Published in final edited form as:
Philos Trans R Soc Lond B Biol Sci. ; 373(1758): . doi:10.1098/rstb.2017.0380.

Geppetto - A reusable modular open platform for exploring neuroscience data and models

Matteo Cantarelli[1,2,3], **Boris Marin**[3,4], **Adrian Quintana**[2,3,5], **Matt Earnshaw**[3], **Robert Court**[6], **Padraig Gleeson**[3], **Salvador Dura-Bernal**[7], **R. Angus Silver**[3], and **Giovanni Idili**[1,2]

[1]OpenWorm Foundation, US

[2]MetaCell Limited, UK

[3]Department of Neuroscience, Physiology and Pharmacology, University College London, UK

[4]Departamento de Física, Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto, Universidade de São Paulo, Brazil

[5]EyeSeeTea Limited, UK

[6]Institute for Adaptive and Neural Computation, School of Informatics, University of Edinburgh, UK

[7]Department of Physiology and Pharmacology, SUNY Downstate, Brooklyn, NY, USA

Abstract

Geppetto is an open-source platform that provides generic middleware infrastructure for building both online and desktop tools for visualising neuroscience models and data and managing simulations. Geppetto underpins a number of neuroscience applications including Open Source Brain, Virtual Fly Brain, NEURON-UI and NetPyNE-UI. Open Source Brain is used by researchers to create and visualise computational neuroscience models described in NeuroML and simulate them through the browser. Virtual Fly Brain is the reference hub for *Drosophila melanogaster* neural anatomy and imaging data including neuropil, segmented neurons, microscopy stacks and gene expression pattern data. Geppetto is also being used to build a new user interface for NEURON, a widely used neuronal simulation environment, and for NetPyNE, a Python package for network modelling using NEURON. Geppetto defines domain agnostic abstractions used by all these applications to represent their models and data and offers a set of

*Correspondence should be addressed to M.C. (matteo@openworm.org).

Contributions

MC and GI conceived, architected and laid the foundations of Geppetto. MC oversaw the development of Geppetto and MC, GI, AQ, BM further contributed to its architecture. MC, GI, AQ, BM, ME, RC, PG contributed to the Geppetto development. PG and RAS conceived Open Source Brain. MC, AQ, BM, ME, PG, GI, RAS contributed to the design and development of Open Source Brain. RC, MC, GI contributed to the design and development of Virtual Fly Brain. SD-B conceived NetPyNE and MC, AQ, SD-B contributed to the design and development of NetPyNE-UI. AQ, MC, SD-B contributed to the design and development of NEURON-UI. MC, BM, GI, RAS, PG, ME, SD-B, AQ, RC wrote the manuscript.

Competing Interests

MetaCell Ltd. was contracted by UCL to develop some of the features of Open Source Brain. MetaCell Ltd. was contracted by EMBL-EBI to develop some of the features of Virtual Fly Brain. MetaCell LLC was contracted by State University of New York to develop NEURON-UI and NetPyNE-UI. MC, GI declare financial interest in MetaCell Ltd, LLC. The other authors declare no competing financial interests.

Europe PMC Funders Author Manuscripts

modules and components to integrate, visualise and control simulations in a highly accessible way. The platform comprises a backend which can connect to external data sources, model repositories and simulators together with a highly customisable frontend.

Introduction

Investigations of fundamental questions in neuroscience, such as the mechanistic basis of behaviour and cognition, generate large volumes of experimental data, as well as complex computational models spanning different levels of biological detail. These push the neuroscience applications available to researchers to their limits. Visualising and managing the heterogeneity of neuroscience data and models in a way that is accessible and usable for both experimentalists and modellers is crucial for driving the field forward. For example, it has been challenging to visualize the data and models required to link the dynamics of the nervous system of *C. elegans* to its behaviour [1], or to understand how the sleep regulatory circuit in *Drosophila melanogaster* is affected by the surrounding environment [2].

In neuroscience, visualisation and simulation tools exist for many of the levels of detail involved [3–7], but it is often far from trivial to use them in concert [8]. One popular approach to solving this issue involves using general purpose programming languages such as Python [9–11]. This approach enables the rapid development of toolchains to solve a specific visualisation and integration problem, gluing together multiple libraries and tools [12]. The problem with this approach is that these toolchains are usually developed for a specific use case, e.g. processing data from a specific source. Over time, as the application is modified to solve different problems (e.g. deal with a new model or with a new type of visualisation), the specificity becomes an obstacle and the codebase becomes a series of ad hoc extensions that are difficult to maintain [13]. An even greater problem comes from the fact that these tools, and even more so their combination, are rather inaccessible to many researchers. Such technological barriers have had a remarkable effect in the neuroscience field as a whole, resulting in modellers and experimentalists working as two different communities separated by a technological divide. This has resulted in computational models that are poorly validated and has left model-generated hypotheses unexplored.

Data and models come in many different types, which are subject to change as the field evolves. Handling such heterogeneities constitutes a significant challenge for neuroscience applications given that not all of the formats that will be required to answer novel scientific questions will be known at design time. Standard neuroscience formats that have emerged to date include NeuroML [14,15] for computational neuroscience and Neurodata Without Borders [16] for experimental data. Dealing with an extensible set of formats in a more generic yet customisable way requires decoupling the software infrastructure from these domain specific representations. Designing such system is not trivial considering that both experimental and computational data and models each come with their own set of challenges. The sheer size of experimental datasets, particularly those arising from connectomics and imaging, require specific visualisation capabilities and optimisations when handling them. Computational models need to be instantiated within an application to let users interact with their state variables and parameters. Different numerical solvers may

Europe PMC Funders Author Manuscripts

be required for these models to be simulated, but the user will not necessarily want to be exposed to the complexity of the software solution and low level libraries involved [17]. In addition, as the biological detail and scale of simulations increase, transparent access to high-performance computing infrastructures [18] will be required. Data and models are also likely to be stored in repositories and databases using disparate technologies, which poses yet another challenge for applications.

