main contributions of this work are:

- proposal of CircuitML, a declarative language, which provides a circuit perspective on the brain modeling;
- proposal of libCircuitML, a Python API that can be imported into a Python script allowing:
  - to load and validate XML files in their respective formats;
  - to parse and to edit circuit models, which closely follows the structure of the XML language;
  - to save valid XML files either on NeuroML format or on CircuitML format;
  - to use additional functionality, such as Python object model and new components (LPU, subcircuit, etc.) that makes easier to create large models;
- the specification of the following fly sensory systems using CircuitML, demonstrating how new components make possible to create more complex neural networks:
  - the visual system;
  - the olfactory system.

1.3. Work organization

The work comprises four additional chapters, as described below.

Chapter 2 reviews the theoretical background needed to follow the entire work. Some principles in computational neuroscience, such as the abstractions commonly used, neuron and synapse models and the stimulus representation, are shortly explained. In addition, the state of the art in Neuroinformatics is discussed and the existing literature is linked with the current work.

In Chapter 3, the framework proposed is presented in detail, where the toolkit comprising the CircuitML specification language, the CircuitML interpreter and the GPU emulator are described.

Chapter 4 presents the simulation results of vision, olfactory and auditory systems, including circuit diagrams and specifications. By comparing the specification of all three systems it is possible to notice how CircuitML can facilitate such task, and how flexible the framework can be.

Finally, in Chapter 5, conclusions are discussed, in addition to the future directions.

2
Theoretical Background

This chapter presents all necessary background to follow the work proposed in Chapter 3. It also introduces the neural science field as it is crucial to understand the importance of this work and how it facilitates the modeling of neural systems. This chapter is divided into 4 main parts: (1) introduction to neuroscience, (2) modelling neurons and synapses (page 23), (3) The connectome and the local processing units (page 28), and (4) Existing computational tools (page 32).

2.1. Introduction to neuroscience

The goal of neural science is to understand the brain and the mind running on it; how we perceive, move, think, and remember. Unlike the short and simple answer, understanding the mind comprises the answer of a huge number of questions, such as: How does the brain develop? How do nerve cells in the brain communicate with one another? How do different patterns of interconnections give rise to different perceptions and motor acts? How is communication between neurons modified by experience? How is that communication altered by diseases? (KANDEL et al., 2000).

During the last two decades, a lot of effort was made towards identifying the link between genes and cells functionality. Such effort resulted in a general plan that provides a common conceptual framework for all cell biology, including neurobiology. After the uncovering of this link, the current challenge is to understand which genes affect what specific behavior. The given approach depends on the view that all behavior is the result of brain function, which is commonly referred as "the mind is a set of operations carried out by the brain" (KANDEL et al., 2000). The result of brain actions range from relatively simple motor behaviors, such as walking or eating, to more complex cognitive actions, such as thinking, speaking or creating.

To prove that all behavior raises from brain operations, it is necessary to verify if particular mental processes are restrained to specific regions of the brain, or if the mind represents a collective and emergent property of the

whole brain. Also, if specific mental processes can be localized at certain brain regions, what is the relationship between the anatomy and physiology of one region and its specific function in perception, thought, or movement? Finally, it is important to know if such relationship, between anatomy and function, is more likely to be revealed by examining the region as a whole or by studying its individual nerve cells. However, regardless the answer for this question, in order to answer it, its crucial study the brain in parts, given its size and its complexity. In the next sections, the basic aspects of neural processing is presented: the processing itself, the perception and the development.

2.1.1. Mental Processes and the Elementary Processing Operations

The evidence of a brain as a multi-block system, where each block is responsible for a particular function, has been rejected in the past, because it was believed that the cerebral cortex consisted of many independent organs, each dedicated to a complete and distinct mental function. In the aftermath of Wernicke's discovery (WERNICKE, 1969), he proposes that there is a modular organization for language in the brain consisting of a complex of serial and parallel processing centers with more or less independent functions. Now it is appreciated that all cognitive abilities result from the interaction of many simple processing mechanisms distributed in many different regions of the brain (KANDEL et al., 2000). Specific brain regions are not concerned with faculties of the mind, but with elementary processing operations. Perception, movement, language, thought, and memory are all made possible by the serial and parallel interlinking of several brain regions, each with specific functions. As a result, damage to a single area need not result in the loss of an entire faculty as many earlier neurologists predicted. Even if a behavior initially disappears, it may partially return as undamaged parts of the brain reorganize their linkages.

Identifying and localizing specific functions in the brain is very hard, because it is enormously difficult to demonstrate which components of a mental operation are represented by a particular pathway or brain region. It requires analyses of mental operations in a cellular level, which is not feasible even today. The idea of splitting functions into sub-functions seems to be one good attempt. However, breaking down mental processes into analytical categories or steps is also challenging since our cognitive experience consists of instantaneous, smooth operations. Actually, these processes are composed of numerous independent information-processing components, and even the simplest task requires coordination of several distinct brain areas.

2.1.2. Perception - the input of the brain

Every brain perceives its environment through receptor cells that are sensitive to one or another kind of stimuli. The philosophy called Positivism (MACIONIS; CLARKE; GERBER, 1994) states that sensation and perception are the origin of all our knowledge, which is obtained by sensory experience. Although nowadays it is known that a newborn's mind is not empty and, even a grown brain extracts only an amount of information from each stimulus, while ignoring others and interpreting such data in the context of the brain's previous experience, it is possible to realize how much of our knowledge comes from our senses.

For neuroscientists, perception, studied in *sensory physiology* and *psychophysics*, was the starting point to study mental processes, since it is a natural input for the neural system. Animal species share three common steps (KANDEL et al., 2000), which are different from each other:

1. **Physical Stimulus**, which is the stimulus itself - the input signal, e.g.: the light that activates photoreceptors in the retina or the sound wave that makes the parts of the auditory system vibrate;

2. **Process** (or a set of processes) that transforms the stimulus into nerve impulses;

3. **Response** to this signal in the form of a perception or conscious experience of sensation.

2.1.3. Brain Development

The development of the nervous system is influenced by internal and external factors. Internally, such development depends on the expression of particular genes at particular places and times during the process. Both spatial and temporal patterns are regulated by hard-wired molecular programs and epigenetic processes (KANDEL et al., 2000). Externally, social experience, sensory stimuli and nutrients are the influencing elements. Many advances in defining the mechanisms that control development of the nervous system have emerged from studies on molecular biology and from the analysis of simpler systems, such as the fruit fly *Drosophila* and the nematode worm *Caenhorabitis elegans*. Actually, most of the key molecules that control the formation of the nervous system are found in most species, a fact that reinforces

the expectation that simpler brains can teach scientists about circuits in a bigger brain (ARMSTRONG et al., 2009).

Other than the development process of the brain, in order to understand perception, communication and information processing, a common language is required to describe efficiently the building blocks of a neural system, in addition to characterize the structural foundation that underlies input-output electrophysiology. In addition, a methodology different from that of Biology, which classify elements either by behavioral observation or by changing it, may be necessary.

Up to this point, this chapter has presented many aspects from the biological side, showing what are the questions that drive neural scientists, how the brain is divided into its morphology, what are the main functions of each part and a general idea on how the brain perceives and computes information. From now on, this chapter will now focus on the engineering perspective, i.e., the Computational Neuroscience field, where scientists are working on mathematical models that imitate the actual brain.

2.1.4. Computational Neuroscience

Computational Neuroscience is the study of brain circuits using mathematical models. It trails the goal of neural science, as discussed in the beginning of this chapter, i.e., to understand the principles and the mechanisms behind development, perception, communication and information processing. In the end, the objective is to develop tools that allow scientists to answer how information is represented in the brain, and how this representation is processed. It is important to notice that Computational Neuroscience field is different from psychological connectionism and from learning theories of disciplines such as machine learning, neural networks and computational learning theory, since the field emphasizes descriptions of functional and biologically realistic neurons and their physiology and dynamics.

The term "computational neuroscience" was introduced by Eric L. Schwartz, who organized a conference, held in 1985 in Carmel, California, at the request of the Systems Development Foundation, to provide a summary of the current status of a field which until that point was referred to by a variety of names, such as neural modeling, brain theory and neural networks. The proceedings of this definitional meeting were published in 1990s as the book Computational Neuroscience (SCHWARTZ, 1993).

The field of computational neuroscience can be roughly categorized into several lines of inquiry. Most computational neuroscientists collaborate closely

with experimentalists in analyzing novel data and synthesizing new models of biological phenomena, but the major topics are:

- Neuron Modelling - studies neural cells and creates mathematical models to mimic them;
- Development of Neural Systems - studies how brain develops during its life: how networks are formed and how they change through time;
- Perception or Sensory Processing - studies the inputs of the brain and how it processes them;
- Memory and Synaptic Plasticity - studies how the brain learns, how it modifies itself and what is the role of external and internal environments on this task;
- Information Processing or Behaviors of Networks - studies how neurons interact with other neurons and what is the relationship between input and output of multi-neuron systems;
- Cognition, Discrimination, and Learning - studies how the brain learns, i.e., how environment or different responses to different stimuli (discrimination) sculpt new behaviors;
- Consciousness - studies the correlation between the information in the brain and the consciousness of this information.

The objective of this work is on better modelling of the brain and the chosen subject is the fruit fly brain, for the reasons presented in Chapter 1. From all topics presented above, this thesis will focus on modeling perception and information processing. The first step to model both perception and the information processing, is to define the primitives used, i.e., neurons and synapses. Next section, presents all neuron and synapses models used in this work, respectively, integrate-and-fire (IAF) and Morris-Lecar neurons, and gradded-potential and alpha synapses.

2.2. Modelling neurons and synapses

The neuron is the fundamental unit for analysis. Its input-output membrane electrophysiology defines its function, and this arises from the detailed, precise structure of the neurons somatic morphology, dendrite morphology and synapse-channel locations. These specific neuronal structures are essentially structurally fixed, and are located in functionally specific connectional circuit architectures.

The core of the brain's functional organization is grounded in three principles:

- Neuron Doctrine → as mentioned before, the nerve cell or neuron is the fundamental building block and elementary signaling unit in the brain;
- Ionic Hypothesis → since individual nerve cells generate electrical signals, called action potentials, neurons can propagate over considerable distances with very precise destinations (very different from other communication channels - e.g.: hormones);
- The chemical theory of synaptic transmission → a nerve cell communicates with another by releasing a chemical signals called neurotransmitters, and the second cell recognizes the signals and responds by means of a specific molecules in its surface membrane called receptors.

All three principles affects how neurons behave, encode information and communicate. However, the form of the spikes generated by stereotyped neurons does not play a dominant role in the nervous system. The underlying assumption is that only the influence of the pre-synaptic spike on the post-synaptic membrane potential, which is also stereotyped, is essential for the firing times. Thus, the detailed ion-channel dynamics during the generation of the spikes will be neglected to concentrate the effort entirely on the dynamics leading up to their generation. The next section describes the mathematical models of neurons and synapses used in this work.

2.2.1. Integrate-and-fire neuron model

The first model to be presented here is the integrate-and-fire (IAF) neuron model. The IAF neuron model consists of a capacitor C in parallel with a resistor R driven by a current $I = I(t)$. The output of the model is the membrane voltage $V = V(t)$, as shown on Equation 2.1; when the resistance $R \neq \infty$, the model is said to be leaky (Equation 2.1).

$$\frac{dV}{dt} = -\frac{V(t)}{RC} + \frac{1}{C} I(t) \tag{2.1}$$

In IAF models the form of the action potential is not explicitly described and the "firing times" t_k, $k \in \mathbb{Z}$, are defined by the threshold criterion $V(t_k) = \delta$, for all $k \in \mathbb{Z}$, where δ is the threshold.

Immediately after time t_k the potential is reset to the resting value V_0 and V is assumed to obey Equation 2.1. Thus, the combination of leaky integration given in Equation 2.1 and reset defines the basic IAF model. The IAF neuron with an absolute refractory period exhibits a dead time Δ just after the occurrence of t_k, during which no integration occurs; the integration process restarts at time $tk + \Delta$.

2.2.2. Morris-Lecar neuron model

The second and last neuron model presented in this thesis is the conductance based model Morris-Lecar (MORRIS; LECAR, 1981). This model was named after Cathy Morris and Harold Lecar, who derived it in 1981. Because it is two-dimensional, the Morris-Lecar model is one of the favorite conductance-based models in computational neuroscience. For simplicity, this work uses a two-dimensional system that resembles the Morris-Lecar neuron model with parameters that do not generate spikes. This decision was made since it will be used in one of the study cases presented in Chapter 4 as a non-spiking neuron. Equation 2.2 presents the description of the model in a the systems of differential equations.

$$\begin{aligned}
\frac{dV}{dt} &= b - I_{syn} - g_L\left(V - E_L\right) - 0.5 g_{Ca} X - g_K n\left(V - E_K\right) \\
X &= \left(1 + \tanh\left(\frac{V - V_1}{V_2}\right)\right)\left(V - E_{Ca}\right) \\
\frac{dn}{dt} &= \left(0.5\left(1 + \tanh\left(\frac{V - V_3}{V_4}\right)\right) - n\right)\left(\phi \cdot \cosh\left(\frac{V - V_3}{2V_4}\right)\right)
\end{aligned} \tag{2.2}$$

In Equation 2.2, V is the membrane potential, b is a preset constant bias current, and $I_{syn} = I_{syn}(t)$ is the input synaptic current. E_L, E_Ca and E_K are reverse potential values, and g_L, g_{Ca} and g_K are maximum conductance values. Finally, V_1, V_2, V_3, V_4 and ϕ are the parameters to be adjusted in order to change the response of the neuron.

2.2.3. Synapses

Synapses are specialized structures that have evolved to pass information from one neuron to another. The origin of the name synapses is from the Greek syn (together) and haptein (to fasten). They can be electrical or chemical.

Electrical synapses or gap junctions (Figure 2.1-A) are suitable for high-speed transfer of information and for synchronization purposes. An example of electrical synapses is the signal processing in the retina. Chemical synapses (Figure 2.1-B) convert a pre-synaptic electrical signal into a chemical signal (neurotransmitters) and back into a post-synaptic electrical signal. Neurotransmitters are molecules released from the pre-synaptic terminal of a chemical synapse into the synaptic cleft. Chemical synapses are divided into two types:

- **Excitatory** - synaptic action that increases the probability that an action potential will occur in the post-synaptic neuron;
- **Inhibitory** - synaptic action that decreases the likelihood of a post-synaptic action potential occur.

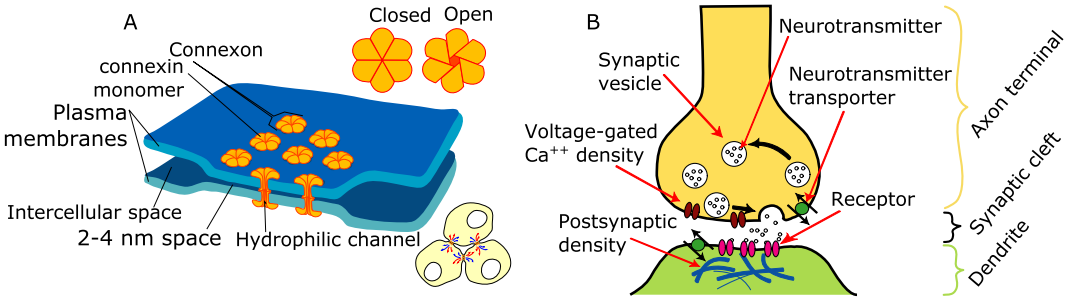

Figure 2.1: A: Gap cell junction that connects the cytoplasm of two cells directly. Ions and electrical impulses are directly transmitted through a regulated gate between cells. B: In a chemical synapse, an action potential from the membrane of the pre-synaptic cell reaches the synapse causing a release of calcium ions, which activates a set of calcium-sensitive proteins attached to vesicles that releases neurotransmitters into the synaptic cleft. Both figures are licensed under Public Domain via Wikimedia Commons.

In the simulations on Chapter 4, there are two main types of synapses used: (1) chemical between spiking neurons and (2) chemical between non-spiking neurons. For chemical synapses between spiking neurons, the model used in this work assumes that synapses are simple excitatory or inhibitory connections between neurons, ignoring the complexity and diversity of synapses for simplicity. However, the synapse function, used here to model post-synaptic conductance changes, is the more realistic α-function that takes in account the decay over time. Equation 2.3 and Figure 2.2-A shows the relation between the conductance and the time that a spike was released.

$$g_s(t) = \frac{t}{\tau_s} e^{-\frac{t}{\tau_s}} \tag{2.3}$$

where τ_s specifies the duration of the response as shown in Figure 2.2.

For the chemical synapses that are activated by the graded potentials of non-spiking neurons, a simple model is used to capture the tonic release of neurotransmitter and its effect on the post-synaptic conductance. Equation 2.4 defines a function that maps the pre-synaptic membrane potential to the post-synaptic conductance:

$$g(t) = \min\left(g_{sat}, k\left(\max\left(V_{pre}\left(t - t_{delay}\right) - V_{th}\right)^n, 0\right)\right) \tag{2.4}$$

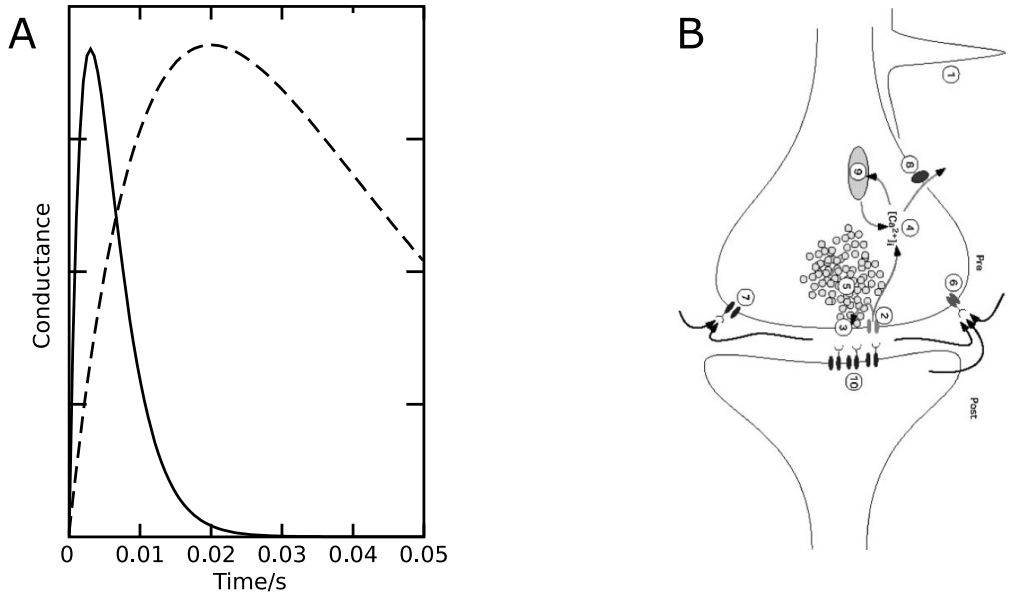

Figure 2.2: A: post-synaptic conductance generated by an alpha function with $\tau_s = 3ms$ (straight line) and $\tau_s = 20ms$ (dashed line). B: Possible sites of regulation of the synaptic decay.

where V_{pre} is the membrane potential of the pre-synaptic neuron and g_{sat}, V_{th}, k, n are parameters representing the saturation of conductance, threshold, scale and power, respectively, each of which can be modified in order to configure the model. A configurable synaptic delay t_{delay} is added to each synapse to approximate the disparate time scales involved in the synaptic release of various neurotransmitters.

In addition to the the pre-synaptic membrane potential and the post-synaptic conductance relationship, the synaptic current between non-spiking neurons can then be determined by Equation 2.5.

$$I_{syn}(t) = g(t)\left(V_{post}(t) - V_{rev}\right) \tag{2.5}$$

where V_{post} and V_{rev} are the membrane potential of the post-synaptic neuron and the reverse potential associated with the neurotransmitter. Whether the synapse is excitatory or inhibitory can be determined by the difference between V_{post} and V_{rev}. Since the synaptic current will affect the first equation of the neuron, if V_{post} is smaller than V_{rev}, the synapse is excitatory, and vice-versa.

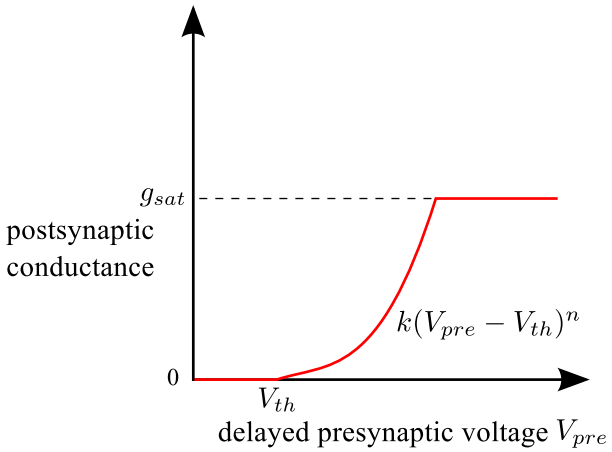

Figure 2.3: Model of synaptic transmission from a chemical synapse of a non-spiking neuron. The model takes the delayed membrane potential of the pre-synaptic neuron and maps it to a post-synaptic conductance. The function can be characterized based on the threshold, saturation, a scale variable, the power and the time delay.

2.3. The connectome and the local processing units

The connectome is the complete wiring diagram of the brain, neuron by neuron, synapse by synapse (SPORNS; TONONI; KÖTTER, 2005; LICHTMAN; LIVET; SANES, 2008). Connectomics is an emerging discipline that aims to determine the connectome of all animal species. The connectome of the fruit fly Drosophila can be represented by a matrix with, approximately, 10^{18} elements, which means many orders of magnitude smaller than that of vertebrates. Equation 2.6 shows how the matrix size is calculated. On the engineering perspective, the reasonable compromise between tractability (connectome size) and behavioral richness is one advantage of using fruit flies as a study subject.

$$\left(10^5_{neurons} \times 10^4_{synapses}\right)_{lines} \times \left(10^5_{neurons} \times 10^4_{synapses}\right)_{columns} \quad (2.6)$$

Also, as mentioned before, the research on the fruit fly transformed this tiny insect into one of the most powerful genetic model organisms (ARMSTRONG et al., 2009), which uncovers many developmental principles, genetic regulation and cell signaling. Such principles are conserved across species (ARMSTRONG et al., 2009) and may help scientists understand more complex brains and explain some inherited diseases.

In addition to the genetic toolkit, recent advances in experimental methods for precise recordings of the fly's neuronal responses to stimuli

(KIM; LAZAR; YEVGENIY, 2010; KIM; LAZAR; SLUTSKIY, 2010; KIM; LAZAR; SLUTSKIY, 2011a; KIM; LAZAR; SLUTSKIY, 2011b; WILSON, 2011) and in techniques for analyzing the fly's behavioral responses to stimuli (BUDICK; DICKINSON, 2006; KATSOV; CLANDININ, 2008; MAIMON; STRAW; DICKINSON, 2008; CHIAPPE et al., 2010) have been facilitating the shaping of circuits. Also, progress in the reconstruction of the fly connectome (CHKLOVSKII; VITALADEVUNI; SCHEFFER, 2010; TAKEMURA et al., 2013), by using identified neurons - stereotyped neurons that can be located in every fly (OLSEN; WILSON, 2008) - has contributed much to circuit modeling.

Although the size of the fruit fly connectome is much smaller than the vertebrates counterparts, there are many difficulties regarding gathering the connectome information (OLSEN; WILSON, 2008; ARMSTRONG et al., 2009). Electrophysiological recordings, imaging techniques and pharmacological methods are very hard to implement in a such small brain. However, the Drosophila connectome may uncover all behavior of the fly, which can help explain (ARMSTRONG et al., 2009) the behavior of other species, including our own.

Studies on the brain of the *Drosophila Melanogaster* have revealed that it comprises about 40 distinct modular subdivisions, most of which correspond to anatomical regions in the brain associated with specific sensory modalities and locomotion. These modules are referred as Local Processing Units (LPUs), because they possess a characteristic population of local neurons. Given that many LPUs are associated with specific stimulus processing that controls behavior, they can be regarded as the functional building blocks of the fly brain. Also, many LPUs' local synaptic connectivity is organized into distinctive and repeated canonical subcircuits that appear integral to their respective functions. To model these LPUs, it is then highly desirable to be able to specify and connect multiple instances of subcircuit models without having to explicitly refer to their contents.

2.3.1. Local Processing Units

Brain regions are traditionally defined by anatomically distinct boundaries not necessarily representative of functional subdivisions (CHIANG et al., 2011). Neurons in the fly brain can be categorized into two functionally distinct populations: local neurons (LNs), whose processes are restricted within a single brain region; and projection neurons (PNs), whose dendrites and axons connect between two or more brain regions.

To develop a strategy for identifying basic brain building blocks, essential

for analyzing network characteristics, Chiang and his group (CHIANG et al., 2011) began with the well-characterized local processing unit, the antennal lobe (AL). The AL comprises four classes of neurons: (1) input olfactory sensory neurons, (2) LNs, (3) output PNs, and (4) centrifugal neurons. With a semi automated search algorithm, Chiang and his group identified all local neurons in the AL and presented at http://www.flycircuit.tw.

Assuming that an LPU is equal or smaller than a neuropil (an anatomically demarcated 3D region), Chiang and his group implemented a method to detect whether a neuropil can be further subdivided into smaller LPUs, each containing its own population of LNs. Such method allowed one to specify a candidate LPU based on a mathematical definition of spatial features of LN branches that entangle with each other. Seven steps are employed for the detection and validation of candidate LPUs: (1) calculate the density of local neurons in a voxel, (2) identify hot spots by statistics, (3) determine subdivisions by cluster analysis, (4) list local neurons into each subdivision, (5) define boundaries for each candidate LPU from local neurons clusters, (6) calculate the spatial distributions of local neurons fibers inside each candidate LPU, and (7) validate an LPU to see whether the region has its own long-range tracts (CHIANG et al., 2011).

In order to facilitate the understand in LPU identification process, Chiang and his group partitioned the ventrolateral protocerebrum (VLP) region of the Drosophila brain into two LPUs, as depicted in Figure 2.4 (D.a). Three criteria suggest that the dorsal and ventral VLPs may be two separate functional units Figure 2.4 (D.b): (1) each unit has its own population of local neurons with segregated cell body locations - Figure 2.4 (D.d), (2) neural fibers projected from the two local neurons clusters are segregated Figure 2.4 (D.e), and (3) each unit has its own characteristic long-range tracts communicating with different partners - Figure 2.4 (D.g). Therefore, an LPU is defined as a brain region consisting of its own local neurons population whose nerve fibers are completely restricted to that region. Further, each LPU is contacted by at least one neural tract. In contrast, a brain region, as defined by morphologically distinct boundaries, may or may not have its own population of local neurons and neural tract. (CHIANG et al., 2011).