To address the challenges posed by heterogeneous data and models, as well as bridging the divide between users with different fields of expertise, we have developed *Geppetto*, an open source, modular middleware platform that can be used to build different neuroscience applications. In order to process diverse types of data and models in a reusable way, the software infrastructure is decoupled from domain data and model specification. This decoupling is achieved through the *Geppetto Model Abstraction*, designed to represent the underlying experimental and computational data and models in a standard way, via reusable modules. Geppetto is also optimized for coping with large amounts of data, through automatic compression and loading on demand, and is able to run simulations on remote supercomputers. To improve accessibility, Geppetto facilitates building novel interfaces by hiding the underlying technologies and by providing prebuilt user friendly User Interface (UI) components. By abstracting and integrating experimental data, computational models and simulators it is hoped that Geppetto will enable the building of neuroscience applications that can bring together theorists, modelers and experimentalists to formulate and answer increasingly challenging scientific questions related to brain function.

Methods

Geppetto is a modular, extensible open source platform based on a client-server architecture (figure 1) that provides a framework for building neuroscience applications for visualization of data, models and for controlling simulations. The Geppetto Backend architecture defines a set of abstract services for which specific implementations can be provided for different domains. The Geppetto Frontend provides visualization capabilities that encompass a wide range of what is typically needed for neuroscience data visualization, be it experimental data or data resulting from simulations. The Geppetto Frontend is based on a typical modern web stack based on JavaScript and React [19], making use of npm [20] to manage dependencies and webpack [21] to package the code into a browser-ready application.

The *Geppetto Model Abstraction* (figure 1, orange boxes) enables the decoupling of domain specific modelling formats from the visualization components, by providing a meta-model that can be used to represent them in a declarative way. To this end, it defines a type system based on core concepts from Object Oriented Programming: *Variables*, *Types* and *Values*. By supporting *Type* inheritance (any Geppetto *Type* can extend another) and composition (Geppetto's *CompositeType* can contain *Variables* of other *Types*) the *Geppetto Model Abstraction* makes it possible to represent hierarchical structures of data and models. Geppetto uses the Eclipse Modelling Framework (EMF) [22] to specify its models' abstractions. The EMF schema is then used to programmatically generate an API for the *Geppetto Model Abstraction* for each one of the supported (user domain) languages [23,24]. Developers can build their own custom *Types* using this *API,* and use them in combination

with the ones provided in the *Geppetto Common Library* (e.g. *State Variable*, *Parameter*, etc.). Any model created using the *Geppetto Model Abstraction* takes the name of a *Geppetto Model.* Once a domain specific model is described in terms of the *Geppetto Model Abstraction* (e.g., by defining a custom *Type*) the entire platform becomes capable of treating its constituent elements appropriately. It is important to note that in Geppetto, *Types* are defined using a domain agnostic meta-model: while an application could for example create a *Library* of *Types* that represent computational models, another application might build one whose *Types* represent sets of microscopy images. Inside a *Geppetto Model,* developers can also specify the *Data Source* services used to fetch data from remote repositories, along with the *Queries* available to interrogate them. The *Geppetto Model Abstraction* also defines *ImportTypes* which can hold references to data and models existing on the backend that haven't yet been loaded. Sending *ImportTypes* to the client, that will be fully loaded upon a request triggered by the user's actions, is what enables Geppetto to load data on demand (i.e. lazy loading).

The entry point for a Geppetto application is the *Geppetto Project*. Each *Geppetto Project* holds a reference to a single *Geppetto Model* and in addition stores the current state of the application (e.g., which components are open along with their content and position). Every Geppetto application can make use of one or multiple *Geppetto Projects.* For example, in Open Source Brain (described below in the Results section), each computational neuroscience model (e.g., cell, network) loaded in from a NeuroML file is mapped to a *Geppetto Model* and contained within a *Geppetto Project*, through which the user will interact with the model.

The Geppetto Backend has a modular architecture that defines multiple service abstractions (figure 1, dashed lines) designed to perform different operations. The specific implementations of these services live in separate modules that can be optionally used by the different applications. For instance Virtual Fly Brain uses the OBJ and SWC [25] *Model Interpreters* while Open Source Brain uses the one for NeuroML (figure 2, 4, 6). New modules that implement these service abstractions can be contributed to expand Geppetto's capabilities. The Geppetto Backend is responsible for loading in memory *Geppetto Projects* and for delegating the user actions that require server side operations to the appropriate services, as specified in the *Geppetto Model*. In this regard the main role of the Geppetto Backend is to orchestrate the interactions of all services available in a particular application. A Geppetto Backend implementation exists for both Java (the reference, fully featured, one) and Python. Different application servers can be used to host the Backend including Virgo [26] for Java and Django [27] or Jupyter [28] for Python. The needs of the specific application will determine the most suitable backend to use, with the Java one currently targeting robust client-server applications aimed at a multi-user deployment (e.g., OSB, VFB) and the Python one also useful for lightweight local deployments aimed at a single-user (e.g., NEURON-UI, NetPyNE-UI).

A central abstract service defined in the Geppetto Backend is the *Model Interpreter*. Specific *Model Interpreter* implementations are used to let Geppetto essentially "understand" a given format representing concepts in the user's original domain – i.e., they allow building instances of the *Geppetto Model Abstraction* from descriptions in the users' domain

language. *Model Interpreters* for popular neuroscience formats such as LEMS [15], NeuroML [14] and NetPyNE [6] are already available.

The abstract *Simulator* service is designed to wrap and control simulators external to Geppetto. The Geppetto backend orchestrates the interactions between *Model Interpreter* and *Simulator* services so that models can be loaded, converted and simulated as result of user operations. Implementations of the *Simulator* service can wrap simulators as external processes or as remote ones running on external servers (e.g. the Neuroscience Gateway supercomputing facilities [29]). A number of computational neuroscience simulators such as NEURON [3] and NetPyNE [6] have already been wrapped and are available for reuse. Following this architecture new simulators can be integrated into Geppetto with relative ease.

Geppetto Data Source services are similarly implemented extending the provided abstract *Data Sources*, and allow Geppetto to pull in data by querying external systems. Multiple *Data Sources* are configurable in the *Geppetto Model* making it possible to use Geppetto as a data integration platform. Data fetched from external *Data Sources* can be post-processed to create a representation of the data once again compatible with the *Geppetto Model Abstraction*. Virtual Fly Brain [30], a hub for *Drosophila melanogaster* nervous system data built using Geppetto (discussed in detail in the Results section) uses two different implementations of the *Data Source* service, one for Neo4j [31] and one for AberOWL [32], to fetch data from their pre-existing data pipeline. Other *Data Source* services for other types of remote servers could be implemented following these existing examples and the same architecture.