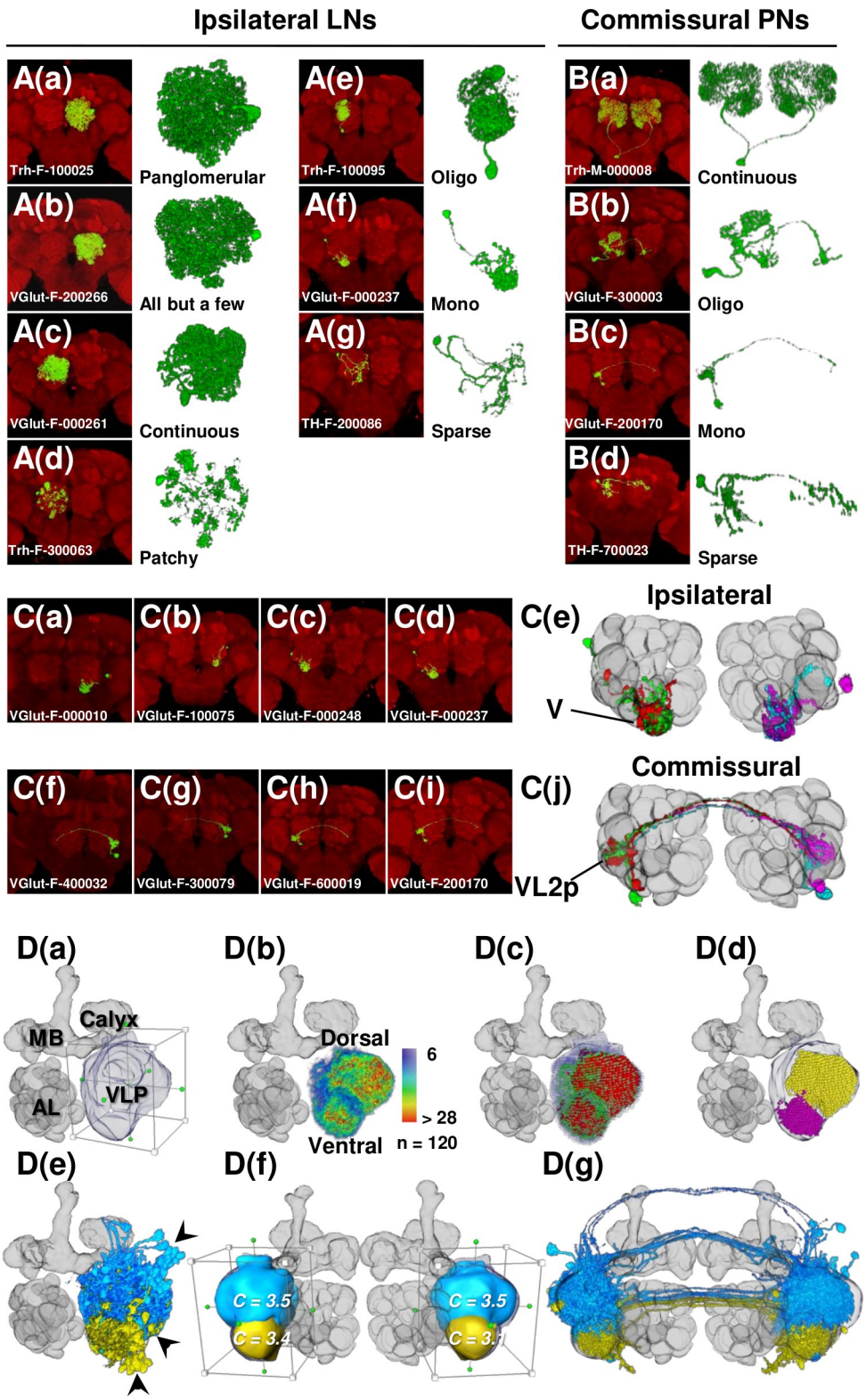

Figure 2.4: Characterization of an LPU: (A) Diversity of AL local neurons; (B) diversity of AL commissural projection neurons; (C) stereotyped monoglomerular local neurons and commissural projection neurons; (D) steps for defining an LPU.

2.4. Existing computational tools

2.4.1. The specification language NeuroML

An important tool that is available for the specification of neural circuitry is NeuroML, a meta-language, based on XML (Extensible Markup Language). Although very simple and easy to use, it is very powerful because it allows detailed models and their components to be defined in a standalone form to be used across multiple simulators (NEURON (CARNEVALE; HINES, 2006), GENESIS (BEEMAN, 2005), MOOSE (CUBERT; FISHWICK, 1997), NEST (GEWALTIG; DIESMANN, 2007)) and to be archived in a standardized format (GLEESON et al., 2010). NeuroML addresses many compatibility issues between software tools, facilitating the reproduction of published models descriptions and results. It also allows the sharing and reuse of model components and the development of new tools for detailed computational modeling (GLEESON et al., 2010).

NeuroML is a declarative, XML-based model description language for computational neuroscience. Currently, NeuroML is in its second version, where the structure and behavior of ion channel, synapse, cell, and network model descriptions are based on underlying definitions provided in LEMS (Low Entropy Model Specification) language, which is a new XML-based language for expressing hierarchical mathematical models of physical entities (CANNON et al., 2012).

Before this second version, NeuroML focussed on conductance-based cell models, often with a corresponding multicompartmental representation of neuronal morphology, which means that every neuron should be extensively described. The current scope of NeuroML covers abstract, point neuron models, conductance based neuron models, morphologically detailed, multicompartmental neuron models, voltage, and calcium dependent ion channel models, both fixed and plastic synapse models, and models for networks of neurons positioned in 3D with synaptic connections among populations of cells. In addition, NeuroML v2 was designed in conjunction with LEMS, which can be used for creating fully machine-readable definitions of the structure and behavior of neuron model components (VELLA et al., 2014).

The LEMS language is used to formally describe the components of models of physical systems, which may contain hierarchical relationships. In addition, LEMS has two important keys for NeuroML: (1) the containment

of components, which encodes the concept that one model element is part of another, and (2) the ability to declare a prototype *ComponentType*, which defines the generic structure and dynamics for a broad class of models.

However, although NeuroML addresses the compatibility problem in many ways, it has its limitations regarding the specification of local processing units, since it was intended to address the variety of neurological systems organized in a biological fashion. In order to have specifications of functional units with their canonical subcircuits abstractions, it is necessary to have a tool that offers both a standardized language and support to components that abstract new blocks on the engineering perspective.

Besides the NeuroML specification language, there are other computational tools available for data analyses, visualization, simulation, etc. In the next sub-sections, many different tools will be discussed. Notice that, every application mentioned in this section, either understands NeuroML or has an interface to translate NeuroML to its proprietary format, which makes NeuroML a common language between them (GLEESON et al., 2010).

2.4.2. Public databases

Public databases, such as FlyCircuit (CHIANG et al., 2011) and NeuroMorpho (ASCOLI; DONOHUE; HALAVI, 2007) and others (MARKRAM, 2006), were the answer to decentralized information. Before those initiatives, research groups used to record data from the organism and store it in a private base without contact with other groups, hindering a cross validation and analysis of those recordings. FlyCircuit is a public database for online archiving, cell type inventory, browsing, searching, analysis and 3D visualization of individual neurons in the Drosophila brain, which people are encouraged to support by uploading their own recordings. NeuroMorpho is another public database accessible through any web browser. Its entire repository of neuronal morphologies can be browsed by cell type, brain region, animal species, or by the contributing laboratory.

2.4.3. Visualization tools

Regardless of the database being used, visualization tools are one of the first efforts usually done to help understand some systems, (DAVISON et al., 2008), allowing the creation of groups, recurring elements or structures, and, additionally, making it easier to represent the problem in focus. In the Biology area, it is not different, even because many inputs are images generated

from the studied organism. In neuroscience, there are many 3D structures inside brain regions that may play a key role in information processing at network level: cerebellum, hippocampus and the complex connectivity between. It can be shown that the shapes of the dendritic trees can affect the electrical behavior of cells and that the spatial pattern of synaptic contacts can influence how signals are integrated (GLEESON; STEUBER; SILVER, 2007). Among many visualization tools, it is noteworthy to examine two programs: **neuroConstruct** (GLEESON; STEUBER; SILVER, 2007) and **CX3D** (ZUBLER; DOUGLAS, 2009).

The neuroConstruct software tool is written in Java and aims to facilitate the creation, visualization (Figure 2.5), and analysis of networks of multicompartmental neurons in 3D space (GLEESON; STEUBER; SILVER, 2007). It provides a GUI for configuration and visualization of neural structures, and import/export mechanisms for two other simulators: NEURON (HINES; CARNEVALE, 1997) and GENESIS (BEEMAN, 2005). The functionalities of neuroConstruct can be grouped into five main areas:

- **import and validation** of Morphologies, where reconstructed neuronal morphologies can be imported and automatically checked for errors;
- **creation of conductance-based cell models** that models detailed cellular mechanism essential for reproducing the complex behavior of real neurons;
- **network generation**, where cell models can be placed within a region of 3D space at a specified density and synaptic connections can be generated according to specified sets of rules;
- **simulation management** for generating script files for the simulator packages NEURON or GENESIS and storing the results in text files;
- **network analysis** for post simulation examination.

CX3D (ZUBLER; DOUGLAS, 2009) is both a development simulator and a visualization tool, where the second functionality is powered by a free software called Blender (*blender.org*). In CX3D, neurons are represented by spherical (for the soma) and cylindrical (for neurites) elements that have appropriate mechanical properties - Figure 2.6. The CX3D simulator can run neural development either by construction algorithms or biologically-inspired growth processes.

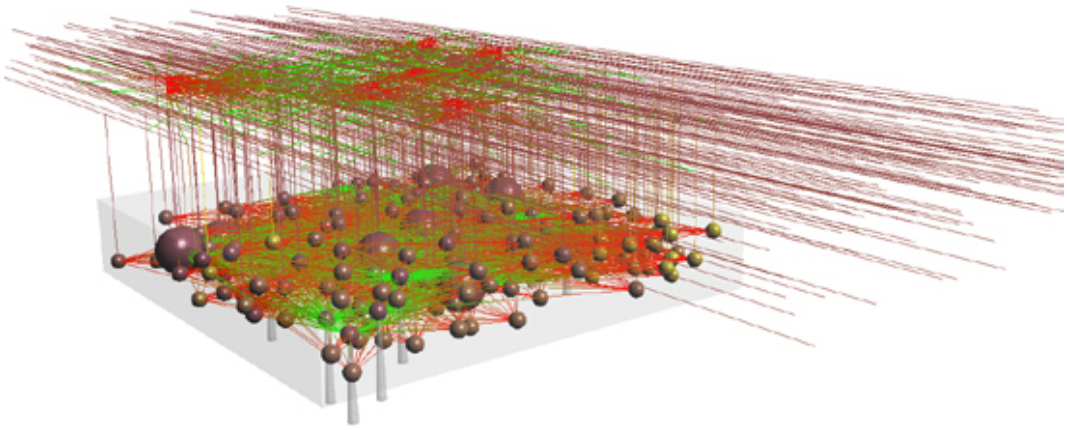

Figure 2.5: A neuronal network visualization generated by neuroConstruct.

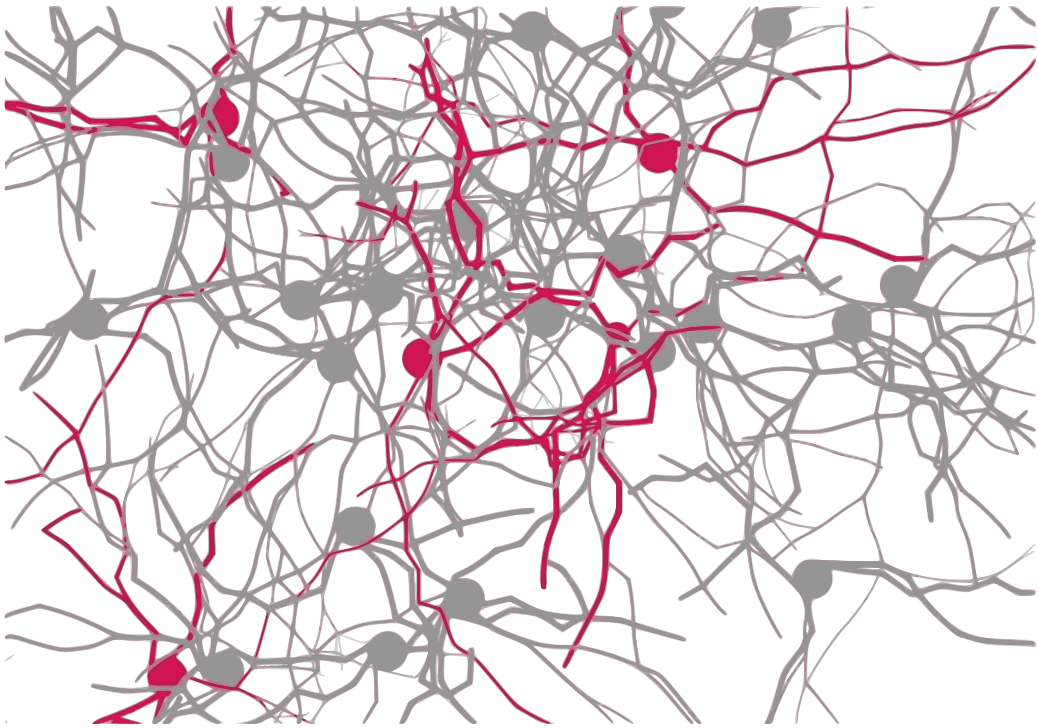

Figure 2.6: A neuronal network grown in the framework CX3D. Some of the cells including their neurites are shown red for better visualization of the growing process.

2.4.4. Simulation tools

Computational models are increasingly being used in neuroscience to assist the exploration and interpretation of complex phenomena (CANNON et al., 2007). However, computational problems related to spiking neurons are numerous and diverse, resulting in solution requirements that range from the use of detailed biophysical representations of the neurons, such as the Hodgkin and Huxley model, to, in other cases where it is not crucial to realistically capture the spike generating mechanisms, simpler models, for example, the integrate-and-fire model (IAF) (BRETTE et al., 2007). IAF cells are also faster to simulate than conductance ones, and their use is justifiable in a real-time simulation or in large-scale network simulations.

Simulation tools can be divided into 3 main groups: (1) neuronal branching simulators, (2) detailed channels simulators, (3) synapses, neurons and network simulators. The principles governing axonal and dendritic branching are considered essential for unravelling the functionality of single neurons and the way in which they connect (CUNTZ et al., 2010). Channel models are used to regard the neuronal activity that is mediated through changes in the probability of stochastic transitions between open and closed states of ion channels (CANNON; O'DONNELL; NOLAN, 2010). Synapses, neurons and network simulators are more common and, usually, they balance the level of details and large scale simulation. Right now, it is not possible to have a full network simulation, with thousands of neurons, and a complete modelled system, with all channels and every morphological detail in a cell, even because this information is not yet available.

One of the first spiking neurons simulators and also the most used one is NEURON (CARNEVALE; HINES, 2006). It was published in (HINES; CARNEVALE, 1997) as a powerful and flexible environment for implementing biologically realistic models of electrical and chemical signaling in neurons and networks of neurons. In its presentation the authors state that *biologically realistic model* does not mean infinitely detailed, but a choice of the investigator who is free to construct the model not limited by the simulator. Thus, instead of having cells or synapses implementations, NEURON offers a set of equations describing phenomena ranging from the relation between current and voltage in a one-dimensional cable to integrators.

NEURON would eventually be improved to work in distributed infrastructures (HINES; CARNEVALE, 2008), but newer simulators already offer GPU-based code. The latest one is (THIBEAULT; HOANG; Harris Jr, 2011), which is a parallel modelling environment, written for large-scale neural

simulations, which aims to prove that a generalized simulation architecture can have both extensibility and high-performance, an issue also discussed in (CHEVITARESE; SZWARCMAN; VELLASCO, 2012). Thibeault et al., in (THIBEAULT; HOANG; Harris Jr, 2011), chose a "lighter" neuron model from (IZHIKEVICH, 2003) for a reasonable compromise between execution time and biophysical plausibility. However, because of this choice, it only simulates spiking neurons.

The workflow of the (THIBEAULT; HOANG; Harris Jr, 2011) simulator is basically made up of the processing of a simple input file, which describes a neuronal net in a proprietary format, and the distribution of the constituents neurons amongst available GPUs for simulation. The process starts by forming a local index and representation of the neurons that will be shared on device threads forming a local neuron structure array. After building this and other memory structures, the simulation itself begins by updating (numerically integrating by Euler method) the neurons, and then pre and post synaptic currents. The results presented by the authors show that a simulation of 100,000 neurons, with 50 connections per neuron, can run in about 1.2 times real time.

All simulators cited before focus either on cells dynamics, on chemical reaction or in network development/organization in a general brain. Although brains seem to compute information on the same way (OLSEN; WILSON, 2008), on the engineering perspective, different brains have different sizes, different cells and connection densities, in addition to different behavioral repertoires. These factors might impact on how the code is implemented. In order to have a fine-tuned application, it is necessary to opt for an organism specific simulator, such as Neurokernel (GIVON; LAZAR, 2012) and Geppetto (*openworm.org*).

Neurokernel is an open software architecture, developed at Columbia University, for emulating neural circuit modules in the fly brain and their responses to recorded or simulated input stimuli on multiple Graphics Processing Units. The key feature of this architecture is its support for integrating instances of different neural circuit models developed by independent researchers by requiring that the models' implementations provide interoperable interfaces that adhere to the specification prescribed by the architecture (GIVON; LAZAR, 2012).

Geppetto is another simulation platform engineered to support simulation of biological systems and their environment. This software is the effort that is closest to the work proposed here, but it is being developed specifically for the C. Elegans organism. Geppetto architecture is divided into

5 levels: **Entity**, the basic building block of the simulated world; **Model**, which describes a specific aspect of an entity; **Simulation**, which is its top level controller; **Simulator**, which is directly responsible for the simulation of a class of models, by employing one or more solvers; and **Solver**, which is the lowest level component of the simulation stack and is in charge of mathematical computation.

2.4.5. Additional existing tools

In addition to the tools presented in this section, there other software envisioned to address the various types of problems regarding the virtualization of the brain.

- **translators** → converts one language to another (JORDAN; PERRY; NARALA, 2012);
- **data analysers** → are used to process the output of recordings (RODRIGUEZ et al., 2008; CARDONA et al., 2012; WEARNE et al., 2005; RODRIGUEZ et al., 2008);
- **neuron set generators** → creates populations of cells based in some pre-defined grown pattern (EBERHARD; WANNER; WITTUM, 2006);

Finally, to allow a virtual fly brain to be constructed and then used afterwards, it is necessary to put together and specialize many of those tools mentioned in this chapter: image processing and analysis, public data resources, efficient simulators and some kind of representation to support annotation and modelling (ARMSTRONG et al., 2009). On the next chapter, we propose a set of tools and methods to address both annotation and modeling.

3
Specifying neural circuits as functional blocks

As an innovative contribution to the computational neuroscience research area, this thesis presents CircuitML (CML), a framework for modeling the virtual brain as a set of functional building blocks, instead of representing them as networks of neurons interconnected by synapses. Each building block comprises smaller components, allowing one to study, for example, consequences of targeted brain disruption in a "IF-THEN" manner, as proposed by (ARMSTRONG et al., 2009): "if I remove this neuron, what behavior would be affected?" or "if someone changes the motion detection system, then...", or even "if one adds more channels to the olfactory system, then...". It also allows scientists to share new discoveries with other research groups, and share those new circuits with the scientific community.

In order to achieve this new level of abstraction, this work presents the design and implementation of CircuitML, a structured description language, which first version was published in (CHEVITARESE et al., 2013). CML can describe neuronal circuits at a level above NeuroML (NML), inheriting its support to many tools and simulators, and allowing scientists to share and validate their discoveries (GLEESON et al., 2010). In addition, such inheritance gives to CircuitML a great variety of elements, ranging from neuronal morphologies, with MorphML (GLEESON et al., 2010), to an entire brain comprised by LPUs.

As a companion tool for CircuitML, a Python API is also developed, called libCircuitML, which can be imported into a Python script allowing users to:

- load and validate CircuitML and NeuroML files;
- parse and edit circuit models, which closely follow the structure of the XML language;
- save valid XML files either on NeuroML format or on CircuitML format;
- use additional functionalities that make it easier to create large models;
- connect to neurokernel for simulation purposes.

The main goal of libCircuitML is to provide easy-to-use utilities for the manipulation of CircuitML using pythonic tools familiar to programmers, but also easy to use by less experienced ones. This chapter is divided in three main sections: (1) CircuitML language, (2) translating process and (3) computing the virtual brain.

3.1. CircuitML language

3.1.1. Technical description on XML specification

A CircuitML document consists of XML elements describing the circuit components of the neuronal system. The structure of a valid CircuitML document is defined using XML Schema Definition (XSD) files and, therefore, standard XML handling libraries can be used to check its validity. An error will be generated if, for example, the name attribute is missing from the sub-circuit element.

Once an XML file is known to be in accordance to the CircuitML format, the contents of the file can be transformed into other formats in a number of different ways, such as SAX (Simple API for XML) or DOM (Document Object Model). It is also possible to convert the original file onto other text or script formats with Extensible Stylesheet Language (XSL) files, which makes CircuitML accessible from simulators including NEURON, GENESIS and PSICS. This approach has the advantage that applications need not be reimplemented to natively support CircuitML, but can still have access to models in the format.

3.1.2. CircuitML Overview

Before starting to describe CircuitML, it is important to understand the its advantages, since NeuroML can describe every single detail that CircuitML does. The most important advantage is that CircuitML makes scientists to look to neural systems as a big circuit with "chips" (LPUs) interconnected with each other. Each "chip" has its own and unique functionality and a well-defined interface. Such aspect of CircuitML takes us back to the very beginning of object-oriented paradigm:

> *"... serves to help manage the complexity of massive software-intensive systems"* (BOOCH, 1986).

NeuroML provides all primitives and functionality declarations, but CircuitML provides all data abstraction and information hiding needed to make the entire specification process simpler. The advantages of CircuitML over NeuroML are:

- minimum-to-zero coupling between circuit elements ("chips");
- clearly defined interfaces allowing the abstraction of data and circuit details inside elements;
- reuse and code clearness.

The current scope of CircuitML covers the definition of functional modules (LPUs) with interfaces, smaller modules, called sub-circuits and the connectivity module that glues all modules together. Figure 3.1 shows the overview of the structure of CircuitML

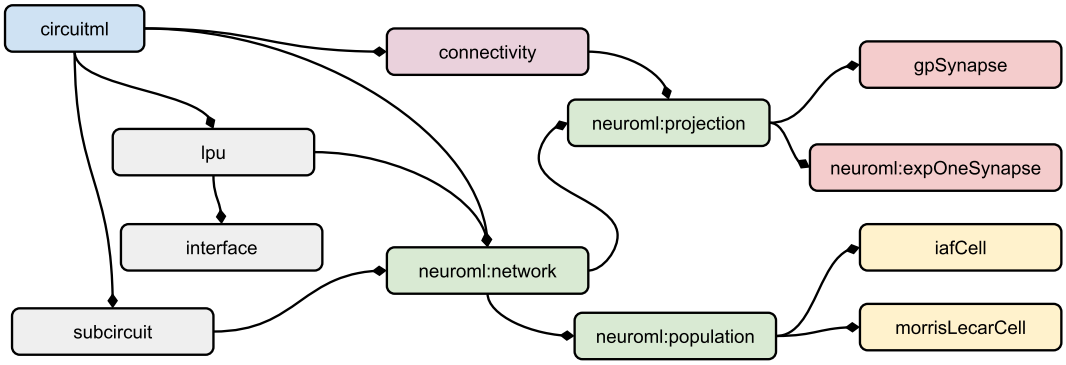

Figure 3.1: Overall structure of CircuitML. The top-level element of CircuitML, *circuitml*, contains a number of child elements of various types The hierarchical structure is depicted in colors: the *circuitml* element can comprise *lpu* and *subcircuit* elements; in turn, *lpu* elements has an *interface* and may ecapsulate *network* elements, which may have populations of cells (*population*) and its local connectivity (*projection*). Finally, the *connectivity* element can encapsulates *projection* elements used to connect *lpu* elements.

3.1.3. Levels in CircuitML

Figure 3.2 shows the relationship between biological scale of information processing in neural systems and CircuiML. The very first level is MorphML followed by ChannelML (level 2) and, then, NetworkML (level 3); all three levels are defined in the NeuroML's abstraction stack. This work introduces a fourth level (CircuitML) over the other ones, which adds a functional modularity to the specification.