For scenarios where user authentication is required and user data needs to be persisted, the *Data Manager* service can be used by developers to configure the Backend to enable authentication and database persistence of the *Geppetto Projects* and simulation results.

The Geppetto Frontend is responsible for presenting the models and data to the user and for allowing them to interact with the application and its workflows. The Geppetto Frontend offers a set of controls and components (figure 1) to build the user interface of Geppetto based applications. While controls (e.g., buttons, dropdowns, dialogs etc.) are generic and data agnostic building blocks, components are more complex constructs that can be used to display data (e.g. 3D, time series, connectivity, MRI, big images, stack, etc.) or to enable specific workflows (e.g., Control Panel, Search, Query, etc.). Components are built using various lower level JavaScript open source libraries (e.g. [33–36]) and are designed to integrate with the *Geppetto Model* using a specific API. Any component can be optionally created inside a draggable dialog window to facilitate data presentation. Components inside these windows are referred to in Geppetto as *Widgets*.

Geppetto Extensions let developers decide what controls and components they need for their specific application, control the layout and look and feel and also create additional domain specific custom components (*Extensions* are represented by the black boxes in Figure 1). Geppetto only loads the user interface components specified in the *Geppetto Extension* of a given application. A default *Extension* is provided as an example and is accessible via

Europe PMC Funders Author Manuscripts

https://live.geppetto.org. By loading the components asynchronously only once the interface needs them, Geppetto optimises the loading times of the application at startup.

Upon receiving a *Geppetto Model* from the Backend, when loading a given *Geppetto Project*, the Frontend will instantiate it. Instantiated Geppetto *Types* are mapped to JavaScript objects (e.g. a population of one cell *Type* would become a JavaScript array containing *Instances* of that *Type*) and augmented with specific *Capabilities* which confer on them the ability to be accessed via a specific API. So for instance, if a *Model Interpreter* in the Backend defined a custom *Type* including a *State Variable,* upon instantiation in the Frontend this would become a JavaScript object with an injected *StateVariableCapability* containing methods specific for state variables, e.g., *getUnit()*, *getInitialValue()*, etc. This has the advantage of giving developers the ability to build UI components that can interact with the *Geppetto Model* in an object oriented way, and allow all the user operations to be fully scriptable, reproducible and testable (e.g. a user interface button designed to plot a state variable would call *Plot.plotData(myStateVariable.getTimeSeries()).* The same principles apply when a custom *Type* defining a cell morphology (*Values* like *Sphere* and *Cylinder* are available to this end in the *Geppetto Model Abstraction*) is sent to the Frontend and passed to the *3D Canvas* component using its API for display. Geppetto has the ability to either visualize a single instance of a *Type* (a cell morphology in this example) or an entire population based on it, depending on whether the *Model Interpreter* responsible for the creation of the model instantiated the *Type* only once or multiple times through an *ArrayType*. In some cases, as with the *Stack Viewer* which connects directly to an IIP3D Server [37], it might be preferable for the UI components to read directly a specific format without requiring a mapping to the *Geppetto Model*, which is also permitted by the architecture.

Code availability

Geppetto is open source (http://git.geppetto.org) and released under the MIT license. Documentation is available at http://docs.geppetto.org. A live demo application to showcase the latest release of Geppetto (0.4.0 at the time of writing, new versions are released monthly) is available at http://live.geppetto.org. Docker images are available for Geppetto at http://docker.geppetto.org, which simplify creation of a local instance of the application with all required libraries preconfigured. Integration tests for the full stack and for the user interface are available for all the main features. These tests are automatically executed after every commit.

Results

In this section we present four examples of neuroscience applications that have been built using Geppetto. Thanks to Geppetto's open source model, many of the features and components described in the Methods section have evolved in concert with the development of these applications in order to satisfy their requirements. Each of the applications have their own *Extension,* where their custom functionality is specified, and a specific deployment configuration. While the first two, Open Source Brain and Virtual Fly Brain, use the Java Backend and are deployed on public web servers where multiple users can access

Europe PMC Funders Author Manuscripts

them simultaneously, the last two, NEURON-UI and NetPyNE-UI, use a Python Backend and are designed to be local deployments aimed at a single user, similar to traditional client applications. Geppetto is currently being used to build a total of seven neuroscience applications [30,38–43].

Open Source Brain

Open Source Brain (OSB, http://www.opensourcebrain.org) is a platform for visualizing, simulating, disseminating and collaboratively developing standardized, biophysically detailed models of neurons and circuits [44]. OSB contains a range of published neuronal and circuit models from multiple brain regions including the neocortex, cerebellum and hippocampus as well as invertebrate neuron models. Model components (e.g. point neuron or morphologically detailed cell models including membrane conductances, synapses, 3D network structures) are contained in user-created projects, each linked to a public code sharing repository (normally hosted on GitHub) that holds the model source code, specified in NeuroML, a widely used model description format for computational neuroscience [14,15]. OSB provides an integrated browser-based workspace that captures many of the infrastructural demands of projects in computational neuroscience, and allows users to interact with the underlying neuronal models through a graphical interface, without requiring programming knowledge or installing and configuring simulators.

Figure 2 shows how Geppetto is configured for OSB. Many aspects of Geppetto's functionality have been developed to provide the core functionality for OSB. The NeuroML *Model Interpreter* and the LEMS *Conversion* services were contributed to Geppetto to deal with the NeuroML and LEMS formats, reusing previously developed libraries [15]. The NeuroML *Model Interpreter* allows standardized model descriptions to be loaded into the OSB Geppetto deployment, providing automatic 3D visualization of morphologies and internal structure of models, such as state variables and parameters (figure 3a) and connectivity within the network (figure 3b). Structured metadata in the NeuroML files can be extracted, as well as the underlying mathematical expressions of dynamical components in the model (e.g. kinetics of membrane conductances). These data are made available in an accessible format to the user through a custom *Extension* to the Geppetto Frontend.

This OSB custom extension to Geppetto adds shortcuts and menu options for interacting with models, running simulations, and visualizing their results. A summary of information extracted from the NeuroML model can be accessed through a "Model Description" widget, which includes links to the source file and original data sources, giving model provenance. This widget also provides easy access to neuronal model specific functionality, such as plotting rates of activation and inactivation for ion channels and overlaying locations and densities of active conductances on neuronal morphologies (bottom right, Figure 3a). A shortcut to the *Connectivity Widget* allows the user to see synaptic connectivity of models at a glance: as a chord diagram (bottom left, Figure 3a), connectivity matrix with weights (bottom right, Figure 3b), force-directed graph, or hive plot. Key parameters present on any given model are thus automatically exposed in a format familiar to neuroscientists.

The simulator agnostic NeuroML format can be converted to simulator-specific formats such as NEURON [3] using a suite of existing converters that implement the Geppetto conversion

service interface (figure 2). Geppetto's external simulator abstraction allows OSB to transparently interface with these converters and their associated simulators, allowing models to be simulated through a simple interface. Geppetto can either dispatch simulator jobs to the Neuroscience Gateway [29], a high-performance computing facility, or run them on OSB servers. The extension provides assistance for simulation workflows; basic protocols can be defined that create batched experiments with a given range of parameters or the user can record all membrane potentials with a single click. Upon completion, the data generated are sent to the browser for visualization using a Geppetto's plotting widget (top left, Figure 3a), or recorded membrane potentials or calcium concentrations can be visualized by pseudocoloring the morphologies to show changes over the course of a simulation, and the simulation can be replayed at various speeds. Alternatively, the raw results can be downloaded or automatically uploaded to Dropbox via Geppetto's dropbox interface functionality. Experiments run asynchronously on remote servers, so users do not need to keep their browser open.

The configurable functionality of Geppetto middleware enables OSB to make models accessible, opening them up to critical scientific scrutiny by a wide range of neuroscientists. This supports the process of ongoing model evolution, which is aided by OSB's deep link to GitHub [45], preventing model development from becoming arrested at the point of publication. OSB therefore provides a resource of robust models that can function as best practice examples for model sharing for the neuroscience community.

In addition to this research aspect, OSB also leverages Geppetto's tutorial component to provide interactive computational neuroscience tutorials aimed at students. These tutorials allow users to run virtual experiments and protocols through an easy to use web interface, allowing basic concepts in neurophysiology and computational neuroscience to be taught without installing simulators or writing code.

Virtual Fly Brain

Virtual Fly Brain (VFB, http://virtualflybrain.org) is a hub for *Drosophila melanogaster* neuroscience research which was born from the need to make the newly standardised fly neuroanatomy available to the public [47–49]. Along with extensive curation of literature in collaboration with FlyBase [50], VFB v1 allowed users to explore labeled confocal immunofluorescent slices of the adult fly brain across the internet. The user could step through the brain and identify anatomy by hovering over it. Later this expanded to include expression, transgene and single neuron image data published by multiple labs that was aligned to the same template brain enabling any of the 40,000 images to be overlayed. Whilst most researchers were used to viewing slices through the brain, with more single neurons appearing as tiny points in cross section, interpreting the morphology was increasingly difficult without a 3D representation.

VFB v2 was designed to provide access to all the complex queries and data an expert might require within an interface a novice can easily navigate. Geppetto's existing 3D browser infrastructure atop a flexible modelling framework was used to enable VFB to run complex queries across multiple backend service APIs whilst maintaining an easy to use UI. The *Geppetto Model* has been utilized to provide abstraction of the specifics of 3rd party API

configuration, query construction and representation into a single simple human-readable file.

Display of the original immunofluorescent confocal gene expression data was implemented in Geppetto as point cloud renderings while the OBJ *Model Interpreter* was reused to display anatomy regions as surface renderings (figure 5a).

A *Model Interpreter* for the SWC format [25] was added to display segmented reconstructions of neuronal morphologies. A *Stack Viewer* was contributed to display 2D confocal microscopy data and synchronized with the pre-existing 3D Canvas component. Geppetto's ability to load data on demand and to optimise the visualisation of neurons as tubes or traced lines was essential for VFB to efficiently display larger amounts of imaging data on the screen. The ability to query third party RESTful APIs through the *Data Source* services allowed VFB to fetch remote data running complex queries (figure 5b) involving multiple configurable Data Sources (figure 1). VFB currently pulls data via an ontology reasoner (OWL-ELK [32]) as well as a graph database (Neo4j [31]). Geppetto's Control Panel and Search components were reused and customised within VFB's Geppetto *Extension* to show custom fields and to provide autocompletion search results utilising a (SOLR [51]) indexing server.

NEURON-UI and NetPyNE-UI

NEURON is a widely used simulator in the neural multiscale modeling domain, allowing models to be built that link reaction-diffusion dynamics at the molecular level, to neuronal electrophysiology, up to the large scale network level [3,6,52,53]. It has thousands of users, a model database [54] with over 600 models, and over 1900 NEURON-based publications. NEURON is being used by major brain research initiatives such as the Human Brain Project and the Allen Institute [18,55]. NEURON includes a native graphical user interface for model construction and control, which while fully functional has limited usability and graphical capabilities and is based on deprecated libraries (Interviews) originally developed in the 1980s.

NetPyNE [56] is a high-level Python interface to NEURON that facilitates the development, simulation and analysis of biologically detailed neuronal networks. It provides a unique high-level declarative language designed to facilitate the definition of data-driven multiscale models (e.g., a concise set of connectivity rules vs. millions of explicit cell-to-cell connections). The user can then easily generate NEURON network instances from these specifications, run efficient simulations (including on high performance parallel computing resources) and exploit the wide array of built-in analysis functions. Its standardized format – compatible with NeuroML – makes it easier to understand, reproduce and reuse models. NetPyNE is being used to develop models of different brain regions – e.g. thalamus, cortex and hippocampus – and phenomena – e.g. neural coding and brain disorders [6,57].