Figure 3.2 shows some examples of those levels on the fly brain. The yellow box represents a simple demo of the antenna lobe (AL) with multiple

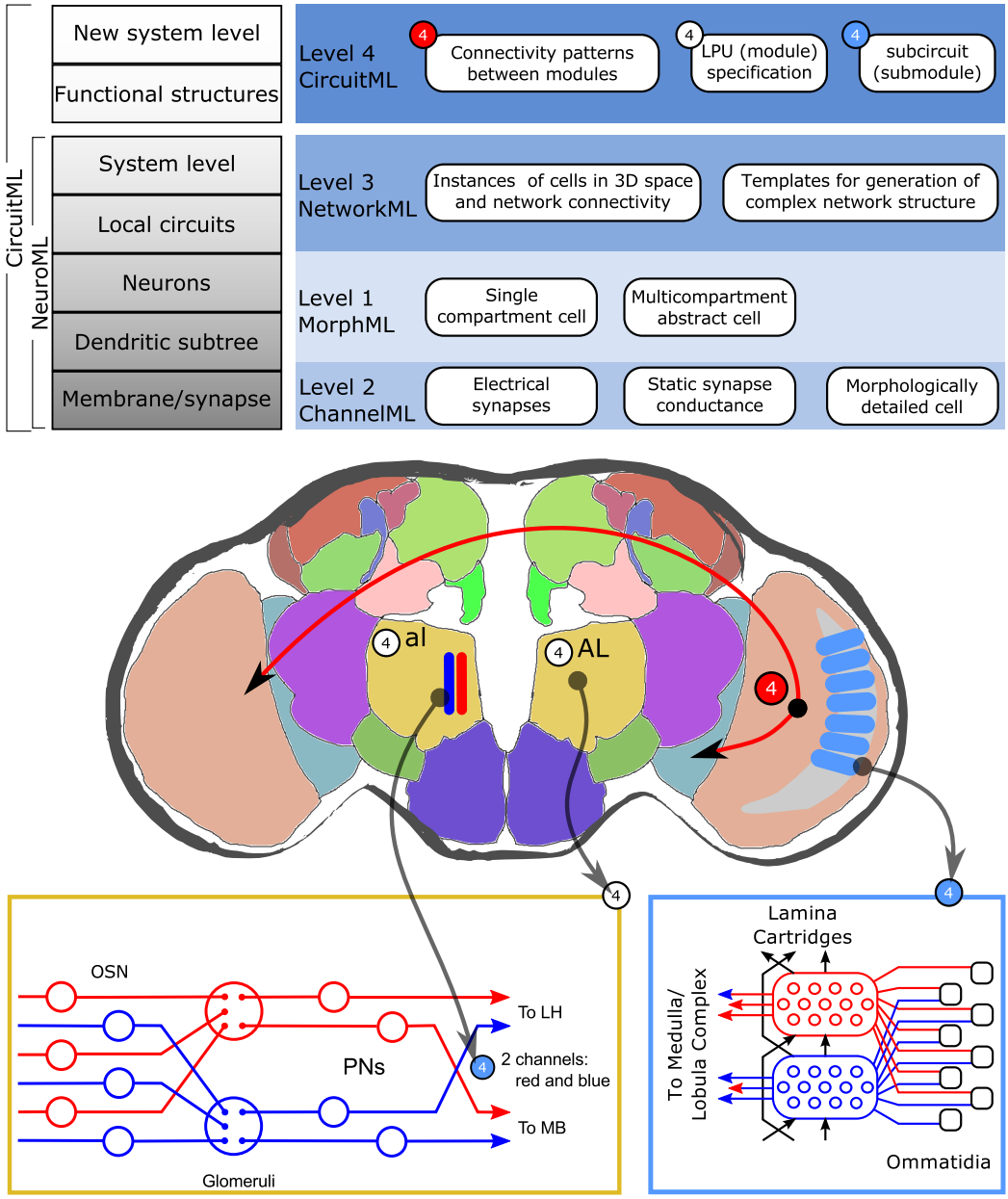

Figure 3.2: Abstraction levels in CircuitML and their relationship with the biological scale in neural systems, where MorphML, ChannelML and NetworkML are, respectively, levels 1, 2 and 3 of NeuroML's abstraction stack. In the middle: fruit-fly brain with many neuropils depicted, where arrows represent projections from one LPU to another and the red, white and blue balls regard one of the three components in level 4 (connectivity, LPU and subcircuit). Blue box: lamina LPU in the visual system that comprises hundreds of cartridge structures. Yellow box: antenna lobe LPU with two cartridges, blue and red lines.

olfactory channels. Each channel comprises olfactory sensory neurons (OSNs) and projection neurons (PNs). In this case, there are three layers: (1) AL, which can be specified as an *lpu* element; (2) channels that can be specified as sub-circuits (*subcircuit* element); and (3) sets of neurons, where each set can be represented by a *population* element. On the blue box, lamina (*lpu*) comprises multiple cartridges (*subcircuit*), each of which receives 8 photoreceptors (*population* of photoreceptors).

Level 1: Morphologies and metadata

The first Level of CircuitML uses NeuroML Level 1 and has two main purposes: to define neuronal morphologies (MorphML) and metadata, which provides additional information about model components at this and subsequent levels. At this level, cells are specified in the XML cell element and are described by lists of segment elements, each containing the 3D location and diameter of its proximal and distal ends (GLEESON et al., 2010).

Listing 3.1 shows an example of a simple cell with 6 segments. There is also a user-friendly view of the code on Table 3.1 and a neuroConstruct render of it on Figure 3.3. By looking to the code, it is possible to notice that, at this level, the specification is a list of segments and a synapse occurs when two segments touch each other. Although this is useful to describe very detailed morphologies, because of its simplicity, a network with many neurons would be hard to be described only by MorphML.

Listing 3.1: MorphML code snippet for a simple cell

```
<morphml length_units="micrometer">
  <cells>
    <cell name="SimpleCell">
      <meta:notes>A Simple cell example.</meta:notes>
      <mml:segments>
        <mml:segment id="0" name="Soma" cable="0">
        <mml:proximal x="0.0" y="0.0" z="0.0" diameter="16.0"/>
        <mml:distal x="0.0" y="0.0" z="0.0" diameter="16.0"/>
        </mml:segment>
        <mml:segment id="1" name="mainDend" parent="0"
         cable="1">
        <mml:proximal x="0.0" y="0.0" z="0.0" diameter="2.0"/>
        <mml:distal x="20.0" y="0.0" z="0.0" diameter="2.0"/>
        </mml:segment>
        <mml:segment id="2" name="subDend1" parent="1"
         cable="2">
        <mml:proximal x="20.0" y="0.0" z="0.0" diameter="2.0"/>
        <mml:distal x="40.0" y="15.0" z="0.0" diameter="2.0"/>
```

```
19          </mml:segment>
20          <mml:segment id="3" name="subDend2" parent="1"
21           cable="3">
22          <mml:proximal x="20.0" y="0.0" z="0.0" diameter="2.0"/>
23          <mml:distal x="45.0" y="0.0" z="0.0" diameter="2.0"/>
24          </mml:segment>
25          <mml:segment id="4" name="subDend3" parent="1"
26           cable="4">
27          <mml:proximal x="20.0" y="0.0" z="0.0" diameter="2.0"/>
28          <mml:distal x="40.0" y="-15.0" z="0.0" diameter="2.0"/>
29          </mml:segment>
30          <mml:segment id="5" name="mainAxon" parent="0"
31           cable="5">
32          <mml:proximal x="0.0" y="0.0" z="0.0" diameter="1.0"/>
33          <mml:distal x="-30.0" y="0.0" z="0.0" diameter="1.0"/>
34          </mml:segment>
35        </mml:segments>
36        <mml:cables>
37          <mml:cable id="0" name="Soma">
38          <meta:group>all</meta:group>
39          <meta:group>soma_group</meta:group>
40          </mml:cable>
41          <mml:cable id="1" name="mainDendSec"
42           fract_along_parent="0.5">
43          <meta:group>all</meta:group>
44          <meta:group>dendrite_group</meta:group>
45          </mml:cable>
46          <mml:cable id="2" name="subDendSec1">
47          <meta:group>all</meta:group>
48          <meta:group>dendrite_group</meta:group>
49          </mml:cable>
50          <mml:cable id="3" name="subDendSec2">
51          <meta:group>all</meta:group>
52          <meta:group>dendrite_group</meta:group>
53          </mml:cable>
54          <mml:cable id="4" name="subDendSec3">
55          <meta:group>all</meta:group>
56          <meta:group>dendrite_group</meta:group>
57          </mml:cable>
58          <mml:cable id="5" name="mainAxonSec"
59           fract_along_parent="0.5">
60          <meta:group>all</meta:group>
61          <meta:group>axon_group</meta:group>
62          </mml:cable>
63        </mml:cables>
64      </cell>
65    </cells>
```

```
66 </morphml>
```

Table 3.1: Simple cell morphology demo

Name	**SimpleCell**
Description	A Simple cell for demonstration purposes.
Number of segments	**6**
Number of cables	**1** soma cable, **4** dendritic cables and **1** axonal cable
Cable details	**Soma (id:0)**, number of segments in cable: **1** *Groups: all; soma_group* **mainDendSec (id:1)**, fraction along parent cable: **0.5**, number of segments in cable: **1** *Groups: all; dendrite_group* **mainDendSec1 (id:2)**, number of segments in cable: **1** *Groups: all; dendrite_group* **mainDendSec2 (id:3)**, number of segments in cable: **1** *Groups: all; dendrite_group* **mainDendSec3 (id:4)**, number of segments in cable: **1** *Groups: all; dendrite_group* **mainAxonSec (id:5)**, fraction along parent cable: **0.5**, number of segments in cable: **1** *Groups: all; dendrite_group*

MorphML also implements metadata specification, which is important for tracking all information other than the expected one for the model components, which provides background information about the model. Many elements are already defined to provide structured information on the author list (***authorList***), publications associated with the model (***publication***) and other semi-structured information (***properties***, ***annotation***), as well as general text based comments (***notes***). In addition to those elements, an status element (***status***) is also included to allow a record of any known limitations of the model.

MorphML provides two types of unit system: SI units (cm, mV, ms, etc.) and physiological units (g, CM, RM, Ra, etc.), which are better explained on

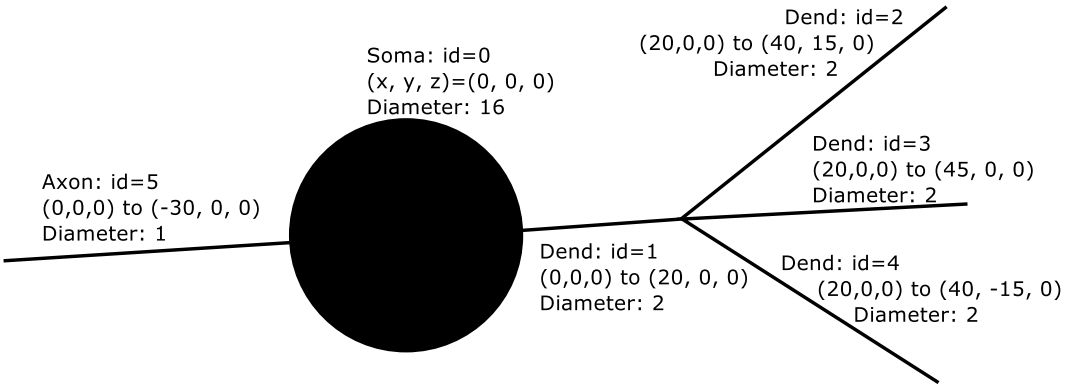

Figure 3.3: Listings 3.1 rendered using neuroConstruct.

GENESIS documentation (BEEMAN, 2005).

Level 2: Channels, synapses and channel distribution

The second Level of CircuitML uses ChannelML language, which describes the electrical properties of the membranes that underlie rapid signaling in the brain. The two main parts of this Level are: an extension of the morphological descriptions from Level 1, which includes details of the passive electrical properties and channel densities on various parts of the cell, and ChannelML, which allows descriptions of the individual conductance mechanisms.

ChannelML supports two main types of conductances:

- **electrical synapses** (*channel_type*) that arise from channels distributed over the plasma membrane, i.e., voltage-gated conductances or conductances gated by intracellular ions;
- **chemical synapses** (*synapse_type*), which are conductances arising at synaptic contacts. An example of a channel definition can be found on Listing 3.2.

At this level, the specification describes how cell channels work adding the electrical properties over MorphML specifications. Listings 3.2 describes a leak channel with the current-voltage relationship, with a simpler view on Table 3.2.

Listing 3.2: ChannelML example

```
1 <channelml units="Physiological Units">
2   <meta:notes>
3     ChannelML file containing a single Channel description
4   </meta:notes>
5   <channel_type name="leak" density="yes">
```

```
    <status value="stable"/>
    <meta:notes>
      Simple example of a leak/passive conductance. Note: for
      GENESIS cells with a single leak conductance, it is better
      to use the Rm and Em variables for a passive current.
    </meta:notes>
    <current_voltage_relation cond_law="ohmic"
     ion="non_specific" default_erev="-54.3"
     default_gmax="0.3"/>
  </channel_type>
</channelml>
```

Table 3.2: Simple leak channel demo

Name	**leak**
General notes	ChannelML file containing a single Channel description
Unit system of ChannelML file	Physiological Units
Status	Stable
Description	Simple example of a leak/passive conductance. Note: for GENESIS cells with a single leak conductance, it is better to use the Rm and Em variables for a passive current.
Current voltage relationship	ohmic
Ion involved in channel	non_specific (default $E_{non\_specific} = -54.3mV$)
Default maximum conductance density	$G_{max} = 0.3mScm^{-2}$
Conductance expression	$G_{non\_specific}(v,t) = G_{max}$
Current due to channel	$I_{non\_specific}(v,t) = G_{non\_specific}(v,t) * (v - E_{non\_specific})$

Level 3: Network connectivity

The third Level of CircuitML extends NetworkML Level 3 (CANNON et al., 2012), which allows specification of the 3D anatomical structure and synaptic connectivity of a network of neurons, together with the properties of the external input used to drive the network. This level has two main

purposes: (1) to define NetworkML and (2) to allow extension of Level 2 cells by specifying regions of the cell membrane to which specific synaptic connections are limited. This means that from this level up, there is no need to define channels in order to specify the electrical properties of neurons, which instead, is specified by point-to-point connections.

At this level, neural networks can be specified by an explicit list of instances of cell positions and synaptic connections, or as an algorithmic template for describing how instances of the network should be generated, for example to place 200 cells randomly in a certain 3D region. Figure 3.4 shows both networks. The network on the left (Listings 3.3 and Table 3.3) have 2 populations of neurons, *PopA* and *PopB*, which have, respectively, 2 and 3 neurons each with their 3D positions written near them. *PopA* is specified on Listings 3.3 (ln3), where the population name, the type of cell it will contain and the number of neurons are declared. *PopB* is specified on Listings 3.3 (ln4) with the same parameters. The connectivity between those two populations are depicted as arrows.

In this first example, the synapses are explicitly defined inside a **projection** - Listings 3.3 (ln7-ln14). **Projections** have a list of connections with pre and post synaptic neuron ids that will be connected. Taking the first connection as an example (Listings 3.3 (ln10)), it connects the first neuron of *PopA* to the second neuron on *PopB*. Notice that at this level, there is no need to inform which segments of the cell will "touch" each other, since this level already supports point-to-point connectivity.

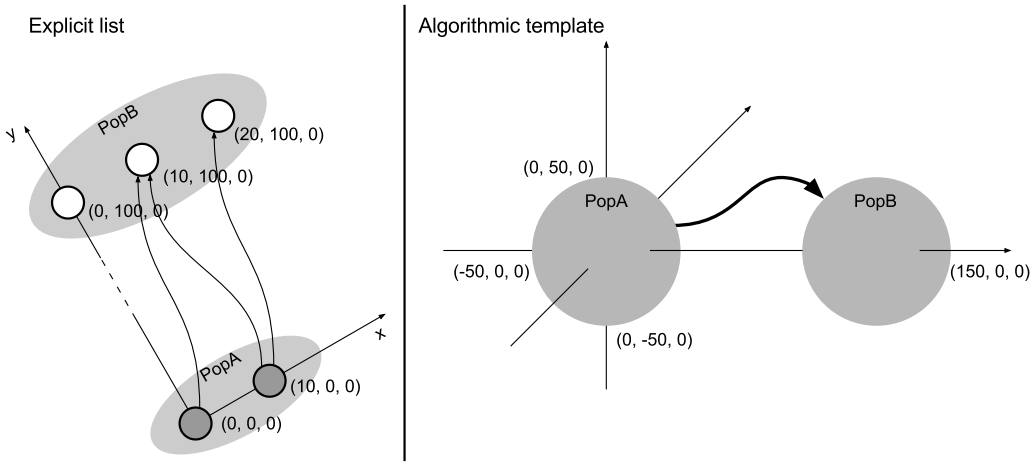

Figure 3.4: Visualization of Listing 3.3 (left) and Listing 3.4 (right). Note that, on the right example, cells are represented only by the regions they are in, and the connectivity between *PopA* and *PopB* is represented by the thick arrow, due the number of cells and synapses.

Listing 3.3: NetworkML simple code snippet - *explicit list*

```
1 <networkml id="simple_net" length_units="micrometer">
2   <populations>
3     <population name="CellGroupA" cell_type="CellA" size="2"/>
4     <population name="CellGroupB" cell_type="CellB" size="3"/>
5   </populations>
6   <projections units="Physiological Units">
7     <projection name="NetworkConnection" source="CellGroupA"
8      target="CellGroupA">
9       <synapse_props synapse_type="DoubExpSynA"
10       internal_delay="5" weight="1" threshold="-20"/>
11      <connections>
12        <connection id="0" pre_cell_id="0" post_cell_id="1"/>
13        <connection id="1" pre_cell_id="1" post_cell_id="1"/>
14        <connection id="2" pre_cell_id="1" post_cell_id="2"/>
15        </connections>
16    </projection>
17  </projections>
18 </networkml>
```

On the second network example (Figure 3.4 - right), there are two populations of neurons randomly distributed in the volume of the two spheres. The first population is defined on Listings 3.4 (ln4-ln12), where the user specifies how cells will be distributed in space, the population size and the region in which cells will be placed. In this demo, cells of *PopA* are randomly distributed (Listings 3.4 (ln6)) in a spherical region (ln8), which has its center at position $(x, y, z) = (0, 0, 0)$ and its diameter of $100\mu m$ (ln9). Length units are defined at the begining of every document.

Figure 3.4 (right) also shows an arrow from *PopA* to *PopB* representing connections between them. As mentioned before, a *projection* encapsulates a set of connections between populations of neurons. However, in this second example, not only neurons are placed following an algorithmic template, but also the connectivity between them. In this demo, Listings 3.4 defines a projection from line 29 to line 34, where the source population is *PopA* and the target population is *PopB*. Such *projection* follows a pattern defined at line 32, where the direction of the generated connectivity goes from pre (synaptic) to post (synaptic) populations; some cells from source population send signals by 3 connections and some target cells receives signals from up to 2 connections.

All elements that was previously shown in leves 1, 2 and 3, can be compiled into three main elements:

1. ***population*** → specifies a set of cells, of a specific type. Optionally, it is

Table 3.3: NetworkML simple code snippet - *explicit list*

Name	**simple_net**
Populations	
Name	**PopA**
Cell Type	**CellA**
2 Instances	**0: (0, 0, 0)** **1: (10, 0, 0)**
Name	**PopB**
Cell Type	**CellA**
3 Instances	**0: (0, 100, 0)** **1: (10, 100, 0)** **2: (20, 100, 0)**
Projections	
Units	**Physiological Units**
Projection	**NetworkConnection**
From	**PopA**
To	**PopB**
Synaptic properties	Type: **DoubExpSynA** Delay: **5 ms** (internal) Weight: **1** Threshold: **-20 mV**
3 connection instances	0: From segment **0** on source cell **0** to segment **1** on target cell **1** 1: From segment **0** on source cell **1** to segment **0** on target cell **1** 2: From segment **0** on source cell **2** to segment **1** on target cell **1**

Table 3.4: NetworkML simple code snippet - *algorithmic template*

Name	**simple_net**
Populations	
Name	**PopA**
Cell Type	**CellA**
200 Instances	**0: (-1, -15, -20)** **...** **199: (20, 49, 23)**
Name	**PopB**
Cell Type	**CellA**
100 Instances	**0: (58, -15, -20)** **...** **99: (142, 20, 3)**
Projections	
Units	**Physiological Units**
Projection	**NetworkConnection**
From	**PopA**
To	**PopB**
Synaptic properties	Type: **DoubExpSynA** Delay: **5 ms** (internal) Weight: **1** Threshold: **-20 mV**
N connection instances	Considering the pattern, $0 <= N <= 200$.

possible to define their locations in a 3D space and a template for auto generation;

2. ***projection*** → defines the set of synaptic connections between two populations or within a single population. Optionally, it is possible to define a generation pattern;

3. ***network*** → encapsulates populations and projections into a single unit with simulation input and output.

All other elements and parameters that were not described here, can be found at NeuroML documentation (CANNON et al., 2012), but are not mandatory for defining functional entities that will be presented next.

Listing 3.4: NetworkML simple code snippet - *algorithmic template*

```
<networkml id="simple_net" length_units="micrometer">
  <populations>
    <population name="PopA" cell_type="CellA">
      <pop_location>
        <!-- A number of cells are arranged randomly in 3D space
         in a spherical region-->
        <random_arrangement>
          <population_size>200</population_size>
          <spherical_location>
            <meta:center x="0" y="0" z="0" diameter="100"/>
          </spherical_location>
        </random_arrangement>
      </pop_location>
    </population>

    <population name="PopB" cell_type="CellB">
      <pop_location>
        <!-- A number of cells are arranged in 3D space in a
         spherical region-->
        <random_arrangement>
          <population_size>50</population_size>
          <spherical_location>
            <meta:center x="100" y="0" z="0" diameter="100"/>
          </spherical_location>
        </random_arrangement>
      </pop_location>
    </population>
  </populations>

  <projections units="Physiological Units">
    <projection name="NetworkConnection" source="PopA"
```

```
32      target="PopB">
33       <synapse_props synapse_type="DoubExpSynA"
34        internal_delay="5" weight="1" threshold="-20"/>
35       <connectivity_pattern>
36         <per_cell_connection direction = "PreToPost"
37          num_per_source="3" max_per_target = "2"/>
38       </connectivity_pattern>
39     </projection>
40   </projections>
41 </networkml>
```

Level 4: Functional entities and connectivity

The fourth level of CircuitML defines new components to allow the specification of system modules and the connectivity between them. This level has two main purposes: (1) to define CircuitML, and (2) to extend NetworkML providing mecanisms to encapsulate networks and their connectivities into functional modules with standardized interfaces. At this level, brain areas can be specified by local processing units (LPU), where each unit stands for a particular function.

Local processing units can be compared to chips in a circuit. Each chip has its own internal functionality, which is independent of the external circuit, and has its own standardized interface. Although chips can be very simple either in its internal circuits or in its functionality, with a small number of elementary kinds of chips combined, its possible to create complex systems with a great variety of functions.

CircuitML implements four core elements, which are depicted in Figure 3.5:

1. ***lpu*** → encapsulates a functional unit with an interface exposing input and output neurons;

2. ***subcircuit*** → encapsulates smaller circuit parts for reuse inside an LPU. Subcircuits may contain other nested subcircuits, neurons, synapses, and other NeuroML elements;

3. ***interface*** → generates externally accessible names for neurons comprised by constituent sub-circuits;

4. ***connectivity*** → describes synaptic connections between two LPUs.

Figure 3.5 depicts CircuitML new components and its hierarchical structure. At the top level (excluding the *circuitml*) lie the ***lpu*** element,

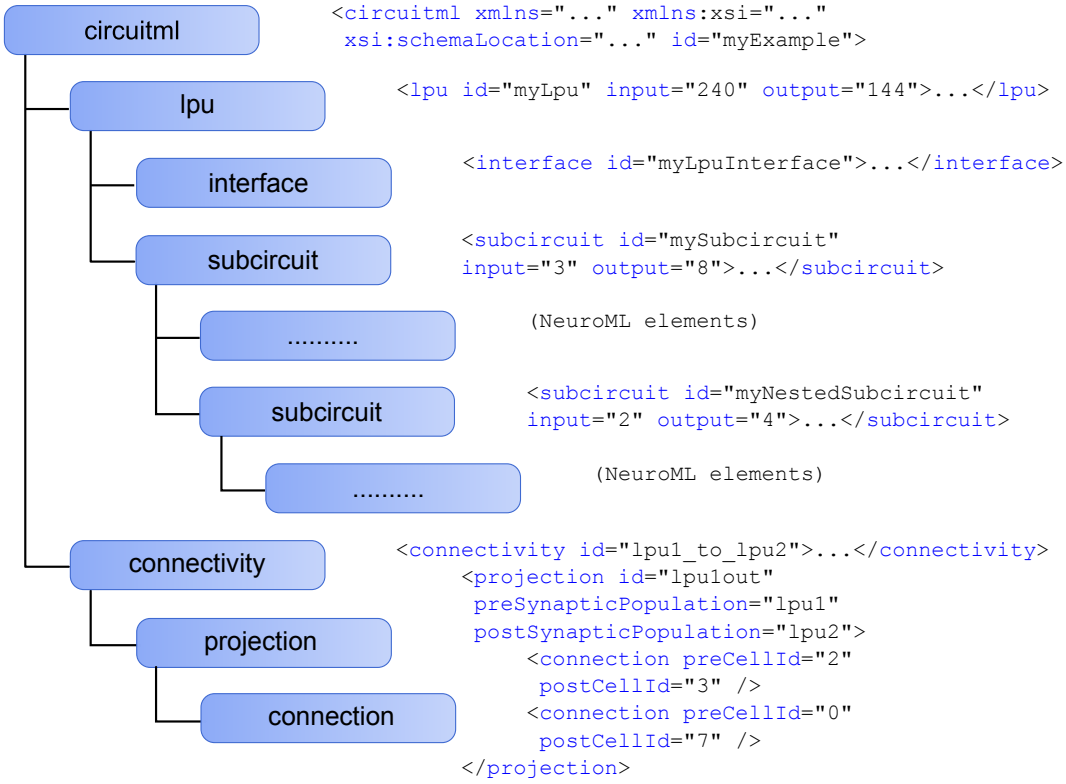

Figure 3.5: Hierarchical structure of CircuitML components.

which encapsulates all circuitry, and the ***conectivity*** element that wraps the connectivity between two LPUs. In order to create such link, LPUs have to comprise an ***interface*** element containing a list of internal neurons that will be handled by the system as "public" and visible to other LPUs.

The ***interface*** element can be understood as a map between the outside of an LPU and its inner circuits. Listings 3.5 shows an example of an interface exposing 4 neurons, respectively, 2 input ports and 2 output ports. In line 4, input 0 exposes neuron_0 from population "my_pop", and in line 6, neuron_3 output is exposed by the interface port 2. Notice that, in this example, neuron_2 of the same population is inaccessible from the outside, because it is not listed in "my_interface".

Listing 3.5: Example of an interface

```
1 <circuitml id="my_example">
2   ...
3   <interface id="my_interface">
4     <port id="in_0" in="0", out="my_pop/0"/>
5     <port id="in_1" in="1", out="my_pop/1"/>
6     <port id="out_0" in="my_pop/3" out="2"/>
7     <port id="out_1" in="my_pop/4" out="3"/>
8   </interface>
9   ...
```

```
10 </circuitml>
```

LPUs may comprise not only networks of neurons, but also smaller functional structures, containing or not neurons, called ***subcircuit***. They are very similar to a *network* element except that they can encapsulate external components, such as filters, pre and post processors, etc. Right now, the *subcircuit* supports the same kind of elements that *lpu* does, but in future releases, it will be possible to add inner elements other than neurons or synapses. This is an important feature for designing a sensory system, for example, that needs pre-processing of analogue inputs or some special spiking encoding.

Notice that *subcircuit* elements are not LPUs, either because they have to expose all inner elements (removing the data abstraction), or because they have no complete functionality (disregarding the main point of an LPU).

A simple example of an ***lpu***, is depicted in Figure 3.6, which shows a basic module called *partner_detector*, containing three main blocks: (1) **TEM** (Time Encoding Machine) that encodes the input analogue signal to spikes, (2) an **analyser**, that processes the encoded signal from TEM and (3) a **pre_motor** unit that sends information to the outside of the LPU. In this example, the "partner_detector" LPU classifies inputs into two classes: *valid* and *invalid*. An input is considered *valid* if it was generated by an animal of the same species, and an *invalid* if otherwise.