Geppetto has been used to build user interfaces for both NEURON and NetPyNE. The two applications, designed to be installed and used locally by a single user, have in common an architecture based on the Geppetto interactive Python Backend. This Backend is implemented as a Jupyter Notebook [28] extension which provides direct communication

with the Python kernel. By defining a set of component extensions, Geppetto's interactive Python Backend makes it possible to synchronize the data model underlying the user interface with a custom Python model. This functionality is at the heart of both NEURON-UI and NetPyNE-UI and means any change made to the Python kernel is immediately reflected in the user interface and vice versa.

Although NEURON-UI and NetPyNE-UI share the same architecture (figure 6 gives an overview of the Geppetto components used in NetPyNE-UI), they differ in certain aspects. In NEURON-UI the graphical interface is created using a custom Python API meant to mimic NEURON's Interviews based API. The panels, buttons and text boxes in the user interface are therefore created from Python and mapped to Geppetto UI components (figure 7a). These components are then connected to the internal Geppetto API to visualize the cells and the networks, run the simulations and plot the results. The idea behind this approach was to retain backward compatibility with the numerous existing NEURON interfaces built with Interviews for various models. Our future aim is to fully map the NEURON API to our NEURON-UI therefore providing a comprehensive alternative to the traditional user interface.

In contrast, in NetPyNE-UI the user interface is defined entirely in JavaScript inside its Geppetto extension. This offers a flexible and intuitive way to create advanced layouts while still enabling each of the elements of the interface to be synchronized with the Python model. The user interface splits the workflows in three tabs: network definition, network exploration and network simulation and analysis (figure 7b). From the first tab it is possible to define -- or import via Python -- the high-level network parameters and rules that will be used for its generation. In the second and third tabs Geppetto's 3D Canvas is used to visualize the instantiated network. The third tab lets the user simulate the instantiated model (this tab is selected in Figure 7b). Geppetto allows NetPyNE-UI also to display on the browser a number of plots that are defined in NetPyNE using matplotlib for network analysis and simulation. Both NEURON-UI and NetPyNE-UI can be installed via pip [58] or used inside provided Docker images.

The new Geppetto-based UIs will make NEURON and NetPyNE accessible to a wider range of researchers and students, including those with limited programming experience. This will enable experimentalists to better collaborate with modelers, or to directly reproduce and explore their own experiments via computational simulations.

Discussion

We have developed Geppetto, an open source middleware platform for building accessible neuroscience applications. Geppetto facilitates the development of complex applications by providing a well-tested, reusable set of building blocks to integrate diverse neuroscience data, models and simulators. Geppetto provides a modular Frontend, where multiple customizable user interface components and *Widgets* make it possible to visualise and analyse models and data, as well as a Backend capable of connecting to multiple data sources and lower level, domain specific descriptions and simulators. This was made possible by designing the *Geppetto Model Abstraction* that can be used to represent a variety

of neuroscience domain models, linked to a modular web-based architecture engineered using various open-source libraries. Geppetto has been used as the basis of a number of online and desktop applications in neuroscience: Open Source Brain, Virtual Fly Brain, NEURON-UI and NetPyNE UI described here, as well as Patient H.M. [38], WormSim [40] and SciDash [43].

Neuroscience applications are typically developed independently, to address a specific requirement. This leads to considerable redundancy with the same functionality being redesigned and implemented over and over again [59–65]. This approach is only justifiable when the shared set of features is negligible. In this paper, we have shown that even for applications whose requirements were specified independently and had minimal overlap, there can be a significant degree of shared infrastructure. Geppetto proposes an alternative approach by exploiting this fact, allowing neuroscience applications to be built from reusable modules – as illustrated by the overlapping blocks in Figures 2, 4 and 6. This strategy fits naturally into the open source model – components and modules are more likely to be reusable compared to monoliths – making Geppetto a flexible and extensible solution for multiple applications in neuroscience.

As middleware that factors out commonalities between different domains, Geppetto's modular structure enables a high level of reuse, allowing developers to skip to writing only code specific to their neuroscience application resulting in a considerable saving of time. As with all software platforms, Geppetto has its own learning curve required for developers to understand its architecture and become familiar with its components. While at first this initial investment might be seen as a complication compared to the apparent ease of starting from a blank slate, developers associated with the applications described above, with no previous experience on Geppetto, have found it only takes from one to four weeks[1] to become productive. This time investment is outweighed by the subsequent savings made in avoiding common pitfalls, replicating solutions to common problems and rewriting entire software components and workflows. There is also a significant advantage in interacting with the active community of Geppetto developers, who can assist with any queries. The net time saving compared to an approach that starts from scratch is difficult to estimate but is likely to range from six months to five years[2] depending on the targeted scope – the more the features required that overlap with Geppetto's the bigger the savings – and on the size and experience of the team of developers involved. Moreover, extensive sharing of modules between applications, results in them being thoroughly tested [66], while having a shared infrastructure that undergoes regular release cycles ensures maintenance is less burdensome for each specific application. Furthermore, the distributed nature of the Geppetto code base and the fact that updates are made independently of any specific project ultimately increases the longevity of any application built with this platform.

[1]Depending on the background and level of experience of the developer.
[2]Estimate based on the actual time that was spent designing and implementing various reusable components, e.g. 3D Canvas 6 months, MRI Viewer 3 months, Plotting widget 6 months, Connectivity Widget 5 months, Stack Viewer 6 months, Control Panel 3 months, Geppetto Model Abstraction 9 months, etc. Building of the infrastructure in its current form took 3 years. All these figures consider 1 Senior Development Engineer FTE.

The diversity of applications that have been built so far with Geppetto illustrates the flexibility of its model abstraction capabilities, which can encompass different domains, data and scientific modelling formalisms. Also, as the platform keeps evolving, new solutions added for a specific application become immediately available to all the other applications. Examples of this include many of the features contributed by Open Source Brain being reused by multiple applications (e.g., *Control Panel*, *Search Bar* or the *Experiments table*); the SWC [25] *Model Interpreter* contributed by Virtual Fly Brain, which is reused in Open Source Brain; and the *3D Canvas*, originally built for the first deployment of the platform and reused by every other application to date. Geppetto combines a model driven design with a service oriented architecture to enable reuse across multiple applications. Its modularity, a centerpiece of both the Backend and the Frontend, is obtained by engineering together a unique set of technologies [19,21,22,67,68] to provide novel functionality. By allowing different neuroscience applications to use the same technologies, Geppetto provides well-tested solutions that bring closer together otherwise disjoint research groups – both computational and experimental, thereby fostering collaboration.

The Geppetto applications described in the Results section are in active development. Some of the planned and ongoing projects include: extending OSB to bring together models and the experimental data used to build and test them, by adding standardised data interpreters (e.g. version 2 of the Neurodata Without Borders format); extending VFB to cover all stages/regions of the fly CNS, incorporating synapse level connectomics data with the extensive light level image and literature knowledge; releasing a new version of WormSim, currently being developed within the OpenWorm project [1] that will integrate the Sibernetic [69] fluid dynamics simulator (see Palyanov et al. in current issue) with the NeuroML based nervous system model (see Gleeson et al. in current issue). The latter will be the first instance of a Geppetto application providing a non-computational neuroscience specific numerical engine, used for fluid dynamics simulations (figure 8).

Thanks to its open, modular, web-based architecture, Geppetto ultimately enables the engineering of a new breed of neuroscience applications that can be used in a collaborative way by theoreticians, modelers and experimentalists to formulate new scientific hypotheses, build and validate new models and help gain insights into the most pressing questions in neuroscience.

Acknowledgements

We would like to thank all Geppetto contributors (http://contributors.geppetto.org) and in particular Jesus Martinez who implemented many Geppetto features. We are grateful to the OpenWorm Foundation for hosting the Geppetto repositories and community and in particular to Stephen Larson for stimulating discussions throughout the development of Geppetto, for his valuable input and continued support. We thank William Lytton, Robert McDougal and Michael Hines for their support while developing NEURON-UI. We would also like to thank Carlo Collodi for inspiring the name of the platform with his novel “Pinocchio”.

Funding

The OSB initiative was funded by the Wellcome Trust (086699, 101445, 095667). RAS is in receipt of a WT PRF (203048) and an ERC advanced grant (294667). In addition, the infrastructure to enable integration of OSB and the Neuroscience Gateway was funded by the BBSRC-NSF/BIO program (BB/N005236/1 & NSF #1458840) and NSF #1458495. The VFB project is supported by a grant from the Wellcome Trust: Virtual Fly Brain (208379/Z/17/Z) (October 2017 to September 2021) after Wellcome Trust: Virtual Fly Brain: a global informatics hub for Drosophila neurobiology (WT105023MA) (October 2014 to September 2017). VFB was previously supported by a research

award from the BBSRC to Douglas Armstrong and Michael Ashburner (Cambridge BB/G02233X/1; Edinburgh BB/G02247X/1). A UK e-Science Theme award to Douglas Armstrong helped establish the VFB project. BM is funded by grant 2017/04748-0, São Paulo Research Foundation (FAPESP). NEURON-UI and NetPyNE-UI were funded by NIH U01EB017695, NIH R01MH086638, NIH R01EB022903 and DOH01-C32250GG-3450000.

References

1. Szigeti B, et al. OpenWorm: an open-science approach to modeling Caenorhabditis elegans. Front Comput Neurosci. 2014; 8:137. [PubMed: 25404913]
2. Lamaze A, Öztürk-Çolak A, Fischer R, Peschel N, Koh K, Jepson JEC. Regulation of sleep plasticity by a thermo-sensitive circuit in Drosophila. Sci Rep. 2017; 7:40304. [PubMed: 28084307]
3. Carnevale NT, Hines ML. The NEURON Book. Cambridge University Press; 2006.
4. Gewaltig M-O, Marc-Oliver G, Markus D. NEST (NEural Simulation Tool). Scholarpedia J. 2007; 2:1430.
5. Goodman DFM, Brette R. The brian simulator. Front Neurosci. 2009; 3:192–197. [PubMed: 20011141]
6. Lytton WW, Seidenstein AH, Dura-Bernal S, McDougal RA, Schürmann F, Hines ML. Simulation Neurotechnologies for Advancing Brain Research: Parallelizing Large Networks in NEURON. Neural Comput. 2016; 28:2063–2090. [PubMed: 27557104]
7. Somogyi ET, Bouteiller J-M, Glazier JA, König M, Medley JK, Swat MH, Sauro HM. libRoadRunner: a high performance SBML simulation and analysis library. Bioinformatics. 2015; 31:3315–3321. [PubMed: 26085503]
8. Cannon RC, et al. Interoperability of neuroscience modeling software: current status and future directions. Neuroinformatics. 2007; 5:127–138. [PubMed: 17873374]
9. Hines ML, Davison AP, Muller E. NEURON and Python. Front Neuroinform. 2009; 3doi: 10.3389/neuro.11.001.2009
10. Bednar JA. Topographica: Building and Analyzing Map-Level Simulations from Python, C/C++, MATLAB, NEST, or NEURON Components. Front Neuroinform. 2009; 3:8. [PubMed: 19352443]
11. Muller E, Eilif M, Bednar JA, Markus D, Marc-Oliver G, Michael H, Davison AP. Python in neuroscience. Front Neuroinform. 2015; 9doi: 10.3389/fninf.2015.00011
12. Dura-Bernal S, Zhou X, Neymotin SA, Przekwas A, Francis JT, Lytton W. Cortical spiking network interfaced with virtual musculoskeletal arm and robotic arm. Front Neurorobot. 2015; 9doi: 10.3389/fnbot.2015.00013
13. Bennett KH, Rajlich VT. Software maintenance and evolution: a roadmap. Proceedings of the Conference on The Future of Software Engineering; ACM; 2000. 73–87.
14. Gleeson P, et al. NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. PLoS Comput Biol. 2010; 6:e1000815. [PubMed: 20585541]
15. Cannon RC, Gleeson P, Crook S, Ganapathy G, Marin B, Piasini E, Silver RA. LEMS: a language for expressing complex biological models in concise and hierarchical form and its use in underpinning NeuroML 2. Front Neuroinform. 2014; 8:79. [PubMed: 25309419]
16. Teeters JL, et al. Neurodata Without Borders: Creating a Common Data Format for Neurophysiology. Neuron. 2015; 88:629–634. [PubMed: 26590340]
17. Djurfeldt M, et al. Run-time interoperability between neuronal network simulators based on the MUSIC framework. Neuroinformatics. 2010; 8:43–60. [PubMed: 20195795]
18. Markram H, et al. Reconstruction and Simulation of Neocortical Microcircuitry. Cell. 2015; 163:456–492. [PubMed: 26451489]
19. [accessed on 8 March 2018] React - A JavaScript library for building user interfaces. See https://reactjs.org/
20. [accessed on 28 February 2018] npm. See https://www.npmjs.com/
21. [accessed on 28 February 2018] Webpack. See https://webpack.js.org/
22. Gronback RC. Eclipse Modeling Project. Addison-Wesley Professional; 2009.
23. [accessed on 24 February 2018] org geppetto model. See https://github.com/openworm/org.geppetto.model