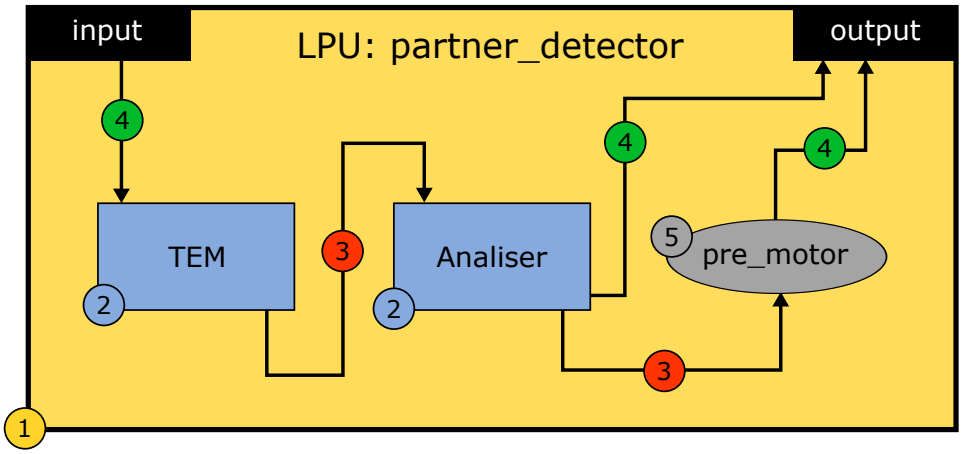

Figure 3.6: Example of a fictitious LPU (yellow box) that detects if some input matches the type expected. The blue boxes comprise circuits with some function associated and the gray ellipse represents a population of cells that sends some info to the LPU output. Green and red bolls stand for projections between interface ports and inner circuits, and between inner circuits, respectively.

The first block (Figure 3.6 - TEM blue box) contains a Time Encoding

Machine (TEM) (LAZAR; PNEVMATIKAKIS; ZHOU, 2010) that convert analogue signal coming from sensors, for example, to spikes that is the same language that neurons inside this LPU understand. Considering that there is an element called **tem** that implements a Time Encoding Machine, the **TEM** block could be represented as in Listings 3.6, where the "tem_block" comprises a "my_tem" element with 1 input and 1 output.

The output of "my_tem" goes to the "Analyser" block (Figure 3.6 - Analyser blue box), which is a network of IAF cells that processes the signal and classifies if the input signal belongs to a valid partner or not. In Listings 3.7, the analyser block comprises, respectively, two populations of IAF cells: first_step and second_step (lines 6-7). In line 9, a projection connects both populations following a full connected pattern.

Listing 3.6: Example of a TEM block

```
1 <circuitml id="tem_block">
2   <subcircuit id="tem_block">
3     <tem id="my_tem">
4       <inputs>
5         <port id="in_0" />
6       </inputs>
7       <outputs>
8         <port id="out_0" />
9       </outputs>
10    </tem>
11  </subcircuit>
12 </circuitml>
```

Listing 3.7: Example of the Analyser block

```
1 <circuitml id="ana_block">
2   <refractiaf id="cell_A" threshold="-40mV"
3    refractoryPeriod="5ms" capacitance="1nF" vleak="-80mV"
4    gleak="100pS" vreset="-70mV" v0="-70mV" deltaV="10mV" />
5   <refractiaf id="cell_B" threshold="-35mV"
6    refractoryPeriod="6ms" capacitance="2nF" vleak="-80mV"
7    gleak="89pS" vreset="-70mV" v0="-70mV" deltaV="10mV" />
8   <expOneSynapse id="syn_1" gbase="0.5nS" erev="0mV"
9    tauDecay="3ms" >
10  <network id="analyser">
11    <population id="first_step" cell="cell_A" size="50">
12    <population id="second_step" cell="cell_B" size="2">
13    <!-- Since there is no connectivity info inside this
14     projection, it will be assumed a full connected pattern -->
15    <projection id="first_to_second"
16     presynapticPopulation="first_step"
```

```
17      postsynapticPopulation="second_step" synapse="syn_1" />
18    </network>
19 </circuitml>
```

The last block (Figure 3.6 - gray ellipse) comprises a population of neurons that will send their output signals to the interface of LPU. Listings 3.8 show all elements instantiated inside the LPU "partner_detector". In lines 3 and 4 of Listing 3.8 there is a new element, called **Include**, that substitutes the regular **include**. This new element indicates to libCircuitML parser that the included code must be preprocessed and all internal connectivity must be merged with the code that is importing it. In the end of this process, each LPU will comprise a single internal connectivity matrix.

Listing 3.8: Example of the lpu depicted in Figure 3.6

```
1 <circuitml id="partner_detector">
2   <!-- Including external references -->
3   <Include href="tem_block.xml" />
4   <Include href="ana_block.xml" />
5   <!-- NeuroML elements  -->
6   <iafCell id="pm_cell" reset="-50mV" C="0.03nF" thresh="-25mV"
7    leakConductance="1uS" leakReversal="-50mV" />
8   <expOneSynapse id="pm_syn" erev="20mV" gbase="65nS"
9    tauDecay="3ms" />
10  <expOneSynapse id="ana_syn" erev="15mV" gbase="85nS"
11   tauDecay="2ms" />
12  <!-- LPU specification -->
13  <lpu id="partner_detector">
14    <!-- Interface -->
15    <interface id="my_interface">
16      <port id="analogue_input" in="0",
17       out="tem_block/tem_block/0"/>
18      <port id="valid_out" in="ana_block/analyser/second_step/0"
19       out="1"/>
20      <port id="invalid_out"
21       in="ana_block/analyser/second_step/1" out="2"/>
22      <port id="pm_out_0" in="pre_motor/0" out="3"/>
23      <port id="pm_out_1" in="pre_motor/1" out="4"/>
24       ...
25      <port id="pm_out_28" in="pre_motor/28" out="31"/>
26      <port id="pm_out_29" in="pre_motor/29" out="32"/>
27    </interface>
28    <!-- Populations -->
29    <population id="tem" structure="tem_block" size="1"/>
30    <population id="analiser" structure ="ana_block" size="1" />
31    <population id="pre_motor" component="pm_cell" size="30" />
```

```
32      <!-- Projections -->
33      <projection id="tem_analyser"
34       presynapticPopulation="tem_block/tem_block"
35       postsynapticPopulation="ana_block/analyser/first_step"
36       synapse="ana_syn" />
37      <projection id="analyser_pm"
38       presynapticPopulation="tem_block/tem_block"
39       postsynapticPopulation="ana_block/analyser/first_step"
40       synapse="ana_syn" />
41      </projection>
42    </lpu>
43 </circuitml>
```

The last component presented here is the ***connectivity*** element, which can be understood as the glue between LPUs. Since elements such as *projections* and *connections* are not able to exist outside the LPU, the *connectivity* element acts as a wrapper for *projections* and *connections* that will link not cells or synapses, but interface ports. In order to understand how such element works, let's consider a new system, depicted in Figure 3.7, where the "partner_detector" (Figure 3.6 and Listings 3.8) receives an analogue signal from an external sensor (Figure 3.7 - top box) and it sends the valid/invalid signals to another LPU that will somehow convert it into a True/False result.

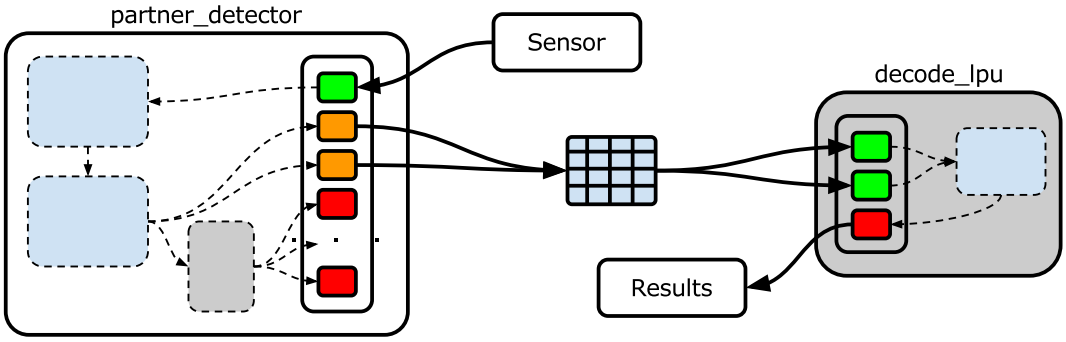

Figure 3.7: Two LPUs inter-connected by a connectivity element. Inner circuits are depicted with dashed borders, because they are not visible from the outside of the LPUs.

The *connectivity* on lines 5 to 8 specifies how both LPUs, "partner_detector" and "decode_lpu", will be connected, i.e., the synapse of each connection and which port to link.

Listing 3.9: Example of two LPUs inter-connected (Figure 3.7)

```
1 <circuitml id="lpus">
2   <include href="partner_detector.xml" />
3   <include href="decode_lpu.xml" />
4   <expOneSynapse id="my_syn" erev="20mV" gbase="65nS"
```

```
5    tauDecay="3ms" />
6   <connectivity id="pd_to_decoder" lpu1="partner_detector"
7    lpu2="decode_lpu">
8     <connection from="partner_detector/valid_out"
9      to="decode_lpu/valid" synapse="my_syn"/>
10    <connection from="partner_detector/invalid_out"
11     to="decode_lpu/nvalid" synapse="my_syn"/>
12  </connectivity>
13 </circuitml>
```

3.2. Translating process

The main reason to extend NeuroML instead of creating a complete new language is the fact that, as mentioned in Chapter 2, there are many tools available that support it, which makes the introduction of new applications painless, since people are familiar with the environment. CircuitML translating process uses some tools and concepts developed and implemented for NeuroML, such as LEMS and libNeuroML (GLEESON et al., 2012). This new proposed tool also shares the same goals of NeuroML initiative, which are exchangeable components, simulator independent and accessible to as many researchers as possible. Thus, CML translator and its simulator (next section) can be replaced by any other simulator that supports NeuroML 2; in Chapter 2 there is a list of some simulators. However, the simulator presented in the next section, is part of a bigger project called neurokernel (GIVON; LAZAR, 2012), which uses GPUs to run neural circuits in real-time.

CircuitML is designed to have machine readable definitions of the core model components, to facilitate unambiguous interpretations of the model behavior across implementations. On the implementation side, it means that all CircuitML elements are described explicitly in a *ComponentClass* (instantiations of which are referred to as Components).

Component classes provide a programmatic layer of abstraction, where each element can be inherited. In this way the language can be easily extended, with simulators knowing that any new *ComponentClass* extending synapse, for instance, exposes a current and receives spike events, etc. CircuitML provides LEMS primitives for defining neural circuits and sub-circuits that are endowed with interface ports that enable their connection to other sub-circuits via neural connectivity patterns. CML primitives (Figure 3.5) are written in Python and CUDA, and glued together using PyCUDA (KLÖCKNER et al., 2012), which allows one to access CUDA parallel computation API from Python.

Figure 3.8 illustrates an example of how explicit definitions of a cell model is specified in CircuitML. The behavior of the model component (in this case a simple Hodgkin-Huxley cell model (HODGKIN; HUXLEY, 1952)) is specified in a *ComponentClass* (pink, bottom), with the state variable (V, n, m and h) specified along with their dynamics in time in terms of the fixed parameters of the model.

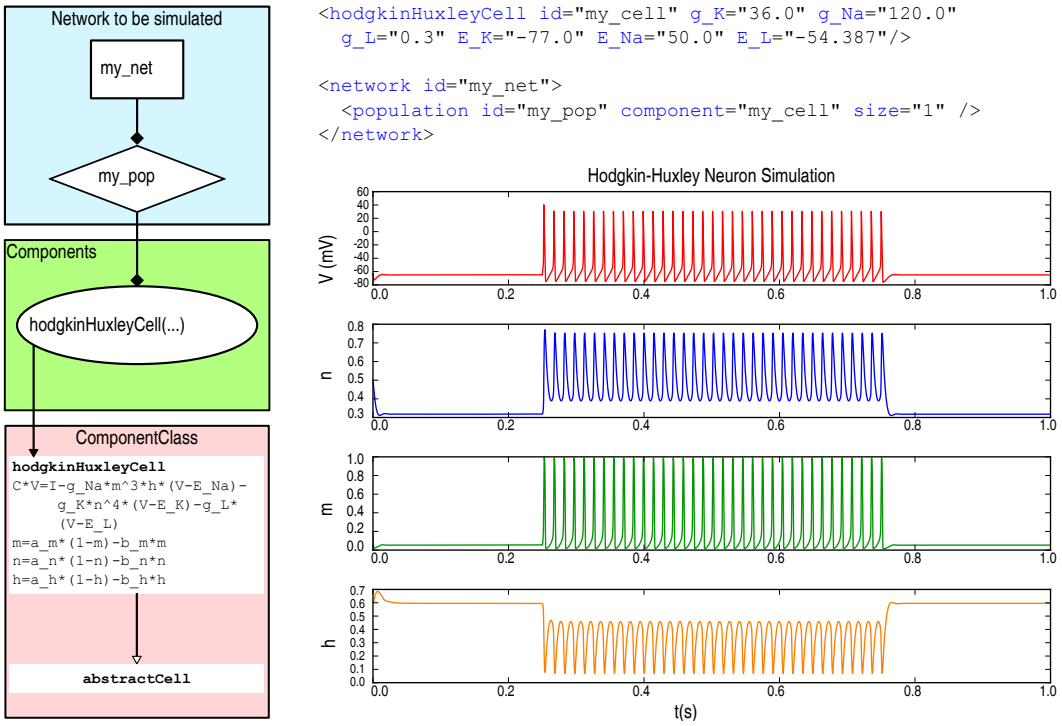

Figure 3.8: Hodgkin-Huxley model cell specified in CircuitML. Left) A network specifies that it contains a single population containing an instance of a *hodgkinhuxleyCell*. The definition for the behavior of this Component is contained in a *ComponentClass*. This graph has been automatically generated from the XML definition of the *ComponentClass* (not shown) and the network definition creating the Component instance, shown on the top-right. Top-right) The XML corresponding to the Component and network. C) The model after being executed by the interpreter, showing behavior of the state variables V (red), n (blue), m (green) and h (orange).

The definition of the *hodgkinhuxleyCell ComponentClass* is inherited from NeuroML and the XML used to create a simple network of one cell is shown in Figure 3.8 (top-right). Other abstract neuron models such as Integrate and Fire (IAF) neurons, and two state variable extensions of this, such as the Morris-Lecar neuron model (*morrisLecarCell*) (MORRIS; LECAR, 1981), are also currently supported (Figure 3.1).

With CML's programmatic layer, called libCircuitML, it is possible to instantiate elements, such as the one defined in Listing 3.10, directly in Python as displayed in Listing 3.11. It is also feasible to call the CML translator in a

Python interactive console and get all XML specification processed into Python elements - Listing 3.12.

Listing 3.10: Example of a cell instantiation in Python

```
<iafCell id="my_iaf", C="1.0 nF", thresh="-50mV", reset="-65mV",
 leak_conductance="10 nS", leak_reversal="-65mV" />
<expOneSynapse id="my_syn", gbase="65nS", erev="0mV",
 tau_decay="3ms" />
```

Listing 3.11: Example of a cell instantiation in Python

```
my_iaf = IafCell(id="my_iaf", C="1.0 nF", thresh="-50mV",
                 reset="-65mV", leak_conductance="10 nS",
                 leak_reversal="-65mV")
my_syn = ExpOneSynapse(id="my_syn", gbase="65nS", erev="0mV",
                       tau_decay="3ms")
```

Listing 3.12: How to call CML's translating process in Python

```
myLpu = LPU("myExample.xml")
```

libCircuitML components and their relationship are depicted in Figure 3.9, where each box represent a specific component with its respective parameters and lines characterize a relationship between two elements. This connection can be one-to-one, when both interconnected elements can only be connected to each other, or one-to-n, when an element can connect to one or more elements.

3.2.1. Why Python and CUDA

The reasons for implementing the libCircuitML in Python and CUDA is, mainly, to have both portability and performance on the same application. Python is a scripting language that has been used by many years among the scientific community (DAVISON; HINES; MULLER, 2009), having a vast number of open-source and well kept libraries, which facilitate the application development and allows it to run on different systems. On the other hand, because Python code are interpreted scripts, it can be considered a bad idea for high performance computing, since this characteristics adds an overhead on the execution stack and the developer has less control over the compiled objects.

In order to address the performance issue, libCircuitML uses CUDA code wrapped by PyCUDA for simulation purposes. By sending time-critical

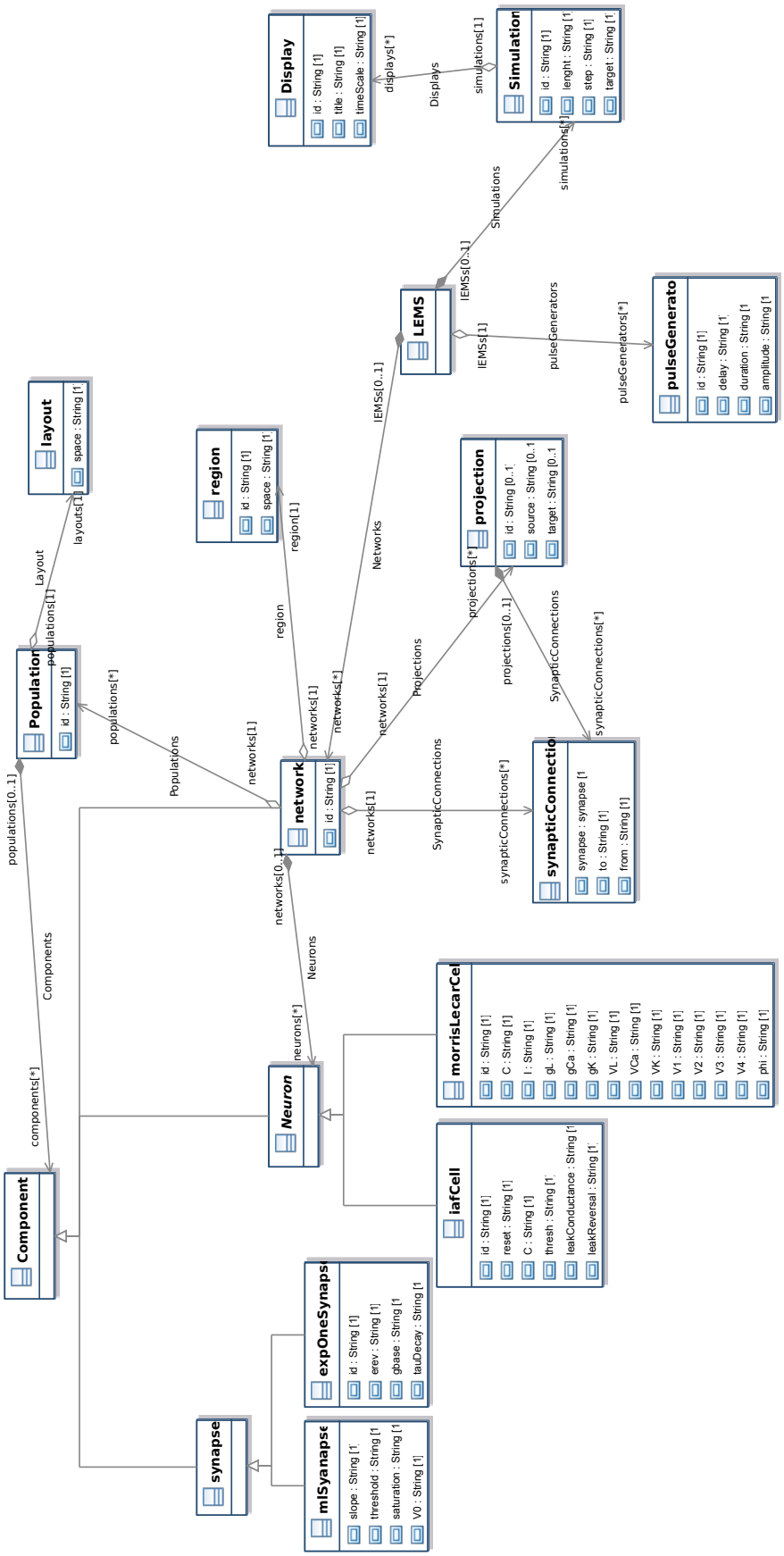

Figure 3.9: libCircuitML class diagram.

functions to the GPU, the most costly portions of the code are processed in parallel by thousands of GPU processing elements. Also, because CUDA part is written in C code, the developer has total control of the compilation process.

3.2.2. Translating steps: from XML to CUDA

The entire translation process is divided into 3 parts: (1) parse the XML file and create Python structures in CPU memory; (2) prepare some of those structures to be loaded into GPU; (3) create the simulation environment in order to run the simulation. Listing 3.12 shows an example of how the parser is called.

The translation process starts by parsing the XML file using an open source XML toolkit called *lxml* (*http://lxml.de/*), which is a Python binding for the C libraries *libxml2* and *libxslt*. After checking that everything is correctly spelt and the XML file is well formed, the parsing step will fully validate the input file's correctness against the CML schema, which means that the validator will check if all components needed are present in order to proper run everything. In general, the most common verifications are:

1. Do all cited components exist? (includes, schemas, cells, sub-circuits, LPUs, etc.)
2. Are all components declared in the correct place?
3. Are all synapses making correct connection?
   (a) Are these cells visible?
   (b) Are these cells inside the same LPU?
4. For each declaration, are all attributes specified with the correct unit (mV, μS, etc.)?

The product of this parser is a flag informing if the document is valid. From an OK flag, the translator will read the specification file and start to instantiate Python objects in CPU memory. Assuming a top-bottom approach, the first elements created are the LPUs, each of which are populated following the same approach, i.e., sub-circuits, populations, projections, etc. After all structures are populated, each module (LPU) comprises the following:

- a list for each type of cell, where each element is a respective cell object with all parameters read from the XML file;

- a spiking storage for the output spikes on the last n dt, where n and dt are, respectively, number of times and time-step size, both simulation parameters;
- a graded potential queue for the output voltages on the last n dt;
- the local connectivity.

Cell lists are basic Python lists, where elements are pointers to cell components, each of which is configured with their own parameters specified on the XML file. Figure 3.10 shows how cells are stored into Python CPU memory.

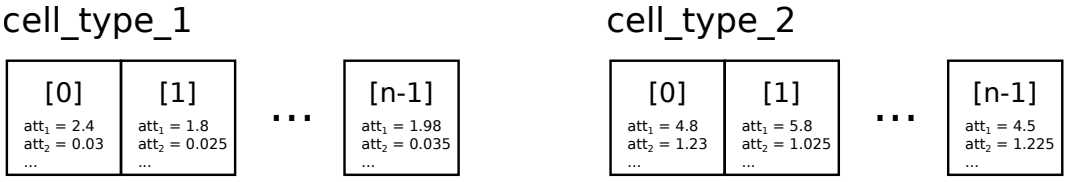

Figure 3.10: How libCircuitML stores cell components in Python CPU memory.

The spiking storage presented here is based on spike container from (BRETTE; GOODMAN, 2011) that minimizes memory usage. In Figure 3.11-A, a dynamic array is an array with unused space (gray squares), which is resized by a fixed factor (in this case, 2). The structure used in this work is the circular array (Figure 3.11-B and Listing 3.13). On this structure, the simulator inserts every time step the indices of all neurons that have spiked (Listing 3.13-ln36), and shift the cursor by the number of spikes (Listing 3.13-ln37). On one hand, the spike container reduces memory usage, since it saves only the indexes of neurons that have spiked, but on the other hand, the algorithm is vectorised over spikes and the length of the circular array depends a priori on the number of spikes produced, which is unknown before the simulation, that is the reason to define a growth policy.

Listing 3.13: Implementation of the circular array and the spike container.

```
1 class CircularVector(object):
2  def __init__(self, n):
3    self.X = zeros(n, dtype=int)
4    self.cursor = 0
5    self.n = n
6
7  def __getitem__(self, i):
8    return self.X[(self.cursor + i) % self.n]
9  def __setitem__(self, i, x):
10   self.X[(self.cursor + i) % self.n] = x
11
```

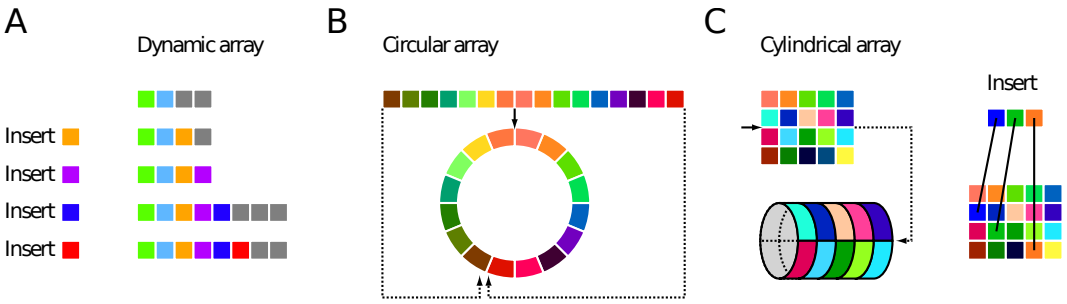

Figure 3.11: Spike container: a circular dynamic array x (outer ring) stores the indices of all neurons that spiked in previous time steps, where the cursor indicates the current time step. Another circular array (inner ring) stores the location of the previous time steps in the dynamic array. Extracted from (BRETTE; GOODMAN, 2011).

```
12  def endpoints(self, i, j):
13    return (self.cursor + i) % self.n, (self.cursor + j) % self.n
14
15  def __getslice__(self, i, j):
16    i0, j0 = self.endpoints(i, j)
17    if j0 >= i0:
18      return self.X[i0:j0]
19    else:
20      return hstack((self.X[i0:], self.X[:j0]))
21
22  def __setslice__(self, i, j, W):
23    i0, j0 = self.endpoints(i, j)
24    if j0 > i0:
25      self.X[i0:j0] = W
26    elif j0 < i0:
27      self.X[i0:] = W[:self.n - i0]
28      self.X[:j0] = W[self.n - i0:]
29
30 class SpikeContainer(object):
31  def __init__(self, n, m):
32    self.S = CircularVector(n + 1)
33    self.ind = CircularVector(m + 1)
34
35  def push(self, spikes):
36    ns = len(spikes)
37    self.S[0:ns] = spikes
38    self.S.cursor = (self.S.cursor + ns) % self.S.n
39    self.ind.cursor = (self.ind.cursor + 1) % self.ind.n
40    self.ind[0] = self.S.cursor
41
42  def __getitem__(self, i):
43    j = self.ind[-i - 1] - self.S.cursor
44    k = self.ind[-i]-self.S.cursor + self.S.n
```

```
45    return self.S[j:k]
46
47  def __getslice__(self, i, j):
48    k = self.ind[-j] - self.S.cursor
49    l = self.ind[-i] - self.S.cursor + self.S.n
50    return self.S[k:l]
```

The graded potential queue is based on (BRETTE; GOODMAN, 2011)'s cylindrical array (Figure 3.12), which is a two dimensional array that wraps around at the end, and which can be rotated at essentially no cost. It has the property that $X[t + M, i] = X[t, i]$, where i is a neuron index, t is the time index, and M is the size of the circular dimension. The difference between (BRETTE; GOODMAN, 2011)'s implementation and the one in libCircuitML is the fact that here the cylinder is not used to vectorize the insertion, but to store the output voltage of all non-spiking cells. Therefore, the cylinder height in libCircuitML is the number of non-spiking neurons.