24. pygeppetto. Github; See https://github.com/openworm/pygeppetto
25. [accessed on 24 February 2018] SWC - Specification. See http://www.neuronland.org/NLMorphologyConverter/MorphologyFormats/SWC/Spec.html
26. [accessed on 25 February 2018] Eclipse Virgo. See http://www.eclipse.org/virgo/
27. Django D. Django The Web Framework. Culver City. California. Estados Unidos: 2007.
28. [accessed on 25 February 2018] IOS Press Ebooks - Jupyter Notebooks - a publishing format for reproducible computational workflows. See http://ebooks.iospress.nl/publication/42900
29. Sivagnanam S, Majumdar A, Yoshimoto K. Introducing the Neuroscience Gateway. 2013
30. VFB. [accessed on 24 February 2018] Virtual Fly Brain, a hub for Drosophila melanogaster neuroscience research. See http://v2.virtualflybrain.org
31. Developers N. Neo4J. Graph NoSQL Database. 2012. [online]
32. Slater L, Gkoutos GV, Schofield PN, Hoehndorf R. 2015 AberOWL: an ontology portal with OWL EL reasoning. In *mons Database We would like to acknowledge all the organizing committee, the program committee, the additional reviewers, the volunteers, authors and attendees of ICBO, and EasyChair for providing such a useful conference management tool for free. Finally, we would like to thank all our sponsors.*, p. 127.
33. three.js. [accessed on 10 March 2018] Javascript 3D library. See https://threejs.org/
34. [accessed on 10 March 2018] Modern Visualization for the Data Era. See https://plot.ly/
35. ami. Github; See https://github.com/FNNDSC/ami
36. Bostock M, Ogievetsky V, Heer J. D^3: Data-Driven Documents. IEEE Trans Vis Comput Graph. 2011; 17:2301–2309. [PubMed: 22034350]
37. Husz ZL, Burton N, Hill B, Milyaev N, Baldock RA. Web tools for large-scale 3D biological images and atlases. BMC Bioinformatics. 2012; 13:122. [PubMed: 22676296]
38. Patient HM. [accessed on 24 February 2018] THE BRAIN OBSERVATORY®. See https://www.thebrainobservatory.org/project-hm/
39. [accessed on 24 February 2018] Open Source Brain. See http://opensourcebrain.org
40. [accessed on 24 February 2018] WormSim. See http://wormsim.org
41. NetPyNE-UI. Github; See https://github.com/MetaCell/NetPyNE-UI
42. NEURON-UI. Github; See https://github.com/MetaCell/NEURON-UI
43. [accessed on 24 February 2018] SciDash. See http://scidash.org
44. Gleeson P, et al. Open Source Brain: a collaborative resource for visualizing, analyzing, simulating and developing standardized models of neurons and circuits. bioRxiv. 2018; 229484. doi: 10.1101/229484
45. GitHub. Github; See https://github.com
46. Traub RD, et al. Single-column thalamocortical network model exhibiting gamma oscillations, sleep spindles, and epileptogenic bursts. J Neurophysiol. 2005; 93:2194–2232. [PubMed: 15525801]
47. Armstrong JD, van Hemert JI. Towards a virtual fly brain. Philos Trans A Math Phys Eng Sci. 2009; 367:2387–2397. [PubMed: 19414461]
48. Costa M, Reeve S, Grumbling G, Osumi-Sutherland D. The Drosophila anatomy ontology. J Biomed Semantics. 2013; 4:32. [PubMed: 24139062]
49. Ito K, et al. A Systematic Nomenclature for the Insect Brain. Neuron. 2014; 81:755–765. [PubMed: 24559671]
50. Tweedie S, et al. FlyBase: enhancing Drosophila gene ontology annotations. Nucleic Acids Res. 2008; 37:D555–D559. [PubMed: 18948289]
51. Smiley D, Pugh E, Parisa K, Mitchell M. Apache Solr Enterprise Search Server. Third Edition. Packt Publishing Ltd; 2015.
52. Tikidji-Hamburyan RA, Narayana V, Bozkus Z, El-Ghazawi TA. Software for Brain Network Simulations: A Comparative Study. Front Neuroinform. 2017; 11:46. [PubMed: 28775687]
53. McDougal R, Hines M, Lytton W. Reaction-diffusion in the NEURON simulator. Front Neuroinform. 2013; 7:28. [PubMed: 24298253]
54. [accessed on 28 February 2018] ModelDB: Home. See https://senselab.med.yale.edu/modeldb/