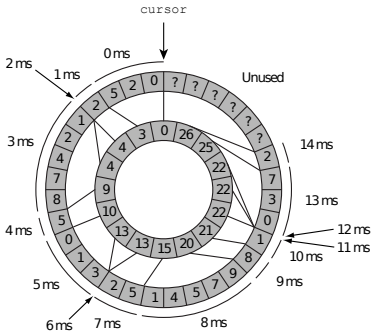

Figure 3.12: Circular array is implemented as an array with a cursor indicating the current position. A cylindrical array is implemented as a two-dimensional array with a cursor indicating the current position. Insertion can be easily vectorized. Extracted from (BRETTE; GOODMAN, 2011).

The local connectivity matrix (M) must store, for each neuron, the list of target neurons, synaptic weights, transmission delays and other parameters that depend on the synapse type. In order to increase the reading speed, each parameter must be stored as contiguous vectors in memory. Figure 3.13(A) depicts a dense representation of M where columns are the pre-synaptic neurons, the rows are the post-synaptic neurons and each element is the synaptic weight. In order to extend such representation to a greater number of parameters, each element in M was replaced by a list of parameters creating a n-dimensional matrix, where $n > 1, n \in \mathbb{N}$ is the number of parameters for a specific type of neuron, as depicted in Figure 3.13(B).

Usually, neurons respect some kind of connectivity pattern among them (MASUDA-NAKAGAWA; TANAKA; O'KANE, 2005; VOSSHALL;

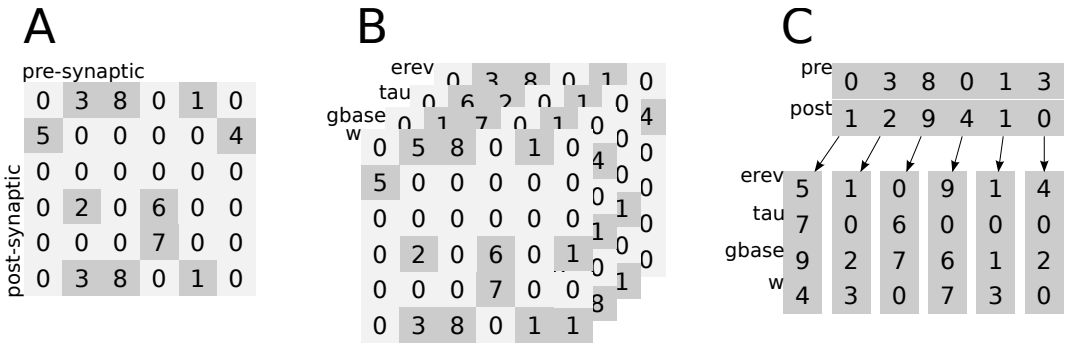

Figure 3.13: Matrix structures for synaptic connectivity. (A) dense representation of the local connectivity matrix, (B) connectivity with many parameters, (C) coordinate list format of the connectivity.

STOCKER, 2007), but they rarely are densely interconnected, and the matrix representation can be sparse. To reduce memory usage and improve random access times in GPU, libCircuitML creates a coordinate list from M dense format - Figure 3.13(C). On Python environment, coordinated lists stores synaptic objects.

The last step for the translator is to prepare cells, connectivity and additional processes to be transferred to GPU before the simulation, since all of them are objects on Python environment making them easier to modify, replace and visualize (regarding the modeler perspective). Although objects have many advantages, in general, object-oriented programming paradigm introduces computational overhead, which is not desirable in a heavy simulation process.

The libCircuitML GPU structures are organized the same way as the Python ones are, but they are stored in memory differently, as follows:

- one structure per cell type, where each structure's attribute is a list as shown in Listing 3.14;
- a spiking storage array (Figure 3.11);
- a graded potential queue as described before;
- the local connectivity as depicted in Figure 3.13(C).

Listing 3.14 shows an example of a cell definition in GPU memory as a C struct with arrays as attributes instead of an array of structures. This last one would be the most direct conversion from an array of cells objects, however, because of reading pattern during the simulation, such choice would generate not coalesced[1] accesses.

[1]Coalesced memory access or memory coalescing refers to combining multiple memory accesses into a single transaction.

Listing 3.14: Example of a GPU cell structure

```
1 struct IafCell{
2     double* conductance;
3     double* c;
4     double* vth;
5     double* resetV;
6     double* leakReversal;};
```

3.3. Computing the virtual brain

The emulator presented here is only a proof-of-concept application that allows testing during the development of CircuitML. There is no compromise to create a fully functional simulator or the fastest one, since there are many other initiatives to do so, as mentioned in Chapter 2. Although there are many other simulators available, the only one that is focused on the simulation of flies is Neurokernel, which is not fully implemented yet, and that is the main reason to build one here. Because the focus here is the simulation of a fruit fly brain, the constraints at the architectural level are related to the fly brain, which are:

- 10^5 neurons to be emulated;
- Up to 10^{10} synapses - based in mammalian cells;
- 8 neurotransmitters;
- Brain divided in Neuropils with up to 6×10^4 neurons each.

The Drosophila brain comprises approximately 150,000 neurons divided, until now, into two neuron models: (1) integrate-and-fire and (2) Morris-Lecar. Both models are described as differential equations and discrete events (spikes or $V(t)$ for non-spiking neurons), so that the simulation time cost (C_T) can be divided into the cost of updating neuron states (U) and the propagation of information (P) (spikes or $V(t)$), as shown in Equation 3.1 for spiking neurons and in Equation 3.2 for non-spiking neurons (BRETTE; GOODMAN, 2011). It is important to mention that c_U and c_P is related, respectively, to the neuron and the synapse type for both equations. Neurotransmitters do not affect these costs, but only synapse parameters.

$$C_T^{spk} = \left[c_U^{spk} \times \frac{N^{spk}}{dt}\right]_{update} + \left[c_P^{spk} \times F \times N^{spk} \times p^{spk}\right]_{propagation} \quad (3.1)$$

where c_U^{spk} is the cost of one update and c_P^{spk} is the cost of one spike propagation, N^{spk} is the number of spiking neurons, p^{spk} is the number of synapses per neuron, F is the average firing rate and dt is the time step (the cost is for one second of biological time).

$$C_T^{gp} = \left[c_U^{gp} \times \frac{N^{gp}}{dt}\right]_{update} + \left[c_P^{gp} \times N^{gp} \times p^{gp}\right]_{propagation} \tag{3.2}$$

where c_U^{gp} is the cost of one update of a non-spiking cell and c_P^{gp} is the cost of one $V(t)$ propagation, N^{gp} is the number of non-spiking neurons and p^{gp} is the number of synapses per neuron. So the total cost (C_T) for the simulation of all neurons in a non parallel environment is:

$$C_T = C_T^{spk} + C_T^{gp} \tag{3.3}$$

In addition to the update and the propagation times, there is also the interpretation time (c_I), since libCircuitML is written in Python. To address this issue, (BRETTE; GOODMAN, 2011) proposes vectorized operations so that c_I could be hidden. Vectorized calculations for both neurons and synapses produces a new function cost (Equation 3.4), where N and p are directly proportional to how c_I impacts on simulation speed.

$$C = \left[\frac{c_U \times N + c_I}{dt}\right]_{update} + \left[F \times N \times (c_P \times p + c_I)\right]_{propagation} \tag{3.4}$$

Although the emulator presented in this work is only for testing purposes, it still has some design goals: (1) scalable, (2) fast, (3) efficient and (4) portable. Since it was written in Python, the emulator can be executed in Windows, Apple OS and Linux machines, what answers the last goal.

In order to be scalable, the internal structure of the emulator has to address some aspects of memory and execution, such as memory availability and massive parallel execution synchronized among computational nodes. Since the subject being emulated in this work is, by definition, a multi-core computer, the brain can be easily split into smaller portions and distributed among computer nodes. CUDA was the first choice considering that the architecture allows developers to dispatch multiple executions synchronized among thousands of processors. If the brain becomes bigger, the emulator can

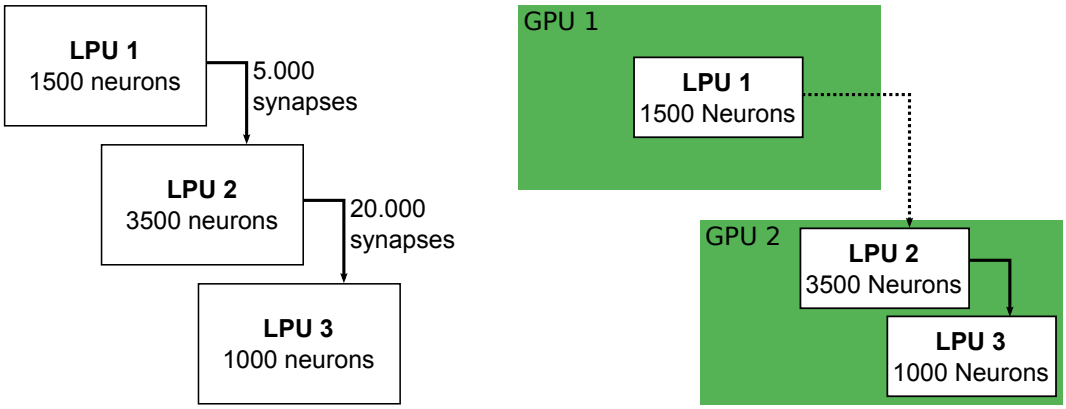

Figure 3.14: Reallocation example (right): Consider a system with two GPUs that can store up to 5000 neurons and run them at once. Three LPUs are specified with some connectivity between them. When one calls *run* from the emulator passing all three LPUs, libCircuitML will allocate them (left picture), because LPUs 2 and 3 have more interconnections and the emulator will always try to put more interconnected LPUs together.

allocate new brain parts (LPUs) into new hardware transparently by the user perspective. Figure 3.14 and Figure 3.15 show how the emulator reallocate the data among new resources.

One advantage of this approach is the fact that fly brain is divided into functional bocks (LPUs), and those LPUs are usually less densely interconnected than the local connectivity inside them. This characteristic allows the system to have a tip on how the specification would be eventually partitioned, if the input file had more than one LPU (CHIANG et al., 2011).

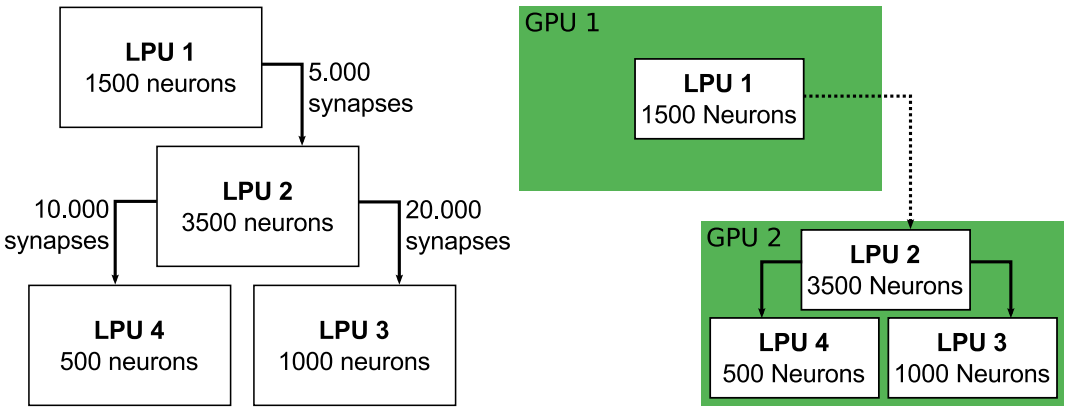

Figure 3.15: Reallocation example (part 2): Now, a new LPU (LPU 4) is added to the system specified in in Figure 3.14, but because it has many connections with LPU 2, the system will put it on the second GPU.

By choosing GPUs as computer nodes, both design goals (2) and (3) are addressed. For problems where many parts are independent, as the emulation of many cells, GPUs can run much faster than CPUs (CHEVITARESE; SZWARCMAN; VELLASCO, 2012; SILVA et al., 2010).

The two lists with libCircuitML functional and non-functional requirements are:

Functional requirements

1. Visualization of brain module network and the modules associated with different sensory subsystems;
2. Simulation of sensory system modules on a single GPU;
3. Automated resource allocation into CUDA streams;
4. Reading/writing of arbitrarily long inputs/outputs from/to disk

Non-Functional requirements

1. Concurrent emulation of multiple modules on a single-GPU ecosystem

The visualization is partially addressed with neuroConstruct (GLEESON; STEUBER; SILVER, 2007), since it provides a good visualization features, as mentioned on Chapter 2. Although neuroContruct does have a rich graphical support, Neuroptikon (*http://openwiki.janelia.org/wiki/display/neuroptikon/Home*) offers a circuit format view that allows the visualization of LPUs and inner structures as shown in Figure 3.16. Functional requirements 2, 3 and 4, and the non-functional one are discussed on the next chapter.

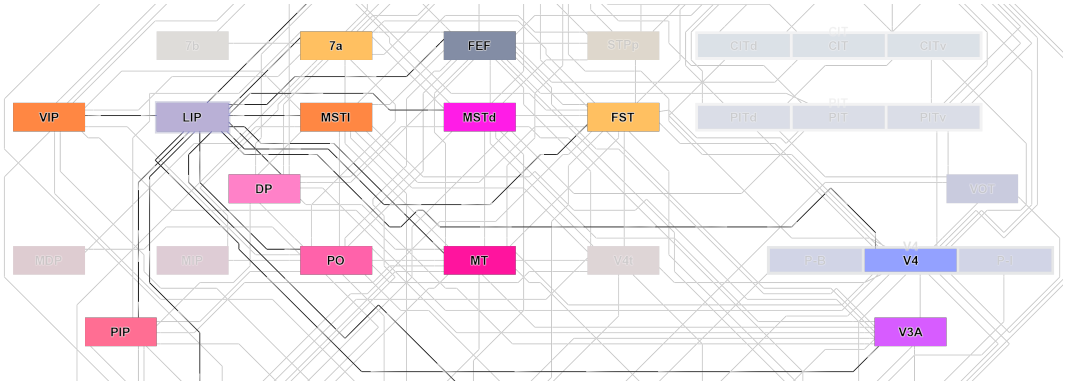

Figure 3.16: Local connectivity of regional network.

GPU emulation process

The emulation part is not the main focus of this work, since it was envisioned to work with Neurokernel, which was discussed before. However, because Neurokernel has been developed in parallel with CircuitML and its parser, in order to test some features of CircuitML, it was implemented a very simple emulator. The emulator presented here is written in Python and CUDA using a wrapper package called PyCUDA. The reason for using CUDA is easy

to understand, since it is very efficient to run all neurons and synapses in parallel.

4
Results

At this point, it is crucial to understand how interconnected LPU can facilitate the specification of complex systems. In order to be able to model an entire fly brain and then simulate such system, it is imperative to have a good representation to support annotation and modeling (ARMSTRONG et al., 2009). Furthermore, recent progress in neuroinformatics confirmed that comprehensive brain wiring maps are needed to formulate hypotheses about how information flows and is processed inside a brain (CHIANG et al., 2011). These are the two main motifs behind the development of CircuitML: functional modularity and connectivity.

To illustrate how functional structures work, this chapter discusses the modeling of the sensory systems of the fruit fly: (1) the olfactory system and (2) the visual system. The sensory system of the fruit fly have been studied by various groups (FISCHBACH; DITTRICH, 1989; ZHU et al., 2009; CARON et al., 2013; VOSSHALL; STOCKER, 2007), but both the visual and the olfactory systems in this chapter are specified from Bionet (Columbia University) data. In order to better understand those systems, in the beginning of each chapter there is an introduction about the related system.

Notice that, since the focus here is on how to better model neural circuits, it is hard to compare the results of a circuit simulation, for example. Because of this, at the end of each section of this chapter, there will be a discussion on how CircuitML can make the modeling process easier, simpler, or even clearer.

4.1. Olfactory organs of adult flies

Like all other higher animals, flies sense odors with olfactory organs located exclusively on the head, and all fly ORNs (Olfactory Receptor Neurons or OSN as Olfactory Sensory Neurons) are housed in the third segment of the antenna and in the maxillary palp (Figure 4.1 right). These two organs are covered with specialized hairs, called sensilla, which protect the ORNs from the insults of the external environment. The morphology of neurons that underlies fly's olfactory system is quite similar to the ones underlying ORNs

in a mammalian nose (VOSSHALL; STOCKER, 2007).

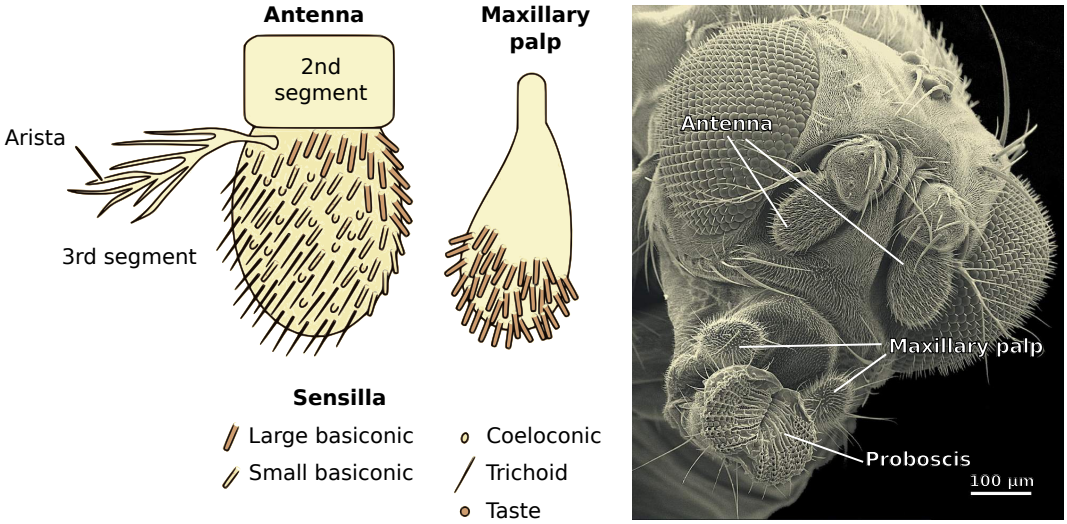

Figure 4.1: Neuroanatomy of the peripheral fly olfactory and gustatory systems. Right: scanning electron micrograph of a fly head, indicating the major chemosensory organs. Image extracted from Rochester University Website. Left: schematic of the exterior surface of the olfactory organs. Part of an image extracted from (VOSSHALL; STOCKER, 2007)

At the architectural side, each ORN extends a sensory dendrite ending in ciliated projections into the shaft of the sensillum, which houses between one and four ORNs that are surrounded by support cells. Such configuration makes electrophysiology possible for a given sensillum, since it allows a sharp electrode to be inserted into the base of the sensillum that can measure extracellular activity of the ORNs in response to a panel of odorous stimuli. A measurement technique can be found in details in (GOLDMAN et al., 2005; BRUYNE; CLYNE; CARLSON, 1999), where they examine odor coding in the fly's maxillary palp and construct a receptor-to-neuron map for an entire olfactory organ in Drosophila.

The antenna is covered with three different types of sensilla: (1) basiconic, (2) trichoid, and (3) coeloconic. They differ in size, morphology (Figure 4.1 left) and the types of substances detected by the underlying neurons (Figure 4.1 left/below). Those types of sensilla are distributed in a stereotyped and bilaterally symmetric pattern, with large basiconic sensilla clustered at the medial-proximal side of the antenna and trichoid sensilla clustered at the lateral-distal edge. In total, there are between 1100-1250 ORNs in each antenna that extends its dendrites in the AL (VOSSHALL; STOCKER, 2007).

4.1.1. Adult olfactory pathway

Drosophila has 39 ORs projecting to 43 morphologically defined glomeruli in the AL in a 1 OR to 1 glomerolus rule (approximately) (VOSSHALL; STOCKER, 2007). OR projections converges onto a common AL glomerulus and defines the input properties of the glomeruli (Figure 4.2). The two major target neurons of the ORNs are local neurons (LNs), which provide "horizontal" connections among glomeruli (primary center - Figure 4.2), and cholinergic projection neurons (PNs) (primary to secondary centers - Figure 4.2) most of which link individual glomeruli "vertically" with two higher olfactory centers (secondary centers - Figure 4.2), the mushroom body (MB) and the lateral horn, as shown in Figure 4.2. The MBs are integrative centers controlling various functions such as olfactory learning, other forms of learning, locomotor activity, male courtship behavior, and sleep (VOSSHALL; STOCKER, 2007). In contrast, the lateral horn seems to be involved in experience-independent odor recognition (VOSSHALL; STOCKER, 2007). However, those higher centers are not covered in this chapter, since our focus here is on how the fly senses odors.

Exactly functions performed by LNs and PNs are unclear, but whole-cell patch recordings demonstrated that PNs are more broadly tuned and display more complex firing patterns in terms of temporal structure than do the ORNs, which is believed to provide the substrate PNs' tasks (VOSSHALL; STOCKER, 2007). Some LNs (GABAergic) receive excitatory input from ORNs and PNs and establish inhibitory synapses with both afferents and PNs. However, the major role that seems to be accomplished by this intricate LN network is to synchronize PN activity, either within a given glomerulus or between PNs innervating different glomeruli (VOSSHALL; STOCKER, 2007).

Many of these anatomical and functional features are shared by the mammalian olfactory system, apart from the glomerular convergence principle of ORNs (Figure 4.2). Both the insect AL and the mammalian olfactory bulb are characterized by inhibitory LNs whose main task seems to be the extraction of behaviorally relevant information from the incoming signals by changing their temporal structure. Given this surprising conservation, the advantage of the numerically much-reduced insect olfactory system as a model is evident. (VOSSHALL; STOCKER, 2007)

Summarizing, the olfactory systems of Drosophila flies is useful model because of its reduced cell numbers and its similar design with the mammalian olfactory system, which may reflect common functional constraints (VOSSHALL; STOCKER, 2007); the olfactory sensory map in those 43 ALs is

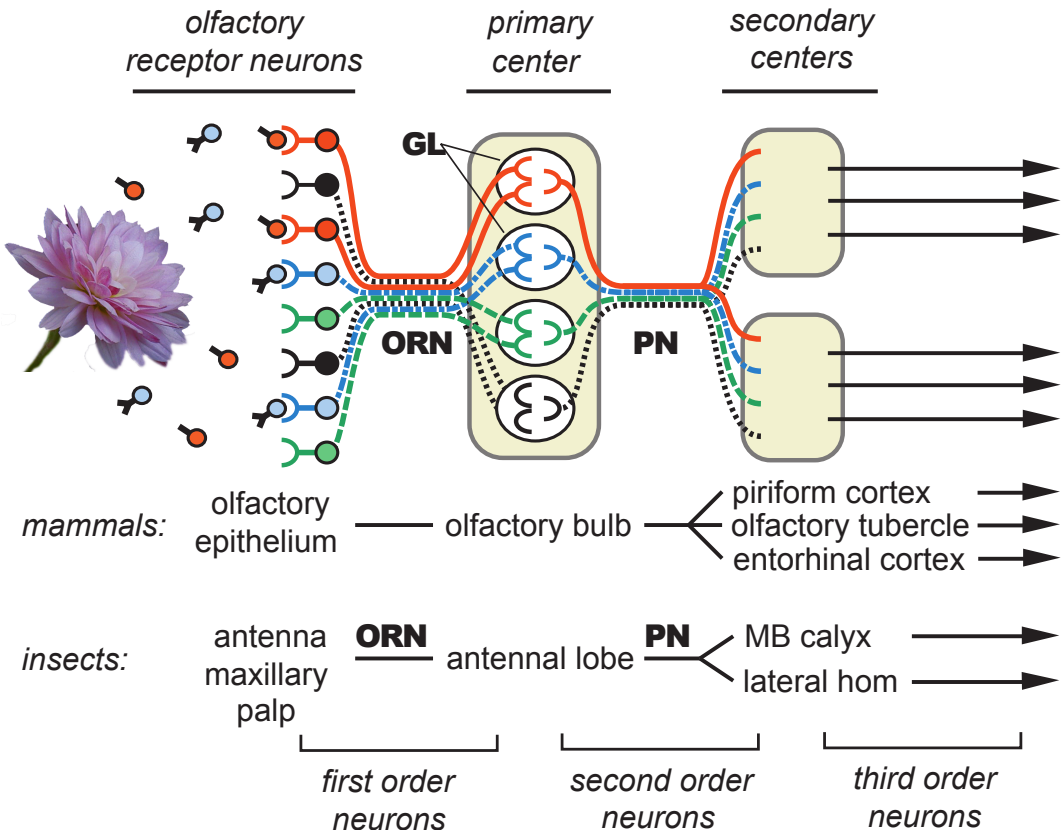

Figure 4.2: Parallels in olfactory processing between mammals and insects. Odorants emitted from a stimulus activate distinct subsets of ORNs, which converge on glomeruli in either the olfactory bulb or the AL. From here information is relayed to higher brain centers, which have functional and neuroanatomical parallels in mammals and insects. Image extracted from (VOSSHALL; STOCKER, 2007)

formed by convergent projections that segregate input from the antenna and maxillary palp according to the type of OR expressed, such that all 1,300 ORNs expressing a given OR target a unique and stereotyped glomerulus (VOSSHALL; STOCKER, 2007), as depicted in Figure 4.3.

4.1.2. Olfactory system in CircuitML

The olfactory model presented here, in CircuitML, is the specification of the olfactory receptors and part of the Antenna Lobe, witch comprises 49 channels - Listing 4.1. It is important to know that this is an early model that is being investigated on the Bionet group (KIM; LAZAR; SLUTSKIY, 2011b) and many aspects of this model is still under development.

In this example, each channel, depicted by lines on different colors in Figure 4.3, encapsulates 3 to 5 projection neurons that receive axons from about 50 ORNs. In addition to PNs, which send olfactory information to higher

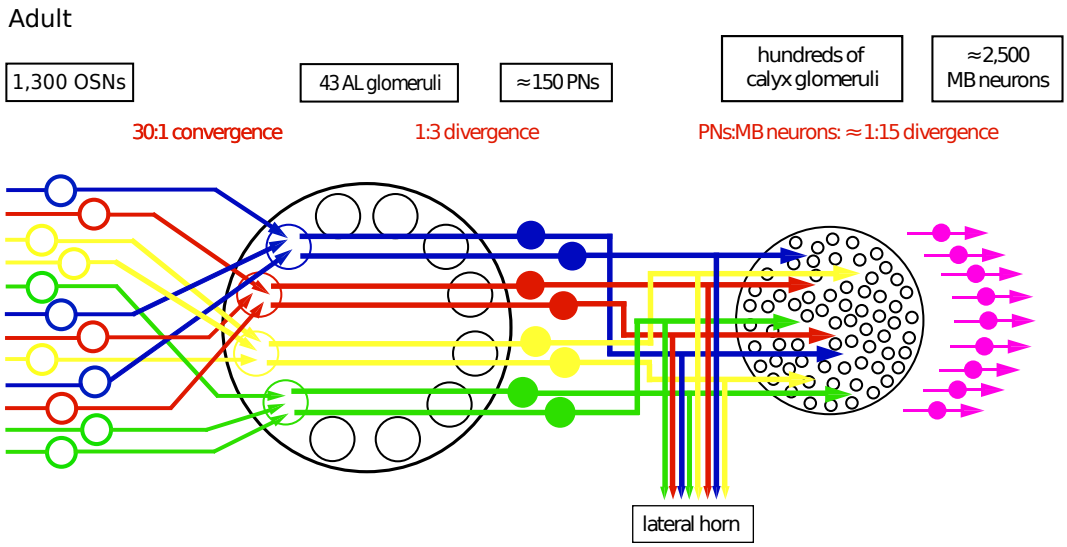

Figure 4.3: Circuit of adult olfactory system. The adult olfactory pathway is characterized by converging and diverging connectivity in the AL (ratios indicated refer to the preceding line). Here, each line that goes from one ORN to one AL is encapsulated into a structure called channel. Image extracted from (VOSSHALL; STOCKER, 2007)

regions of the brain, the antenna lobe contains local neurons (LNs), whose connections are restricted to the lobes; inter-glomeruli connectivity therefore includes synaptic connections between ORNs and PNs, ORNs and LNs, LNs and PNs, and feedback from PNs to LNs. The current specification does not include local neurons, but future specifications will. By encapsulating some of those neurons into channels, it becomes easier to provide mechanisms by which the activation of different sets of ORNs is transformed into an odor perception in the fly brain that produce a given behavioral output. Eventually, with a good AL specification in the future, it would be possible to abstract such module and focus on gathering more information about secondary centers' input and how this data is processed.

The entire model comprises 2,800 neurons, or 70% of the fly's entire antenna lobe. All neurons in the system are modeled using the Leaky Integrate-and-Fire (LIF) model and all synaptic currents elicited by spikes are modeled using alpha functions. Parameters for 24 of the glomerular channels are based upon currently available ORN type data (HALLEM; CARLSON, 2006); all other parameters are configured with artificial data by Bionet.

In Listing 4.1, some of the lines were removed to reduce space. From line 5 to line 28 *antenna left* is specified with 1375 inputs and 147 outputs, where the connectivity pattern between I/O and the neurons inside the LPU is defined from line 7 to line 23 by using the component *interface*. Since all projection neurons comprised by the glomeruli may emit output visible to other LPUs,

their connectivity to outside is defined inside *interface*, where every port will be exposed by the LPU. Local neurons or OSNs that connect glomeruli do not emit any output to other LPUs and their connectivity is defined inside an LPU component, where they are not exposed by the LPU.

Listing 4.1: Demo version of the olfactory system.

```
1 <circuitml id="antenna_demo">
2   <!-- Including external specification -->
3   <Include href="channel_demo.xml" />
4   <!-- First LPU: left antenna -->
5   <lpu id="antenna_left" input="1375" output="147">
6     <!-- LPU I/O -->
7     <interface id="io">
8       <port id="Or2a_ch0" in="0" out="channel/0/OSN/0"/>
9       <port id="Or7a_ch0" in="1" out="channel/0/OSN/1"/>
10      <!-- ... -->
11      <port id="Or98a_ch0" in="23" out="channel/0/OSN/23"/>
12      <!-- ... -->
13      <port id="Or2a_ch49" in="1350" out="channel/49/OSN/0"/>
14      <port id="Or7a_ch49" in="1" out="channel/49/OSN/1"/>
15      <!-- ... -->
16      <port id="Or98a_ch49" in="1374" out="channel/49/OSN/23"/>
17      <!-- output -->
18      <port id="gl_0_0" in="channel/0/GL/0" out="0"/>
19      <port id="gl_0_1" in="channel/0/GL/1" out="1"/>
20      <!-- ... -->
21      <port id="gl_48_2" in="channel/48/GL/2" out="146"/>
22    </interface>
23    <!-- Network specification -->
24    <network id="antenna">
25      <!-- Channels instantiation -->
26      <population id="channels" structure_type="channel"
27       size="49"/>
28    </network>
29  </lpu>
30  <!-- Second LPU: right antenna -->
31  <lpu id="antenna_right" input="1375" output="147">
32    <!-- SAME AS ABOVE -->
33  </lpu>
34 </circuitml>
```

The inner network of neurons is specified using channels, which is defined on Listing 4.2. Synapses and neurons are defined in *synapses.xml* (ln3) and *iafcells.xml* (ln5), respectively, as shown in Chapter 3. Also, some of the lines were hidden to reduce space. From line 8 to line 39, channel's inner circuit is

specified, where all neurons and synapses, once declared on external files, are instantiated by *populations* (for neurons) and *projections* (synapses).

Listing 4.2: Demo version of the channel sub-circuit.

```
<subcircuit id="channel_demo">
  <!-- Synapses types -->
  <Include href="synapses.xml" />
  <!-- Neuron cells -->
  <Include href="iafcells.xml" />
  <iafCell id="pn" V="-0.0716137080262" reset="-0.0725377214812"
   thresh="-0.024302993706" leakConductance="1.01628888072"
   C="0.0691989436227"/>
  <!-- Channel definition -->
  <network id="channel">
    <population id="GL" cell_type="pn" size="3"/>
    <population id="Osn_default" cell_type="osn_default"
     size="1"/>
    <population id="Osn_Or43b" cell_type="osn_Or43b" size="1"/>
    <population id="Osn_Or9a" cell_type="osn_Or9a" size="1"/>
    <population id="Osn_Or47b" cell_type="osn_Or47b" size="1"/>
    <population id="Osn_Or85b" cell_type="osn_Or85b" size="1"/>
    <population id="Osn_Or98a" cell_type="osn_Or98a" size="1"/>
    <population id="Osn_Or82a" cell_type="osn_Or82a" size="1"/>
    <population id="Osn_Or49b" cell_type="osn_Or49b" size="1"/>
    <population id="Osn_Or2a" cell_type="osn_Or2a" size="1"/>
    <population id="Osn_Or43a" cell_type="osn_Or43a" size="1"/>
    <population id="Osn_Or19a" cell_type="osn_Or19a" size="1"/>
    <population id="Osn_Or65a" cell_type="osn_Or65a" size="1"/>
    <population id="Osn_Or10a" cell_type="osn_Or10a" size="1"/>
    <population id="Osn_Or35a" cell_type="osn_Or35a" size="1"/>
    <population id="Osn_Or67c" cell_type="osn_Or67c" size="1"/>
    <population id="Osn_Or7a" cell_type="osn_Or7a" size="1"/>
    <population id="Osn_Or85f" cell_type="osn_Or85f" size="1"/>
    <population id="Osn_Or88a" cell_type="osn_Or88a" size="1"/>
    <population id="Osn_Or23a" cell_type="osn_Or23a" size="1"/>
    <population id="Osn_Or22a" cell_type="osn_Or22a" size="1"/>
    <population id="Osn_Or47a" cell_type="osn_Or47a" size="1"/>
    <population id="Osn_Or59b" cell_type="osn_Or59b" size="1"/>
    <population id="Osn_Or85a" cell_type="osn_Or85a" size="1"/>
    <population id="Osn_Or67a" cell_type="osn_Or67a" size="1"/>
    <!-- Conectivity -->
    <projection id="proj_osn_glomerulus"
     presynapticPopulation="OSN" postsynapticPopulation="GL"
     synapse="syn_osn" />
    <projection id="Osn_default-DA1"
     presynapticPopulation="Osn_default"
     postsynapticPopulation="GL" synapse="osn_default-DA1" />
```

```
44      <!-- ... -->
45      <projection id="Osn_Or67a_24-DM6"
46       presynapticPopulation="Osn_default"
47       postsynapticPopulation="GL" synapse="osn_Or67a_24-DM6" />
48    </network>
49 </subcircuit>
```

Before CircuitML, the way many scientists used to specify such system, would be using graphs, where nodes are neurons and edges are synapses. Using graphs to represent those systems have many advantages on the engineering perspective, but not for the scientist who is specifying the system with thousands, or even, millions of nodes and edges. In order to show the difference between both approaches, Listing 4.3 show the same demo in a graph representation, where cells are defined from line 31 to line 21590 (21559 lines) and synapses are defined from line 21592 to line 71092 (49500 lines).

More important than the huge difference between the number of lines necessary to specify the system, what you can extract looking to both specifications is what matters most. CircuitML was also envisioned to make much clearer the specification. In Listing 4.1 and Listing 4.2 it is easy to understand the big picture of the entire system: the LPUs with input and output, how to circuits are organized and interconnected. Also, using CircuitML, the amount of memory used to save the same specification. The graph version uses 3MB not compressed and 210KB compressed, whereas the CircuitML version uses 790KB not compressed and 171KB compressed.

Listing 4.3: Snippet version of the olfactory system (graph).

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <gexf version="1.2" xmlns="http://www.gexf.net/1.2draft">
3   <graph defaultedgetype="directed">
4     <attributes class="node">
5       <attribute id="0" title="model" type="string">
6         <default>LeakyIAF</default>
7       </attribute>
8       <attribute id="1" title="name" type="string"/>
9       <attribute id="2" title="V" type="float"/>
10      <attribute id="3" title="Vr" type="float"/>
11      <attribute id="4" title="Vt" type="float"/>
12      <attribute id="5" title="R" type="float"/>
13      <attribute id="6" title="C" type="float"/>
14      <attribute id="7" title="spiking" type="boolean"/>
15      <attribute id="8" title="public" type="boolean"/>
16      <attribute id="9" title="extern" type="boolean"/>
17    </attributes>
```

```
    <attributes class="edge">
      <attribute id="0" title="model" type="string">
        <default>AlphaSynapse</default>
      </attribute>
      <attribute id="1" title="name" type="string"/>
      <attribute id="2" title="reverse" type="float"/>
      <attribute id="3" title="ar" type="float"/>
      <attribute id="4" title="ad" type="float"/>
      <attribute id="5" title="gmax" type="float"/>
      <attribute id="6" title="class" type="integer"/>
      <attribute id="7" title="conductance" type="boolean"/>
    </attributes>
    <nodes>
      <node id="0">
        <attvalues>
          <attvalue for="0" value="LeakyIAF"/>
          <attvalue for="1" value="osn_default_0"/>
          <attvalue for="2" value="-0.064215272382"/>
          <attvalue for="3" value="-0.0675489770451"/>
          <attvalue for="4" value="-0.0251355161007"/>
          <attvalue for="5" value="1.02445570216"/>
          <attvalue for="6" value="0.0669810502993"/>
          <attvalue for="7" value="true"/>
          <attvalue for="8" value="true"/>
          <attvalue for="9" value="true"/>
        </attvalues>
      </node>
      <!-- ~21500 lines -->
      <node id="1539">
        <attvalues>
          <attvalue for="0" value="LeakyIAF"/>
          <attvalue for="1" value="VL2p_pn_2"/>
          <attvalue for="2" value="-0.0622911499237"/>
          <attvalue for="3" value="-0.0704026495474"/>
          <attvalue for="4" value="-0.0250364665111"/>
          <attvalue for="5" value="0.974394566901"/>
          <attvalue for="6" value="0.0717119870296"/>
          <attvalue for="7" value="true"/>
          <attvalue for="8" value="true"/>
          <attvalue for="9" value="false"/>
        </attvalues>
      </node>
    </nodes>
    <edges>
      <edge id="0" source="0" target="1375">
        <attvalues>
          <attvalue for="0" value="AlphaSynapse"/>
```

```
65            <attvalue for="1" value="osn_default_0-VM6_pn_0"/>
66            <attvalue for="2" value="0.0"/>
67            <attvalue for="3" value="385.455225163"/>
68            <attvalue for="4" value="101.949722443"/>
69            <attvalue for="5" value="0.00503075744033"/>
70            <attvalue for="6" value="0"/>
71            <attvalue for="7" value="true"/>
72          </attvalues>
73        </edge>
74        <!-- ~50000 lines -->
75        <edge id="4124" source="1374" target="1539">
76          <attvalues>
77            <attvalue for="0" value="AlphaSynapse"/>
78            <attvalue for="1" value="osn_default_24-VL2p_pn_2"/>
79            <attvalue for="2" value="0.0"/>
80            <attvalue for="3" value="417.450869326"/>
81            <attvalue for="4" value="104.596640555"/>
82            <attvalue for="5" value="0.00506945769919"/>
83            <attvalue for="6" value="0"/>
84            <attvalue for="7" value="true"/>
85          </attvalues>
86        </edge>
87      </edges>
88    </graph>
89 </gexf>
```

4.1.3. Olfactory system simulation results

For the simulation of the olfactory system, this study has used, as input, the configuration and odorant waveforms from (HALLEM; CARLSON, 2006). The waveforms database consist of the spontaneous firing rate of 24 ORNs (2a, 7a, 9a, 10a, 22a, 23a, 33b, 35a, 43a, 43b, 47a, 49b, 59b, 65a, 67a, 67c, 82a, 85a, 85b, 85f, 88a and 98a) and the response to the onset of 110 odorant stimuli, which means, for example, if one presents E2-hexenal to neuron 7a, it will produce more than 200 spikes per second. The spontaneous firing rate is used to model the time constant of IAF neurons, and the odorant response is used to define the equivalent input current for the IAF neuron. Using the previous example, to model neuron 7a to respond to odorant E2-hexenal with a firing rate R, which here is greater than 200 sikes per second, the input current is computed such that the IAF exhibits constant firing rate R.

Figure 4.4 shows the input stimulus at the top graph, the spikes produced by the 25 OSNs and 3 PNs associated with glomerulus DA1 in the model. Just

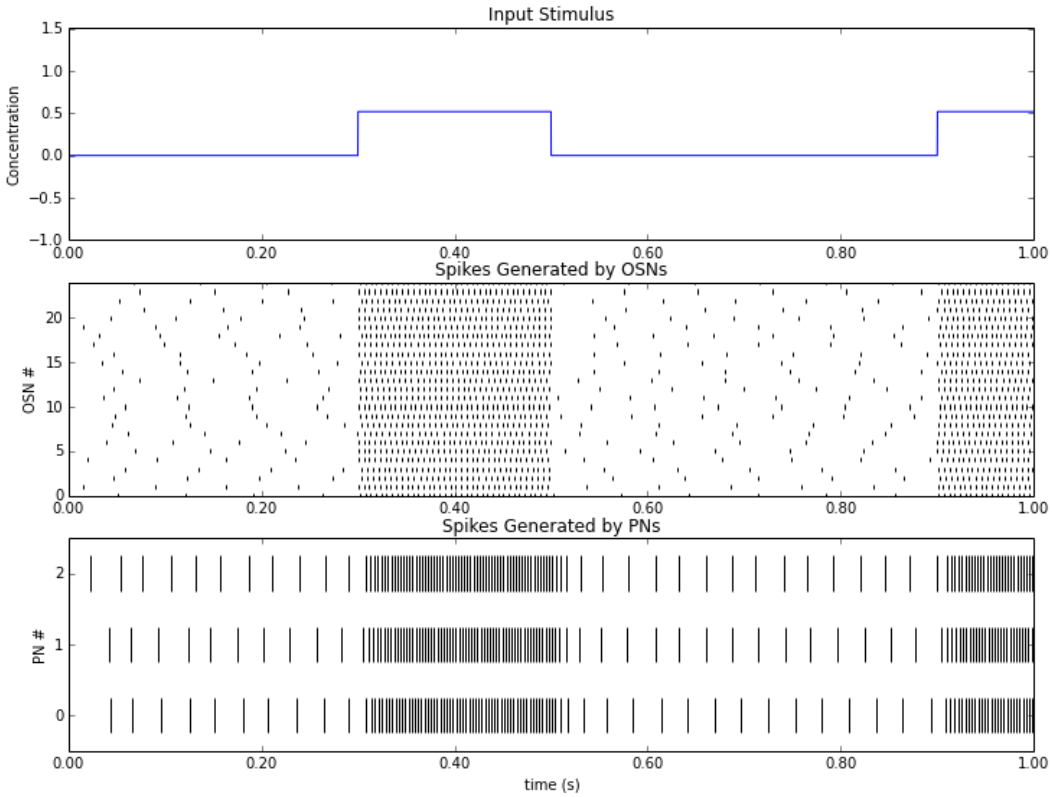

Figure 4.4: Odorant concentration profile and the spikes produced by the 25 OSNs and 3 PNs associated with glomerulus DA1 in the model.

by looking to Figure 4.4, it is possible to notice that during the time the odor was presented, the spikes from OSN increased and, as result, the neurons that received the output from them, also increased their spike rate. This model was simulated using the last version of neurokernel and the circuit developed by Bionet as in November of 2014.

4.2. Visual system of the fly

Drosophila Melanogaster visual system is more complex than its olfactory system and it is organized into modules: retina, lamina, medulla and lobula plate (Figure 4.5) (CHIANG et al., 2011; PAULK; MILLARD; SWINDEREN, 2013), which affects most aspects of visual processing. Figure 4.5 shows the anatomy of the fly's visual system: (a) visual system location on the fly's head and visual information data flow. Visual information enters the eye via the retina, which contains photoreceptors R_{1-8} in individual ommatidium. Each ommatidium contains a structure called cartridge from retina to medulla; (c) R_{1-6} (b) photoreceptors input to the lamina, where neurons L_1, L_2, and L_4 also send output synapses of the motion circuit into the medulla. L_4 receives input from neighboring cartridges, including L_5 neurons, and outputs to multiple

medulla neurons; (d) medulla, like the lamina, is organized into reiterated units. Transmedulla (Tm2) neurons are postsynaptic to lamina neurons and presynaptic to cells in the lobula complex; (d) motion information flows from the lobula to the lobula plate (e) via T5 cells where vertical and horizontal lobula plate tangential cells (LPTCs) respond to specific motion stimuli; (f) further information on patterns such as elevation (top) or orientation (bottom) is discriminated at different levels of the fan-shaped body (green and blue layers) in the central complex, a central brain structure.

4.2.1. Visual organs of adult flies

The first module, considering the direction from input to output, retina comprises 750 individual facets, called ommatidia, make up each compound eye. Each ommatidium is physically separated from its neighbor and contains one each of eight different photoreceptor (R) cells called R_{1-8} (PAULK; MILLARD; SWINDEREN, 2013). Retina is followed by lamina, which maintain the same neurons organization of its predecessor, where R_{1-6} cells target to approximately 750 independent units, called cartridges, and form the first connections with downstream neurons involved in motion processing (PAULK; MILLARD; SWINDEREN, 2013). The next module is called medulla that comprises 750 columns, and the first synapses for color vision (receptor cells R7 and R8) and the second synapses of the motion circuit assemble at distinct vertical positions in each column. The projections of photoreceptors and that of the majority of neurons in the optic lobe are retinotopic, meaning that the spatial relationship between neurons activated by the visual stimulus at the retina is preserved when this information is mapped onto the optic lobe (PAULK; MILLARD; SWINDEREN, 2013). Neurons from medulla project into lobula and lobula plate and a retinotopic map is loosely maintained in these brain regions as well. One of the major challenges the fly visual system faces in terms of organization is in the specificity of the connections within a unit; even though compatible neurons are available close by (just 5-10 microns away), connections with neurons in neighboring columns are prevented in order to preserve the modularity required for efficient visual processing. The mechanisms used to achieve these boundaries within and between columns have only recently begun to be revealed (PAULK; MILLARD; SWINDEREN, 2013).

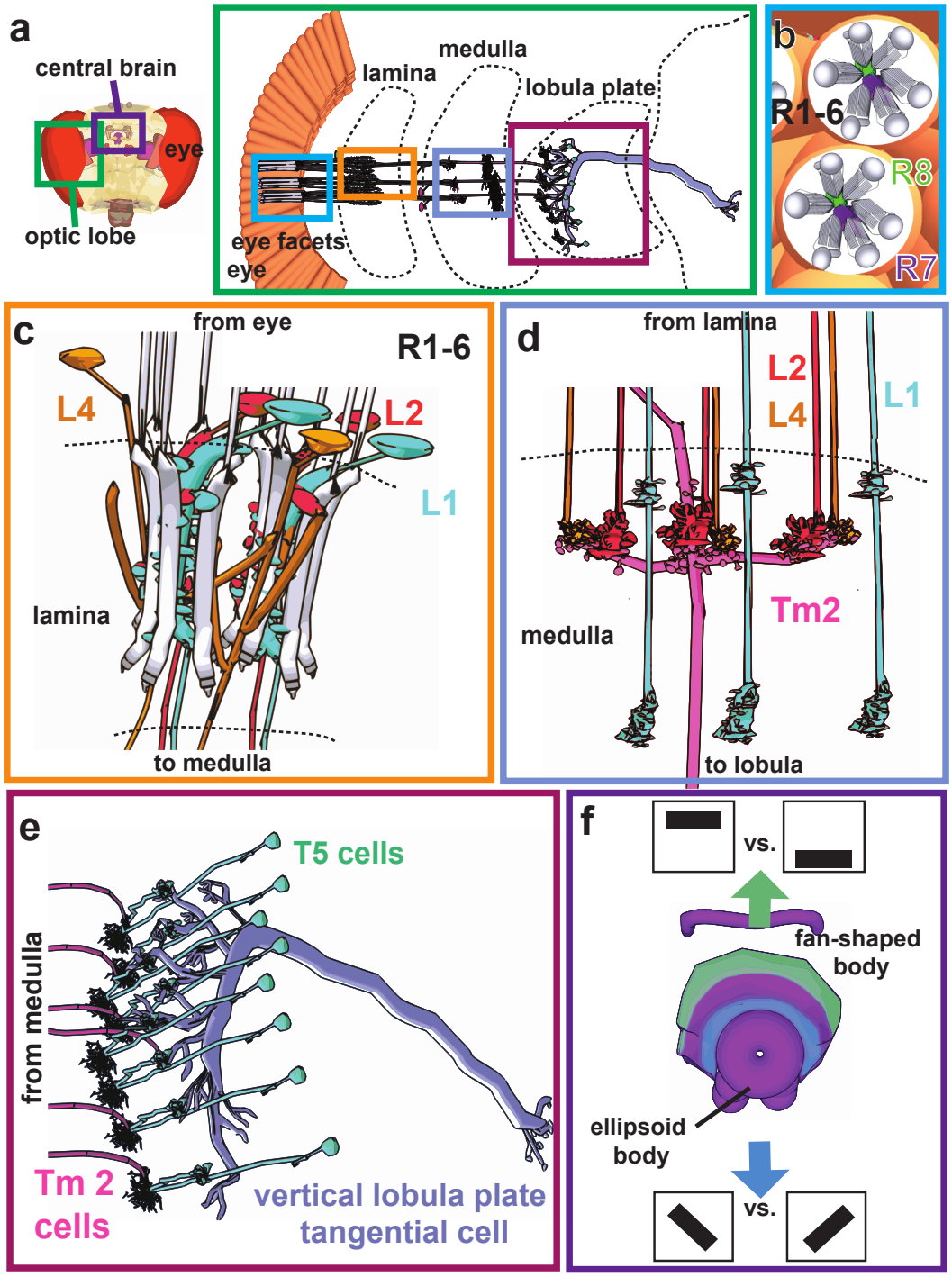

Figure 4.5: The anatomy of the fly visual system. See chapter 4.2 for more information about each part of this figure. Image based on (PAULK; MILLARD; SWINDEREN, 2013).

4.2.2. Abstractions in the visual system

A Lamina circuit model is being proposed by Bionet group at Columbia University (LAZAR; UKANI; ZHOU, 2014), which is based in (RIVERA-ALBA et al., 2011; FISCHBACH; DITTRICH, 1989), and has been previously written in Matlab, it is presented here in CircuitML (Listing 4.5) with some examples comparing both implementations. The Lamina neural circuit consists of approximately 750 cartridges, which are defined as independent sub-circuits (Listing 4.5) interconnected following a repetitive pattern (Listing 4.6). By defining cartridges as atomic and independent sub-circuits has some advantages (LAZAR; UKANI; ZHOU, 2014), such as:

- complexity reduction, since it makes the big lamina's circuit as a set of much smaller sub-circuits;
- simplification of input processing by assigning each cartridge to a pixel of the input image, a univariate signal;
- functionality isolation when considering the cartridge as a fundamental computational unit

Cartridges are defined on a hexagonal grid on a 2D plane, as shown on Figure 4.6. Each cartridge then has a coordinate associated with it. Adjacent cartridges can be easily located with their unique coordinates.

The cartridge's input comes from 6 retina photo receptors, which are represented by neurons R_1 to R_6 that have no sensitivity to color and are located at ommatidium. Each input connection is associated to one ommatidium adjacent following a pattern illustrated in Figure 4.7, where each solid circle is one ommatidium and each dashed circle represents a cartridge. That structure creates an overlaid hexagonal grid, in Figure 4.7 depicted in 2D (see Figure 4.5(b-c) for a 3D representation).

Also, in Figure 4.7, the target cartridge A receives inputs from one of the photoreceptors in each of the six ommatidia that are highlighted with individual photoreceptors R_1 to R_6 (black dots) and R_7 to R_8 (green dots). From the ommatidia perspective, each ommatidium projects its signal to its six neighbor cartridges (LAZAR; UKANI; ZHOU, 2014).

The following tables shows the parameters of cells and synapses in the visual system. As a quick reference, Equation 4.1 show the synapse model used, and Equation 4.2 shows the neuron model used (Both were presented in more details in Chapter 2). All cells defined in the Lamina LPU are listed Table 4.1

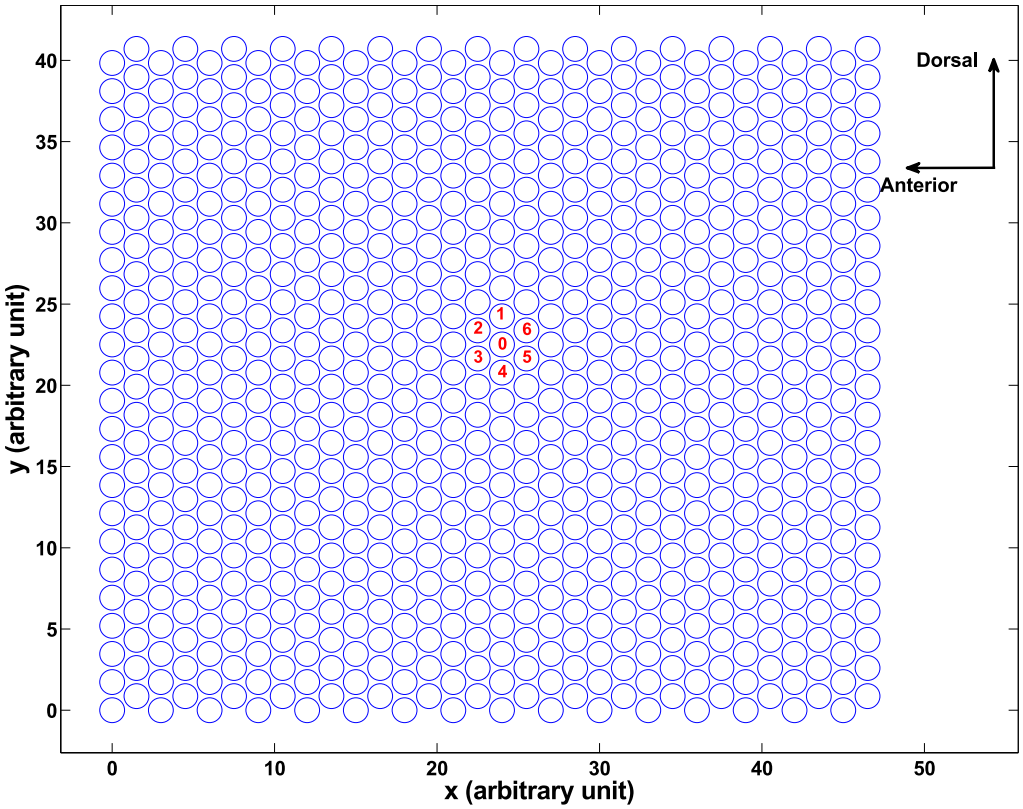

Figure 4.6: Hexagonal grid organization of a cross section of the Lamina cartridges on a 2D plane (right eye). Each circle represents a cartridge. Anterior and Dorsal direction are indicated, and distal direction is into the paper sheet. Figure extracted from (LAZAR; UKANI; ZHOU, 2014).

(LAZAR; UKANI; ZHOU, 2014), which includes the output neurons L1-L5, T1, C2 and C3 neurons and local neuron Am.

$$\begin{aligned} g &= min\left(g_{sat}, k\left(max\left(\left(V_{pre}\left(t - t_{delay}\right) - V_{th}\right)^{n}, 0\right)\right)\right) \\ I_{syn}\left(t\right) &= g\left(V_{post}\left(t\right) - V_{rev}\right) \end{aligned} \tag{4.1}$$

where V_{pre} is the membrane potential of the pre-synaptic neuron and g_{sat}, V_{th}, k and n are, respectively, the saturation of conductance, threshold, scale and power. There is also a t_{delay} added to each synapse regarding the transmission time. In addition, V_{post} and V_{rev} are the membrane potential of the post-synaptic neuron.

$$\frac{dV}{dt} = b - I_{syn} - g_L\left(V - E_L\right) - 0.5 g_{Ca} X - g_K n\left(V - E_K\right)$$
$$X = \left(1 + tanh\left(\frac{V - V_1}{V_2}\right)\right)\left(V - E_{Ca}\right)$$

$$\frac{dn}{dt} = \left(0.5\left(1 + tanh\left(\frac{V - V_3}{V_4}\right)\right) - n\right)\left(\phi \cdot cosh\left(\frac{V - V_3}{2V_4}\right)\right) \quad (4.2)$$

where b is a preset constant bias current, $I_{syn} = I_{syn}(t)$ is the input synaptic current, $E_L = 50mV$, $E_{Ca} = 100mV$ and $E_K = 70mV$ are reverse potential values, and $g_L = 0.5$, $g_{Ca} = 2.0$ and $g_K = 1.1$ are maximum conductance values. The other parameters can be modified in order to adjust the simulation; for this demo, the variables are in Table 4.1.

Table 4.1: Cell parameters in the lamina

Type	V_1	V_2	V_3	V_4	ϕ	$V(0)$	$n(0)$	b
L_1	-0.001	0.015	-0.05	0.001	0.0025	-0.05	0.5	0.02
L_2	-0.001	0.015	-0.05	0.001	0.0025	-0.05	0.5	0.02
L_3	-0.001	0.015	-0.05	0.001	0.0025	-0.05	0.5	0.02
L_4	-0.001	0.015	-0.05	0.001	0.0025	-0.05	0.5	0.02
L_5	-0.001	0.015	-0.05	0.001	0.0025	-0.05	0.5	0.02
T_1	-0.001	0.015	-0.05	0.001	0.0025	-0.05	0.5	0.02
C_2	-0.001	0.015	-0.05	0.001	0.0025	-0.05	0.5	0.02
C_3	-0.001	0.015	-0.05	0.001	0.0025	-0.05	0.5	0.02
Am	-0.001	0.015	0	0.03	0.2	-0.05184	0.0306	0
α_1	-0.001	0.015	-0.05	0.001	0.0025	0	0	0
α_2	-0.001	0.015	-0.05	0.001	0.0025	0	0	0
α_3	-0.001	0.015	-0.05	0.001	0.0025	0	0	0
α_4	-0.001	0.015	-0.05	0.001	0.0025	0	0	0
α_5	-0.001	0.015	-0.05	0.001	0.0025	0	0	0
α_6	-0.001	0.015	-0.05	0.001	0.0025	0	0	0

Cartridge input connectivity is presented in Table 4.2, where *Cartridge* regards which neighbour cartridge receives the connection (Figure 4.6), and

$V_{rev}(V)$, $t_{delay}(ms)$, $V_{th}(V)$, k, n and g_{sat} are the synaptic parameters (LAZAR; UKANI; ZHOU, 2014).

Table 4.2: Cartridge input connectivity (per cartridge)

Pre	Post	V_{rev}	t_{delay}	V_{th}	k	n	g_{sat}	#	mode
R_1	L_1	-0.08	1	-0.05214	0.02	1	0.0008	40	0
R_2	L_1	-0.08	1	-0.05214	0.02	1	0.0008	43	0
R_3	L_1	-0.08	1	-0.05214	0.02	1	0.0008	37	0
R_4	L_1	-0.08	1	-0.05214	0.02	1	0.0008	38	0
R_5	L_1	-0.08	1	-0.05214	0.02	1	0.0008	38	0
R_6	L_1	-0.08	1	-0.05214	0.02	1	0.0008	45	0
L_2	R_1	0	1	-0.0505	1	1	0.02	1	1
α_5	L_3	-0.7	1	-0.5284	0.05	1	0.01	5	0

The output of each cartridge, which is sent to medulla neurons, comes from five L_1 to L_5 neurons, one T_1 neuron and two centrifugal neurons, C_2 and C_3 (As in 2013, centrifugal neurons were not specified yet).

Other than L_n neurons, C_n neurons and T_1 neuron, there are a repetitive element in cartridges called α-profiles and β-profiles as well; they are "light" processes, also known as neurites[1], that are generated by the Am cells. Each cartridge has, in general, 6 α-profiles, each of which may innervate to up to 12 other cartridges. β-profiles receives input from α-profiles and are generated by the T_1 neurons (Figure 4.8). In this circuit model, there are 6 α-profiles, α_1 to α_6, represented as neurons for simplification. They connect Am cells to other cartridge cells and provide feedback onto the photoreceptor input axons (LAZAR; UKANI; ZHOU, 2014). The circuit connectivity of each cartridge in this demo is summarized on Table 4.3, where each table cell show the number of connectivity between two neurons. If one neuron has more than one synaptic contact with another, it is registered as a single synapse, but the weight of the synapse is multiplied by the number of actual synaptic contacts.

The connectivity between cartridges are intermediated by L_4 cells with direct synapses (Table 4.4), but the majority of connections are made by other types of neurons. Although these connections may perform many functions, in this demo, only connections from L_4 cells will be considered. Figure 4.9 shows how L_4 neurons are connected to neighbour cartridges.

[1]A thin tube extending from a neuronal cell body. (WOOD, 1996)

Table 4.3: Cartridge local connectivity matrix

Pre / Post	R_1	R_2	R_3	R_4	R_5	R_6	L_1	L_2	L_3	L_4	L_5	T_1	C_2	C_3	α_1	α_2	α_3	α_4	α_5	α_6
R_1								1												
R_2								1								1				
R_3																				
R_4																	1	1		
R_5										1								1	1	
R_6																				
L_1	40	43	37	38	38	45		3						3						
L_2	46	45	39	41	39	47				2			3	5						
L_3	11	10	4	8	6	12									1	1	1	3	5	1
L_4						2		4												
L_5								1										1	3	
T_1		2	2	2											8	6	7	12	3	13
C_2					1		5										1			
C_3			2				3									1				
α_1	19	16											1	1						
α_2		22	18											3						
α_3			20	16									2	2						
α_4				17	26									2						
α_5					10	14							1	2						
α_6	17					22							1	3						

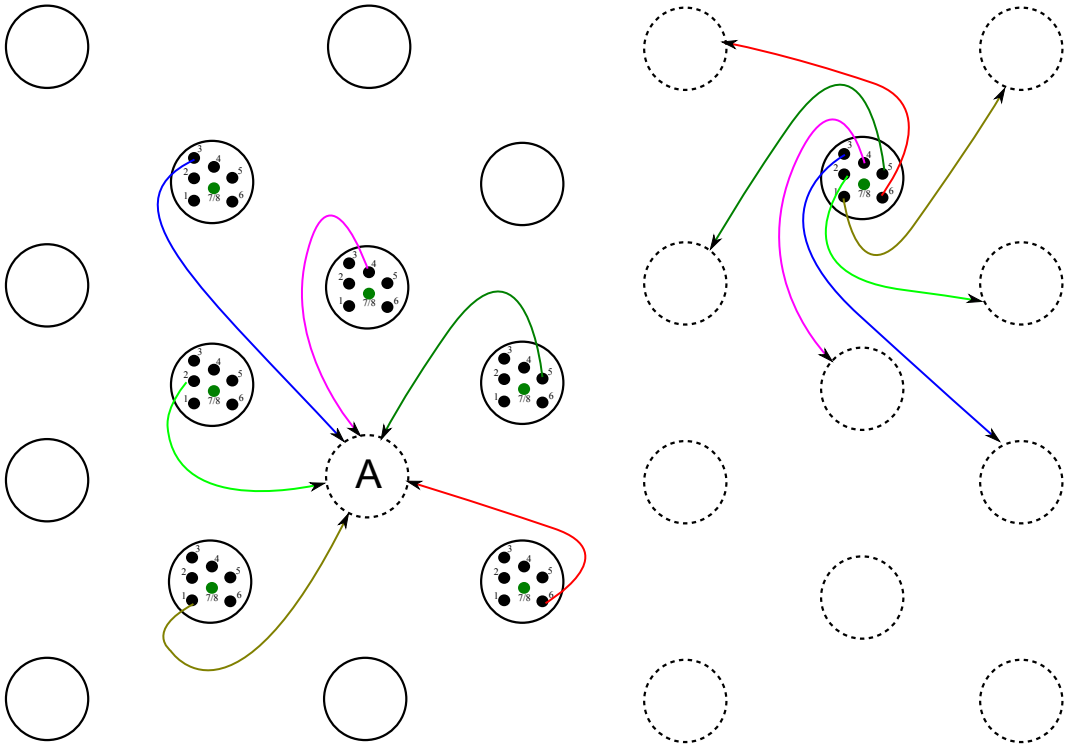

Figure 4.7: Neural superposition rule of the fruit flys eye. A hexagonal grid of ommatidia/cartridges is shown with circles. Dashed circles indicate cartridges and solid circles indicate ommatidia. Note that ommatidia and cartridges are shown on the same plane only for compactness of illustration. Individual photoreceptors R_{1-6} are numbered and their relative position highlighted in some of the ommatidia. Cartridge A receives 6 photoreceptor inputs, each from a different ommatidium. The arrows indicate the 6 photoreceptors that project to the target cartridge A. On the right, 6 photoreceptors from a single ommatidium each projects to a different cartridge. It is clear from the color code that the relative position between the ommatidia is always the same. For example, the location where the R_3 cell (blue) resides and the cartridge where R_3 projects to is locally always the same. Figure extracted from (LAZAR; UKANI; ZHOU, 2014).

Table 4.4: Inter-cartridge connectivity

Pre	Post	Cartridge	V_{rev}	t_{delay}	V_{th}	k	n	g_{sat}	#	mode
L_2	L_4	2	0	1	-0.0505	2	1	0.03	4	0
L_2	L_4	3	0	1	-0.0505	2	1	0.03	2	0
L_4	L_4	4	0	1	-0.0505	2	1	0.05	2	0
L_4	R_3	5	0	1	-0.0505	2	1	0.1	2	1
L_4	L_4	5	0	1	-0.0505	2	1	0.05	1	0
L_4	L_2	6	0	1	-0.0505	0.5	1	0.2	3	0

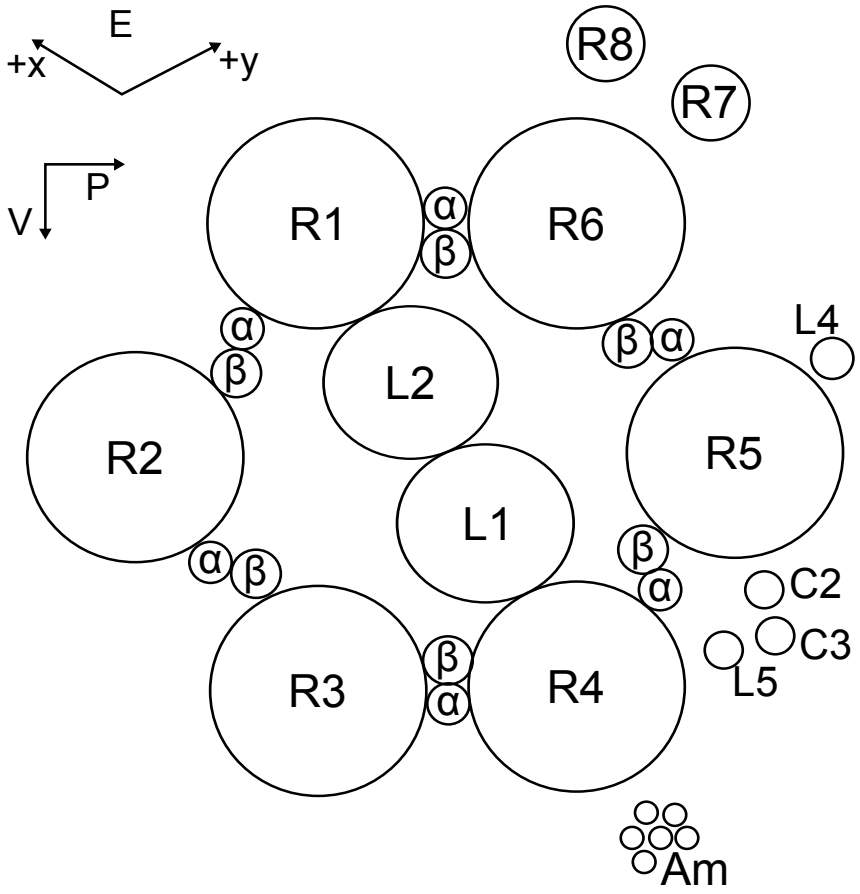

Figure 4.8: Cartridge elements in a cross section of a single cartridge. α-profiles from Amacrine cells and β-profiles from a T_1 neuron can be seen in between each pair of adjacent photoreceptor axons. Copied from (MEINERTZHAGEN; O'NEIL, 1991) Copyright ©1991 Wiley-Liss.

4.2.3. Visual system in CircuitML

The demo code presented here is divided into three main parts: (1) ***cartridge*** abstraction (Listings 4.4), (2) ***lamina*** specification (Listings 4.5) and the connectivity pattern among cartridges (Listings 4.6). In this version of libCircuitML, there is no support to repetitive connectivity patterns between components other than LPUs, but in future releases it will be possible to reuse the connectivity as presented in Listings 4.6.

Listing 4.4: Demo version of cartridge specification.

```
1 <circuitml id="cart_demo">
2   <!-- Synapses types -->
3   <gpSynapse id="R_L_syn" reverse="-0.8" threshold="-0.5214"
4    delay="1" slope=".002" power="1" saturation=".0008"/>
5   <gpSynapse id="R_a_syn" reverse="0" threshold="-0.5214"
6    delay="1" slope=".001" power="1" saturation=".0002"/>
7   <gpSynapse id="R_TC_syn" reverse="-0.7" threshold="-0.5214"
```

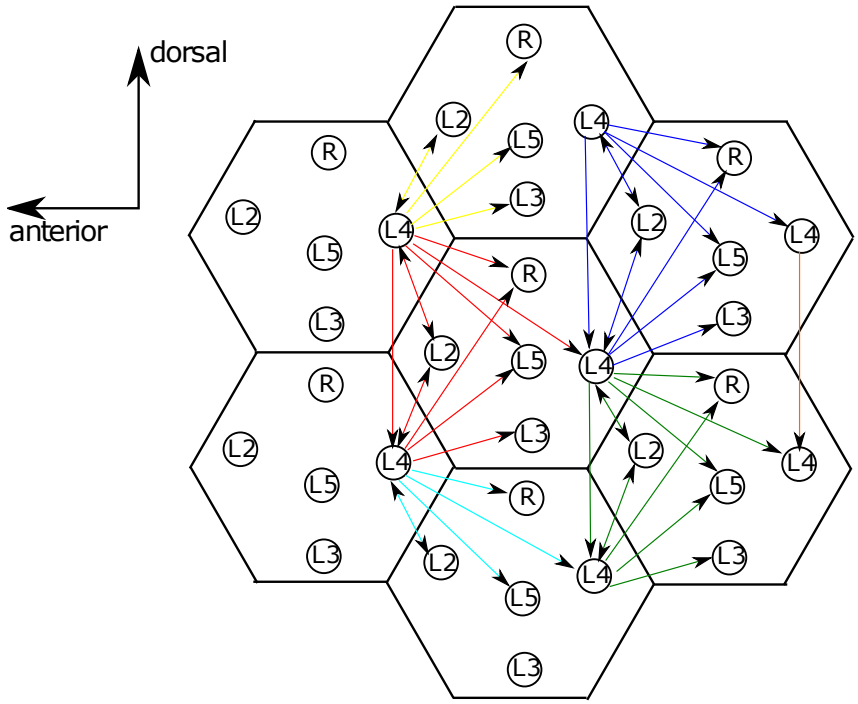

Figure 4.9: Inter-cartridge connectivity between cartridge output neurons, which are mediated by L_4 collaterals from two adjacent cartridges. Figure extracted from (LAZAR; UKANI; ZHOU, 2014).

```
8    delay="1" slope=".001" power="1" saturation=".0002"/>
9   <gpSynapse id="a_L_syn" reverse="-0.7" threshold="-0.5284"
10   delay="1" slope=".05" power="1" saturation=".01"/>
11  <gpSynapse id="L_R_syn" reverse="-0.7" threshold="-0.5284"
12   delay="1" slope=".05" power="1" saturation=".01"/>
13  <!-- For simplification, some parameters of the synapses
14   between L cells were condensed into one type of synapse. In
15   the future, these parameters will be generated by the system.
16   -->
17  <gpSynapse id="L_syn" reverse="-0.7" threshold="-0.505"
18   delay="1" slope=".05" power="1" saturation=".06"/>
19  <gpSynapse id="a_TC_syn" reverse="-0.7" threshold="-0.05"
20   delay="1" slope=".05" power="1" saturation=".06"/>
21  <!-- Neuron cells -->
22  <morrisLecarCell id="R" EL="50" I="65" ECa="100" EK="70"
23   gL="0.5" gCa="2.0" gK="1.1" V1=".3" V2=".15" V3="0" V4=".3"
24   phi="0.025" b="0.02" n0="0.5" V0="0.05"/>
25  <morrisLecarCell id="L" EL="50" I="65" ECa="100" EK="70"
26   gL="0.5" gCa="2.0" gK="1.1" V1="-0.01" V2="0.15" V3="-0.5"
27   V4="0.01" phi="0.0025" b="0.02" n0="0.5" V0="0.05"/>
28  <morrisLecarCell id="T_and_C" EL="50" I="65" ECa="100" EK="70"
29   gL="0.5" gCa="2.0" gK="1.1" V1="-0.01" V2=".15" V3="-0.5"
30   V4=".01" phi=".0025" b="0.02" n0="0.5" V0="0.05"/>
31  <morrisLecarCell id="a" EL="50" I="65" ECa="100" EK="70"
```

```
32    gL="0.5" gCa="2.0" gK="1.1" V1="-0.01" V2=".15" V3="-0.05"
33    V4=".01" phi=".0025" n0="0" V0="0"/>
34   <morrisLecarCell id="Am" EL="50" I="65" ECa="100" EK="70"
35    gL="0.5" gCa="2.0" gK="1.1" V1="-0.01" V2="0.15" V3="0"
36    V4="0.03" phi=".0025" b="0" n0="0.0306" V0="-0.05184"/>
37   <!-- Cartridge definition -->
38   <subcircuit id="cartridge">
39     <!-- Interface definition -->
40     <population id="R_pop" component="R" size="6"/>
41     <population id="a_pop" component="a" size="6"/>
42     <population id="L_pop" component="L" size="5"/>
43     <population id="T_and_C_pop" component="T_and_C" size="3"/>
44     <!-- Local connectivity - based on Nikul's and Yiyin's
45      specification. -->
46     <projection id="R_L" presynapticPopulation="R_pop"
47      postsynapticPopulation="L_pop" synapse="R_L_syn"/>
48       <connection from="0" to="0"/>
49       <connection from="1" to="0"/>
50       <connection from="2" to="0"/>
51       <connection from="3" to="0"/>
52       <connection from="4" to="0"/>
53       <connection from="5" to="0"/>
54       <connection from="0" to="1"/>
55       <connection from="1" to="1"/>
56       <connection from="2" to="1"/>
57       <connection from="3" to="1"/>
58       <connection from="4" to="1"/>
59       <connection from="5" to="1"/>
60       <connection from="0" to="2"/>
61       <connection from="1" to="2"/>
62       <connection from="2" to="2"/>
63       <connection from="3" to="2"/>
64       <connection from="4" to="2"/>
65       <connection from="5" to="2"/>
66     </projeciton>
67     <projection id="R_a" presynapticPopulation="R_pop"
68      postsynapticPopulation="a_pop" synapse="R_a_syn"/>
69       <connection from="0" to="0"/>
70       <connection from="1" to="0"/>
71       <connection from="1" to="1"/>
72       <connection from="2" to="1"/>
73       <connection from="2" to="2"/>
74       <connection from="3" to="2"/>
75       <connection from="3" to="3"/>
76       <connection from="4" to="3"/>
77       <connection from="4" to="4"/>
78       <connection from="5" to="4"/>
```

```
79        <connection from="5" to="5"/>
80        <connection from="0" to="5"/>
81      </projeciton>
82      <projection id="R_T" presynapticPopulation="R_pop"
83       postsynapticPopulation="T_and_C_pop" synapse="R_TC_syn"/>
84        <connection from="1" to="0"/>
85        <connection from="2" to="0"/>
86        <connection from="3" to="0"/>
87      </projeciton>
88      <projection id="a_L" presynapticPopulation="a_pop"
89       postsynapticPopulation="L_pop" synapse="a_L_syn"/>
90        <connection from="5" to="1"/>
91        <connection from="0" to="2"/>
92        <connection from="1" to="2"/>
93        <connection from="2" to="2"/>
94        <connection from="3" to="2"/>
95        <connection from="4" to="2"/>
96        <connection from="5" to="2"/>
97      </projeciton>
98      <projection id="a_T" presynapticPopulation="a_pop"
99       postsynapticPopulation="T_and_C_pop" synapse="a_TC_syn"/>
100       <connection from="0" to="0"/>
101       <connection from="1" to="0"/>
102       <connection from="2" to="0"/>
103       <connection from="3" to="0"/>
104       <connection from="4" to="0"/>
105       <connection from="5" to="0"/>
106     </projeciton>
107     <projection id="L_L" presynapticPopulation="L_pop"
108      postsynapticPopulation="L_pop" synapse="L_syn"/>
109       <connection from="1" to="3"/>
110       <connection from="1" to="4"/>
111       <connection from="0" to="4"/>
112       <connection from="1" to="0"/>
113       <connection from="3" to="1"/>
114       <connection from="3" to="4"/>
115       <connection from="4" to="0"/>
116     </projeciton>
117     <projection id="L_R" presynapticPopulation="L_pop"
118      postsynapticPopulation="R_pop" synapse="L_R_syn"/>
119       <connection from="1" to="0"/>
120       <connection from="1" to="1"/>
121       <connection from="3" to="4"/>
122     </projeciton>
123   </subcircuit>
124 </circuitml>
```

Listing 4.5: Demo version of lamina.

```
<circuitml id="lamina_demo">
  <Include href="cart_demo.xml" />
  <lpu id="lamina">
    <interface>
      <!-- Since there is no specification of which element in
       cartrigdes population, a full-connected fashion is
       set. -->
      <port id="R1" in="0" out="cartrigdes/R/0"/>
      <port id="R2" in="768" out="cartrigdes/R/1"/>
      <port id="R3" in="1536" out="cartrigdes/R/2"/>
      <port id="R4" in="2304" out="cartrigdes/R/3"/>
      <port id="R5" in="3072" out="cartrigdes/R/4"/>
      <port id="R6" in="3840" out="cartrigdes/R/5"/>
    </interface>
      <gpSynapse id="R_L_syn" reverse="-0.8" threshold="-0.5214"
       delay="1" slope=".002" power="1" saturation=".0008"/>
    <morrisLecarCell id="am" C="20" I="65" gL="2" gCa="4"
     gK="8" VL="-50" VCa="100" VK="-70" V1="-0.01" V2=".15"
     V3="0" V4=".3" phi=".2"/>
    <population id="cartrigdes" subcircuit="cart_demo">
      <!-- The space is a hexagonal tiling like {6,3} in
       Schl fli symbol. See
       http://en.wikipedia.org/wiki/Hexagonal_tiling -->
      <layout space="Hexagonal">
        <grid xSize="32" ySize="24" />
      </layout>
    </population>
    <population id="amacrine_cells" component="amacrine">
      <layout>
        <random size="300" />
      </layout>
    </population>
    <projection id="North_west_connectivity"
     source="cartrigdes/0" target="cartrigdes/1" />
      ...
  </lpu>
</circuitml>
```

Listing 4.6: Demo version of the connectivity among cartridges.

```
<circuitml id="lpus">
  <include href="partner_detector.xml" />
  <include href="decode_lpu.xml" />
  <expOneSynapse id="my_syn" erev="20mV" gbase="65nS"
   tauDecay="3ms" />
```

```
6   <connectivity id="pd_to_decoder" lpu1="partner_detector"
7    lpu2="decode_lpu">
8     <connection from="partner_detector/valid_out"
9      to="decode_lpu/valid" synapse="my_syn"/>
10    <connection from="partner_detector/invalid_out"
11     to="decode_lpu/nvalid" synapse="my_syn"/>
12  </connectivity>
13 </circuitml>
```

4.2.4. Visual system simulation results

The neuron and synapse models were implemented as part of a Neurokernel LPU class and libCircuitML process CircuitML files and instantiate those elements using neurokernel own components. The neurokernel LPU class implements GPU-level execution of the LPU models. Consequently, the main point in this results is to compare how this new language can make the design process easier.

To simulate the visual system, we have presented a striped video to the virtual eye, in order to stimulate neurons in the retina (Figure 4.10). The simulation of this video shows an interesting relationship between retina (blue plot) and lamina (red plot), in which it is possible to notice the information propagating from one to another, in a derivative-like form and with a small time difference, which can be modified in a future version that consider the propagation time. Also, from results similar to these, scientists can start to understand functional relations between brain parts.

```
<morrisLecarCell id="L" C="20" I="65" gL="2" gCa="4" gK="8"
  VL="-50" VCa="100" VK="-70" V1="0" V2="15" V3="10" V4="10"
  phi="0.1" />
```

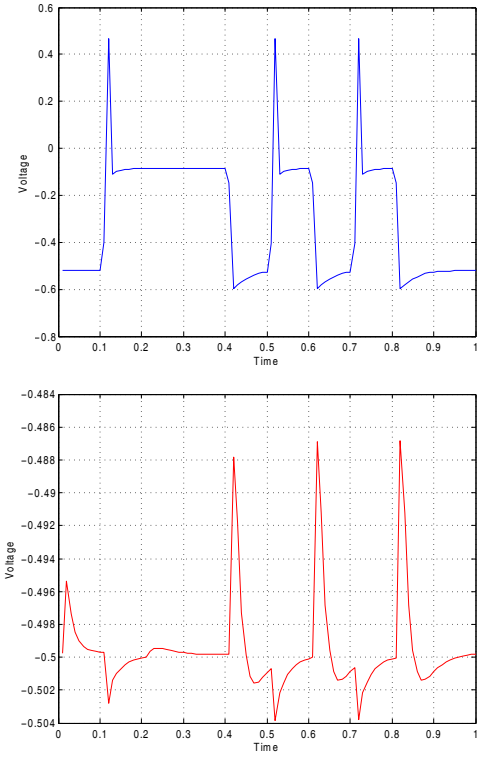

Figure 4.10: Simulation results for the motion detection circuit in the visual system. On the top-left side, the pattern presented to the fly; on the right, one second of output recorded from one random neuron in the lamina (blue) and another random one from the medulla (red).

5
Conclusions

In this thesis, we proposed a toolkit comprising both the specification language CircuitML, and libCircuitML, a Python API that can be imported into a Python script in order to have programmatic functionality over CircuitML. This was motivated by the difficulty in creating models of complex neural networks with their functionality clearly grouped in smaller units. In addition, we specified the fly's olfactory and visual systems in CircuitML to show how the language manages the various parts of those systems. The *Drosophila Melanogaster* was chosen because it represents a good model for studying LPUs.

5.1.
CircuitML

The language of CircuitML extends NeuroML in order to have neuronal systems designed as functional building blocks. Each of those building blocks, also presented as an LPU, may comprise elements ranging from single populations of cells to complex circuitry with many layers of abstractions. Although LPUs can be very complex in the inside, they are endowed with interface ports, which hide such complexity by exposing only necessary parts to be interconnected via neural connectivity patterns, which makes CircuitML a very powerful specification language for neuronal circuitry.

During the development of CircuitML, the language proved to be a powerful ally by encapsulating the many parts of demonstration circuits, each of which could be specified gradually, where each part could be validated without the necessity of having the entire system. As any other object-oriented language, the beginning of the specification takes more time because the modeler have to organize which parts will be encapsulated and which parts will be exposed via interface ports, although the definition of LPUs are easier, since they are already delineated by the biological system into neuropils. However, after having all parts defined, it is much easier to add or remove pieces without affecting other parts of the specification.

5.2. libCircuitML

The companion tool for CircuitML, libCircuitML, was mainly developed to load, validate, parse and save CircuitML files. In addition, libCircuitML was envisioned to provide additional functionality, such as Python object model and other programmatic tools (loops, conditional clauses, etc.) that makes easier to create large models.

During the development of libCircuitML, the hardest part was the processing of the connectivity of the various levels of components and external resources added with the new *Include* element. In order to create one single neural network, grouped by neuron type, from an entire LPU circuitry, the library has to take into account how many cells exists in each group, what type of output that each type generates (continuous or spikes) to create the memory structures, what type of synapse and how long it takes to a signal to reach the other neuron, etc. Also, if there is more than one GPU card available, libCircuitML may, not only send some networks to another GPU cards, but split bigger ones into two or more GPU cards.

As libCircuitML uses DOM (Document Object Model) to transform the XML file in structures both in Python and CUDA, even not being simple it is easier to process the entire LPU connectivity. However, the translation of very big LPUs may take a long time and consume a huge amount of memory. In our examples, the maximum number of neurons was around 15,000 and there was no memory issue nor elapsed time above 2-3 minutes. Actually, there is also a discussion on whether bigger LPUs should not be split into smaller ones.

5.3. The fruit fly virtual brain

As the *Drosophila Melanogaster* represents a good model for studying LPUs, for many reasons, the specification of the Antenna Lobe and of the Lamina LPUs has shown to be an interesting case study for CircuitML.

The first case study presented in this thesis, describing the early olfactory system comprising two Antenna Lobes encapsulating 49 glomerular channels with full intraglomerular connectivity, shows that the specification of neuronal systems as circuits with interconnected chips that have their own functionality, makes the resulting code cleaner and more systematically delineated.

The second study case is the Lamina LPU model, which is based upon available connectome data gathered by researchers at Bionet Group. The LPU model contains 9516 neurons (about 90% of the cells) in the retina and lamina

that were modeled using the Morris-Lecar model with parameters selected to not elicit spiking activity.

Despite the fact that the visual system is much more complex than the olfactory system, and that many peculiarities have not yet been implemented, the CircuitML version of the model makes the specification much cleaner by hiding the complex connectivity between cartridges.

The contributions of this work not only provide new tools for the specification of virtual brains, but it also offers the first two real system specifications in CircuitML. Although far from a complete model, both olfactory and visual systems may be the start point for a complete virtual *Drosophila* and, after that, more complex organisms.

5.4. Future plans

5.4.1. The sensory system of the fruit fly

Albeit the fact that most of the *Drosophila* connectome is still missing, it is already possible to create a draft of all sensory systems of the fly, including the auditory systems, which was not included in this thesis.

The specification of the entire sensory system of the fruit fly with its individual interconnections and its singularities, may prove that CircuitML is able to address the entire system and also will point to possible issues, such as missing connectivity patterns. In addition, since many parts of those systems are still missing, CircuitML may help scientists to reconstruct the fly brain by providing support to functional building blocks that can be tested and validated independently. By sharing their discoveries, research groups will be able to focus on single LPUs, disregarding other parts that are already encapsulated into other LPUs.

The specification of the auditory system seems to be meaningful, since it will complete the entire fruit fly sensory system, and a step forward to a complete virtual fly brain. The auditory system will comprise 250 Johnston's organ neurons (JONs) (TOOTOONIAN et al., 2012) covering the entire frequency range of the sound wave that a fly can hear, and classifying the input into "match"/"not match". This system is used for coupling among flies, being crucial to avoid copulation between flies from different species.

5.4.2. System integration

After implementing the three different sensory systems of the Drosophila fly, in order to move towards a virtual brain, it is imperative to interconnect those systems and validate against the real fly. For example, how both natural and artificial systems would behave if it had two kinds of contradictory inputs: a good smell and a predator approaching on its visual field. For us, this example seems to be naive, but for a computer, it may be not.

5.4.3. Tools for automatic growth of circuits in CircuitML

As mentioned before, one problem for the scientists when specifying a circuit, is the lack of information about cells (positioning and parameters) and connectivity. A system that could autonomously develop itself, i.e. grow from a newborn form, connectivity patterns based on some development rules, statistical inference or even some machine learning mechanism, might serve as temporary solution until the real data become available.

There are some tools (BAUER et al., 2012; KOENE et al., 2009; ZUBLER; DOUGLAS, 2009; ZUBLER et al., 2013; ZUBLER et al., 2011) that already provide many functionality in self-generating networks. Some of them already have NeuroML support, such as CX3D that uses a translator NeuroML to CX3D (ZUBLER; DOUGLAS, 2009; ZUBLER et al., 2013; ZUBLER et al., 2011). However, none of them has a native support neither to NeuroML nor to CircuitML. If libCircuitML could embed such function, it would help scientists to work with less different tools.

A
List of abbreviations

AL	Antenna Lobe
API	Application Programming Interface
CML	CircuitML
CUDA	Compute Unified Device Architecture
DOM	Document Object Model
XML	Extensible Markup Language
GUI	Graphical User Interface
GPU	Graphics Processing Unit
IAF	Integrate-and-fire
JON	Johnston's organ neuron
LN	Local Neuron
LEMS	Low Entropy Model Specification
LPU	Local Processing Unit
MB	Mushroom Body
NML	NeuroML
ORN	Olfactory Receptor Neurons
OSN	Olfactory Sensory Neuron
PN	Projection Neuron
SAX	Simple API for XML
TEM	Time Encoding Machine

Bibliography

ARMSTRONG, J. D. et al. Towards a virtual fly brain. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, The Royal Society, v. 367, n. 1896, p. 2387–2397, 2009. 1, 2.1.3, 2.3, 2.4.5, 3, 4

ASCOLI, G. A.; DONOHUE, D. E.; HALAVI, M. NeuroMorpho. Org: a central resource for neuronal morphologies. *The Journal of Neuroscience*, Soc Neuroscience, v. 27, n. 35, p. 9247–9251, 2007. 2.4.2

BAUER, R. et al. Developmental origin of patchy axonal connectivity in the neocortex: a computational model. *Cerebral Cortex*, Oxford Univ Press, p. bhs327, 2012. 5.4.3

BEEMAN, D. GENESIS Modeling Tutorial. *Brains, Minds, and Media*, v. 1, 2005. 2.4.1, 2.4.3, 3.1.3

BOOCH, G. Object-oriented development. *Software Engineering, IEEE Transactions on*, SE-12, n. 2, p. 211–221, Feb 1986. ISSN 0098-5589. 3.1.2

BRETTE, R.; GOODMAN, D. F. Vectorized algorithms for spiking neural network simulation. *Neural computation*, MIT Press, v. 23, n. 6, p. 1503–1535, 2011. (document), 3.2.2, 3.11, 3.2.2, 3.12, 3.3, 3.3

BRETTE, R. et al. Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of computational neuroscience*, Springer, v. 23, n. 3, p. 349–398, 2007. 2.4.4

BRUYNE, M. de; CLYNE, P. J.; CARLSON, J. R. Odor coding in a model olfactory organ: theDrosophila maxillary palp. *The Journal of neuroscience*, Soc Neuroscience, v. 19, n. 11, p. 4520–4532, 1999. 4.1

BUDICK, S. A.; DICKINSON, M. H. Free-flight responses of drosophila melanogaster to attractive odors. *Journal of experimental biology*, The Company of Biologists Ltd, v. 209, n. 15, p. 3001–3017, 2006. 1, 2.3

CANNON, R. et al. A declarative model specification system allowing NeuroML to be extended with user-defined component types. *BMC Neuroscience*, BioMed Central Ltd, v. 13, n. Suppl 1, p. P42, 2012. 2.4.1, 3.1.3, 3.1.3

CANNON, R. C. et al. Interoperability of neuroscience modeling software: current status and future directions. *Neuroinformatics*, Springer-Verlag, v. 5, n. 2, p. 127–138, 2007. 2.4.4

CANNON, R. C.; O'DONNELL, C.; NOLAN, M. F. Stochastic ion channel gating in dendritic neurons: morphology dependence and probabilistic synaptic activation of dendritic spikes. *PLoS computational biology*, Public Library of Science, v. 6, n. 8, p. e1000886, 2010. 2.4.4

CARDONA, A. et al. TrakEM2 software for neural circuit reconstruction. *PLoS One*, Public Library of Science, v. 7, n. 6, p. e38011, 2012. 2.4.5

CARNEVALE, N. T.; HINES, M. L. *The NEURON book*. [S.l.]: Cambridge University Press, 2006. 2.4.1, 2.4.4

CARON, S. J. et al. Random convergence of olfactory inputs in the Drosophila mushroom body. *Nature*, Nature Publishing Group, 2013. 4

CHEVITARESE, D. S. et al. CircuitML: a Modular Language for Modeling Local Processing Units in the Drosophila Brain. In: *Frontiers Neuroinformatics*. [S.l.]: Frontiers Research Foundation, 2013. p. 80–81. 3

CHEVITARESE, D. S.; SZWARCMAN, D.; VELLASCO, M. Speeding up the training of neural networks with CUDA technology. In: SPRINGER. *Artificial Intelligence and Soft Computing*. [S.l.], 2012. p. 30–38. 2.4.4, 3.3

CHIANG, A.-S. et al. Three-Dimensional Reconstruction of Brain-wide Wiring Networks in Drosophila at Single-Cell Resolution. *Current Biology*, Elsevier, v. 21, n. 1, p. 1–11, 2011. 1, 2.3.1, 2.4.2, 3.3, 4, 4.2

CHIAPPE, M. E. et al. Walking modulates speed sensitivity in¡ i¿ drosophila¡/i¿ motion vision. *Current Biology*, Elsevier, v. 20, n. 16, p. 1470–1475, 2010. 1, 2.3

CHKLOVSKII, D. B.; VITALADEVUNI, S.; SCHEFFER, L. K. Semi-automated reconstruction of neural circuits using electron microscopy. *Current opinion in neurobiology*, Elsevier, v. 20, n. 5, p. 667–675, 2010. 1, 2.3

CUBERT, R. M.; FISHWICK, P. A. MOOSE: an object-oriented multimodeling and simulation application framework. *Simulation*, DTIC Document, v. 6, 1997. 2.4.1

CUNTZ, H. et al. One rule to grow them all: a general theory of neuronal branching and its practical application. *PLoS computational biology*, Public Library of Science, v. 6, n. 8, p. e1000877, 2010. 2.4.4

DAVISON, A. P. et al. PyNN: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, Frontiers Research Foundation, v. 2, 2008. 2.4.3

DAVISON, A. P.; HINES, M. L.; MULLER, E. Trends in programming languages for neuroscience simulations. *Frontiers in neuroscience*, Frontiers Research Foundation, v. 3, n. 3, p. 374, 2009. 3.2.1

DUFFY, J. B. Gal4 system in drosophila: a fly geneticist's swiss army knife. *genesis*, Wiley Online Library, v. 34, n. 1-2, p. 1–15, 2002. 1

EBERHARD, J.; WANNER, A.; WITTUM, G. NeuGen: A tool for the generation of realistic morphology of cortical neurons and neural networks in 3D. *Neurocomputing*, Elsevier, v. 70, n. 1, p. 327–342, 2006. 2.4.5

FISCHBACH, K.-F.; DITTRICH, A. The optic lobe of drosophila melanogaster. i. a golgi analysis of wild-type structure. *Cell and tissue research*, Springer, v. 258, n. 3, p. 441–475, 1989. 4, 4.2.2

GEWALTIG, M.-O.; DIESMANN, M. Nest (neural simulation tool). *Scholarpedia*, v. 2, n. 4, p. 1430, 2007. 2.4.1

GIVON, L. E.; LAZAR, A. A. An open architecture for the massively parallel emulation of the Drosophila brain on multiple GPUs. *BMC Neuroscience*, Springer, v. 13, p. 1–2, 2012. 2.4.4, 3.2

GLEESON, P. et al. NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS computational biology*, Public Library of Science, v. 6, n. 6, p. e1000815, 2010. 1, 2.4.1, 3, 3.1.3

GLEESON, P.; STEUBER, V.; SILVER, R. neuroConstruct: a tool for modeling networks of neurons in 3D space. *Neuron*, Elsevier, v. 54, n. 2, p. 219, 2007. 2.4.3, 3.3

GLEESON, P. et al. NeuroML. In: Le Novère, N. (Ed.). *Computational Systems Neurobiology*. Springer Netherlands, 2012. p. 489–517. ISBN 978-94-007-3857-7. Disponível em: ⟨http://dx.doi.org/10.1007/978-94-007-3858-4_16⟩. 1, 3.2

GOLDMAN, A. L. et al. Coexpression of two functional odor receptors in one neuron. *Neuron*, Elsevier, v. 45, n. 5, p. 661–666, 2005. 4.1

HALLEM, E. A.; CARLSON, J. R. Coding of odors by a receptor repertoire. *Cell*, Elsevier, v. 125, n. 1, p. 143–160, 2006. 4.1.2, 4.1.3

HINES, M.; CARNEVALE, N. Translating network models to parallel hardware in NEURON. *Journal of neuroscience methods*, Elsevier, v. 169, n. 2, p. 425–455, 2008. 2.4.4

HINES, M. L.; CARNEVALE, N. T. The NEURON simulation environment. *Neural computation*, MIT Press, v. 9, n. 6, p. 1179–1209, 1997. 2.4.3, 2.4.4

HODGKIN, A. L.; HUXLEY, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, Blackwell Publishing, v. 117, n. 4, p. 500, 1952. 3.2

IZHIKEVICH, E. M. Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, IEEE, v. 14, n. 6, p. 1569–1572, 2003. 2.4.4

JORDAN, N. M.; PERRY, K. B.; NARALA, N. Design and Implementation of an NCS-NeuroML Translator. -, 2012. 2.4.5

KANDEL, E. R. et al. *Principles of neural science*. [S.l.]: McGraw-Hill New York, 2000. v. 4. 1, 1.2, 2.1, 2.1.1, 2.1.2, 2.1.3

KATSOV, A. Y.; CLANDININ, T. R. Motion processing streams in¡ i¿ drosophila¡/i¿ are behaviorally specialized. *Neuron*, Elsevier, v. 59, n. 2, p. 322–335, 2008. 1, 2.3

KIM, A. J.; LAZAR, A. A.; SLUTSKIY, Y. System identification of dm4 glomerulus in the drosophila antennal lobe using stationary and non-stationary odor stimuli. *BMC Neuroscience*, BioMed Central Ltd, v. 11, n. Suppl 1, p. P174, 2010. 1, 2.3

KIM, A. J.; LAZAR, A. A.; SLUTSKIY, Y. Drosophila projection neurons encode the acceleration of time-varying odor waveforms. In: *Computational and Systems Neuroscience Meeting*. [S.l.: s.n.], 2011. p. 2. 1, 2.3

KIM, A. J.; LAZAR, A. A.; SLUTSKIY, Y. B. System identification of drosophila olfactory sensory neurons. *Journal of computational neuroscience*, Springer, v. 30, n. 1, p. 143–161, 2011. 1, 2.3, 4.1.2

KIM, A. J.; LAZAR, A. A.; YEVGENIY, B. S. 2d encoding of concentration and concentration gradient in drosophila orns. In: *Computational and Systems Neuroscience Meeting, Salt Lake City, Utah*. [S.l.: s.n.], 2010. 1, 2.3

KLÖCKNER, A. et al. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. *Parallel Computing*, Elsevier, v. 38, n. 3, p. 157–174, 2012. 3.2

KOENE, R. A. et al. NETMORPH: a framework for the stochastic generation of large scale neuronal networks with realistic neuron morphologies. *Neuroinformatics*, Springer, v. 7, n. 3, p. 195–210, 2009. 5.4.3

LAZAR, A. A.; PNEVMATIKAKIS, E. A.; ZHOU, Y. Encoding natural scenes with neural circuits with random thresholds. *Vision research*, Elsevier, v. 50, n. 22, p. 2200–2212, 2010. 3.1.3

LAZAR, A. A.; UKANI, N. H.; ZHOU, Y. The cartridge: A canonical neural circuit abstraction of the lamina neuropil construction and composition rules. 2014. (document), 4.2.2, 4.2.2, 4.6, 4.2.2, 4.2.2, 4.7, 4.9

LICHTMAN, J. W.; LIVET, J.; SANES, J. R. A technicolour approach to the connectome. *Nature Reviews Neuroscience*, Nature Publishing Group, v. 9, n. 6, p. 417–422, 2008. 2.3

MACIONIS, J.; CLARKE, J. N.; GERBER, L. M. *Sociology: Canadian Edition*. [S.l.]: Prentice Hall Canada Inc.: Scarborough, Ontario, 1994. 2.1.2

MAIMON, G.; STRAW, A. D.; DICKINSON, M. H. A simple vision-based algorithm for decision making in flying¡ i¿ drosophila¡/i¿. *Current Biology*, Elsevier, v. 18, n. 6, p. 464–470, 2008. 1, 2.3

MAISAK, M. S. et al. A directional tuning map of drosophila elementary motion detectors. *Nature*, Nature Publishing Group, v. 500, n. 7461, p. 212–216, 2013. 1

MARKRAM, H. The blue brain project. *Nature Reviews Neuroscience*, Nature Publishing Group, v. 7, n. 2, p. 153–160, 2006. 2.4.2

MASUDA-NAKAGAWA, L. M.; TANAKA, N. K.; O'KANE, C. J. Stereotypic and random patterns of connectivity in the larval mushroom body calyx of Drosophila. *Proceedings of the National Academy of Sciences of the United States of America*, National Acad Sciences, v. 102, n. 52, p. 19027–19032, 2005. 3.2.2

MEINERTZHAGEN, I.; O'NEIL, S. Synaptic organization of columnar elements in the lamina of the wild type in drosophila melanogaster. *Journal of comparative neurology*, Wiley Online Library, v. 305, n. 2, p. 232–263, 1991. (document), 4.8

MORRIS, C.; LECAR, H. Voltage oscillations in the barnacle giant muscle fiber. *Biophysical journal*, Elsevier, v. 35, n. 1, p. 193–213, 1981. 2.2.2, 3.2

OLSEN, S. R.; WILSON, R. I. Cracking neural circuits in a tiny brain: new approaches for understanding the neural circuitry of¡ i¿ Drosophila. *Trends in neurosciences*, Elsevier, v. 31, n. 10, p. 512–520, 2008. 1, 2.3, 2.4.4

PAULK, A.; MILLARD, S. S.; SWINDEREN, B. van. Vision in Drosophila: Seeing the World Through a Model's Eyes. *Annual review of entomology*, Annual Reviews, v. 58, p. 313–332, 2013. (document), 4.2, 4.2.1, 4.5

RISTER, J. et al. Dissection of the peripheral motion channel in the visual system of¡ i¿ drosophila melanogaster¡/i¿. *Neuron*, Elsevier, v. 56, n. 1, p. 155–170, 2007. 1

RIVERA-ALBA, M. et al. Wiring economy and volume exclusion determine neuronal placement in the¡ i¿ drosophila¡/i¿ brain. *Current Biology*, Elsevier, v. 21, n. 23, p. 2000–2005, 2011. 4.2.2

RODRIGUEZ, A. et al. Automated three-dimensional detection and shape classification of dendritic spines from fluorescence microscopy images. *PLoS One*, Public Library of Science, v. 3, n. 4, p. e1997, 2008. 2.4.5

SCHWARTZ, E. L. *Computational neuroscience*. [S.l.]: The MIT Press, 1993. 2.1.4

SILVA, C. P. da et al. Exploring data streaming to improve 3d fft implementation on multiple gpus. In: IEEE. *Computer Architecture and High Performance Computing Workshops (SBAC-PADW), 2010 22nd International Symposium on*. [S.l.], 2010. p. 13–18. 3.3

SONG, Z. et al. Stochastic, adaptive sampling of information by microvilli in fly photoreceptors. *Current Biology*, Elsevier, v. 22, n. 15, p. 1371–1380, 2012. 1

SPORNS, O.; GIULIO, T.; ROLF, K. The Human Connectome: A Structural Description of the Human Brain. *PLoS Comput Biol*, Public Library of Science, v. 1, n. 4, p. e42, 09 2005. Disponível em: ⟨http://dx.plos.org/10.1371%2Fjournal.pcbi.0010042⟩. 1

SPORNS, O.; TONONI, G.; KÖTTER, R. The human connectome: a structural description of the human brain. *PLoS computational biology*, Public Library of Science, v. 1, n. 4, p. e42, 2005. 2.3

STURTEVANT, A. Experiments on sex recognition and the problem of sexual selection in Drosoophilia. *Journal of Animal Behavior*, Henry Holt and Company, Inc., v. 5, n. 5, p. 351, 1915. 1

TAKEMURA, S.-y. et al. A visual motion detection circuit suggested by drosophila connectomics. *Nature*, Nature Publishing Group, v. 500, n. 7461, p. 175–181, 2013. 1, 2.3

THIBEAULT, C.; HOANG, R.; Harris Jr, F. *A Novel Multi-GPU Neural Simulator*. [S.l.]: BICoB, 2011. 2.4.4

TOOTOONIAN, S. et al. Neural representations of courtship song in the drosophila brain. *The Journal of Neuroscience*, Soc Neuroscience, v. 32, n. 3, p. 787–798, 2012. 5.4.1

VELLA, M. et al. libneuroml and pylems: using python to combine procedural and declarative modeling approaches in computational neuroscience. *Frontiers in Neuroinformatics*, Frontiers Media SA, v. 8, 2014. 2.4.1

VOSSHALL, L. B.; STOCKER, R. F. Molecular architecture of smell and taste in Drosophila. *Annu. Rev. Neurosci.*, Annual Reviews, v. 30, p. 505–533, 2007. (document), 3.2.2, 4, 4.1, 4.1, 4.1.1, 4.1.1, 4.2, 4.3

WARDILL, T. J. et al. Multiple spectral inputs improve motion discrimination in the drosophila visual system. *Science*, American Association for the Advancement of Science, v. 336, n. 6083, p. 925–931, 2012. 1

WEARNE, S. et al. New techniques for imaging, digitization and analysis of three-dimensional neural morphology on multiple scales. *Neuroscience*, Oxford; New York: Pergamon Press, 1976-, v. 136, n. 3, p. 661–680, 2005. 2.4.5

WERNICKE, C. The symptom complex of aphasia. In: SPRINGER. *Proceedings of the Boston Colloquium for the Philosophy of Science 1966/1968*. [S.l.], 1969. p. 34–97. 2.1.1

WILSON, R. I. Understanding the functional consequences of synaptic specialization: insight from the¡ i¿ drosophila¡/i¿ antennal lobe. *Current opinion in neurobiology*, Elsevier, v. 21, n. 2, p. 254–260, 2011. 1, 2.3

WOOD, I. K. Neuroscience: Exploring the brain. *Journal of Child and Family Studies*, Springer, v. 5, n. 3, p. 377–379, 1996. 1

ZHU, Y. et al. Peripheral visual circuits functionally segregate motion and phototaxis behaviors in the fly. *Current Biology*, Elsevier, v. 19, n. 7, p. 613–619, 2009. 4

ZUBLER, F.; DOUGLAS, R. A framework for modeling the growth and development of neurons and networks. *Frontiers in computational neuroscience*, Frontiers Research Foundation, v. 3, 2009. 2.4.3, 2.4.3, 5.4.3

ZUBLER, F. et al. An instruction language for self-construction in the context of neural networks. *Frontiers in computational neuroscience*, Frontiers Media SA, v. 5, 2011. 5.4.3

ZUBLER, F. et al. Simulating cortical development as a self constructing process: a novel multi-scale approach combining molecular and physical aspects. *PLoS computational biology*, Public Library of Science, v. 9, n. 8, p. e1003173, 2013. 5.4.3