55. Hawrylycz M, et al. Inferring cortical function in the mouse visual system through large-scale systems neuroscience. Proc Natl Acad Sci U S A. 2016; 113:7337–7344. [PubMed: 27382147]
56. Dura-Bernal S, Suter B, Neymotin S, Kerr C, Quintana A, Gleeson P, Shepherd GMG, Lytton WW. NetPyNE: a Python package for NEURON to facilitate development and parallel simulation of biological neuronal networks. Computational Neuroscience (CNS); 2016.
57. Dura-Bernal S, Neymotin SA, Suter BA, Shepherd G. Long-range inputs and H-current regulate different modes of operation in a multiscale model of mouse M1 microcircuits. bioRxiv. 2018
58. [accessed on 1 March 2018] pip Python Package Index. See https://pypi.python.org/pypi/pip
59. Mark E. The Whole Brain Catalog: Platform for neuroscientific data integration. Front Neuroinform. 2011; 5doi: 10.3389/conf.fninf.2011.08.00137
60. Ukani NH, et al. The Fruit Fly Brain Observatory: from structure to function. bioRxiv. 2016; 092288. doi: 10.1101/092288
61. Sunkin SM, Ng L, Lau C, Dolbeare T, Gilbert TL, Thompson CL, Hawrylycz M, Dang C. Allen Brain Atlas: an integrated spatio-temporal portal for exploring the central nervous system. Nucleic Acids Res. 2013; 41:D996–D1008. [PubMed: 23193282]
62. Cessac B, Kornprobst P, Kraria S, Nasser H, Pamplona D, Portelli G, Viéville T. PRANAS: A New Platform for Retinal Analysis and Simulation. Front Neuroinform. 2017; 11:49. [PubMed: 28919854]
63. Sanz Leon P, Knock SA, Woodman MM, Domide L, Mersmann J, McIntosh AR, Jirsa V. The Virtual Brain: a simulator of primate brain network dynamics. Front Neuroinform. 2013; 7:10. [PubMed: 23781198]
64. [accessed on 8 March 2018] Brain Simulation Platform. See https://www.humanbrainproject.eu/en/brain-simulation/brain-simulation-platform/
65. Tomita M, et al. E-CELL: software environment for whole-cell simulation. Bioinformatics. 1999; 15:72–84. [PubMed: 10068694]
66. Wikipedia contributors. [accessed on 8 March 2018] Linus's Law. Wikipedia, The Free Encyclopedia. 2018. See https://en.wikipedia.org/w/index.php?title=Linus%27s_Law&oldid=818874010
67. Alliance O. Osgi Service Platform, Release 3. IOS Press, Inc; 2003.
68. [accessed on 11 March 2018] Spring. See https://spring.io/
69. Palyanov A, Khayrulin S, Larson SD. Application of smoothed particle hydrodynamics to modeling mechanisms of biological tissue. Adv Eng Softw. 2016; 98:1–11.

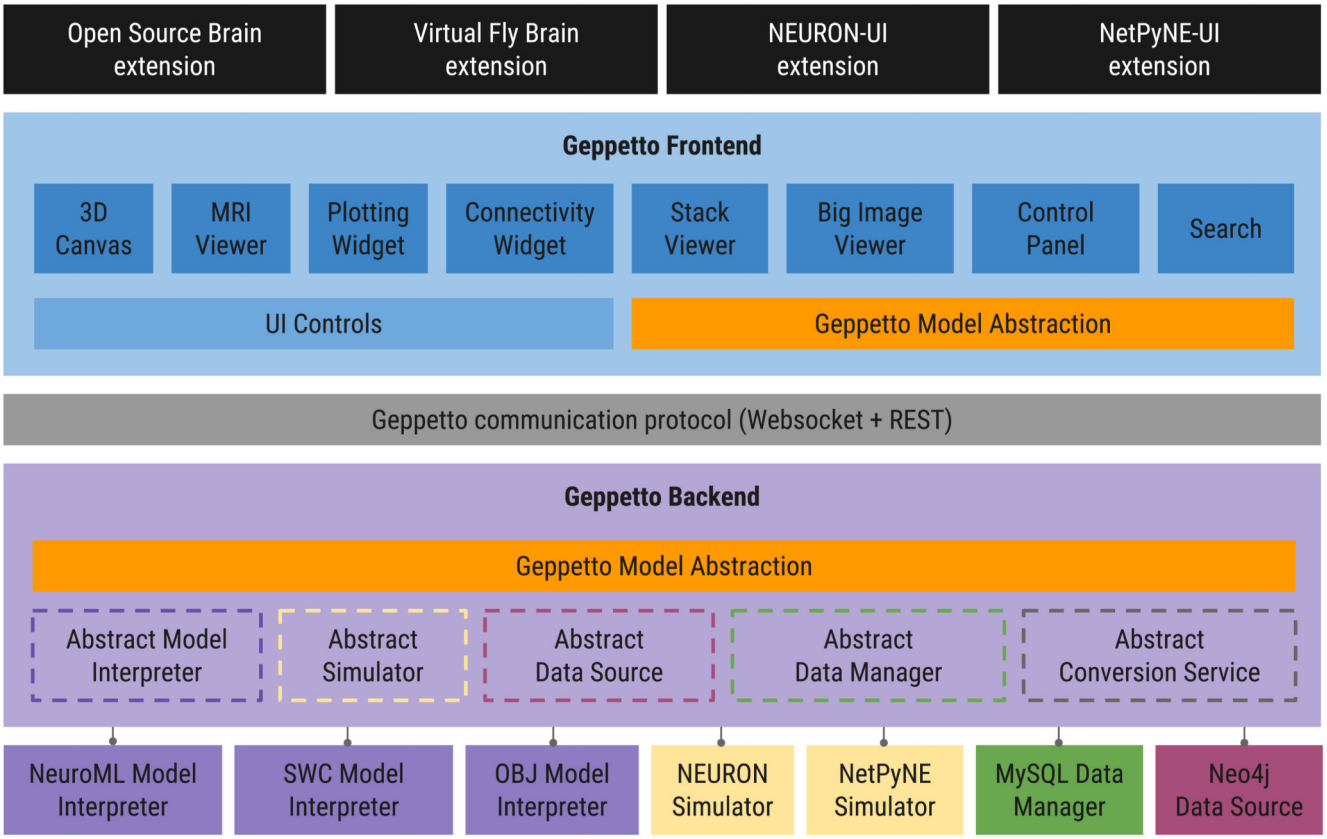

Figure 1. Geppetto Architecture.

Graphical representation of the components of Geppetto illustrating how the *Geppetto Model Abstraction* (orange blocks) allows Backend model and data sources to be accessed by users through browser-based Frontend components. Black blocks in the figure are Geppetto *Extensions*, used by applications built on top of the Geppetto platform. The Geppetto Frontend (shades of blue) is shown containing a diverse set of visualization components. Communication between the Frontend and Backend happens via Websockets and a REST-API layer (grey block). The Geppetto Backend (light purple block) orchestrates the various services available in a given Geppetto application, including specific *Model Interpreters* (dark purple blocks), external *Simulators* (cream blocks), *Data Managers* (green) and *Data Sources* (pink).

Europe PMC Funders Author Manuscripts

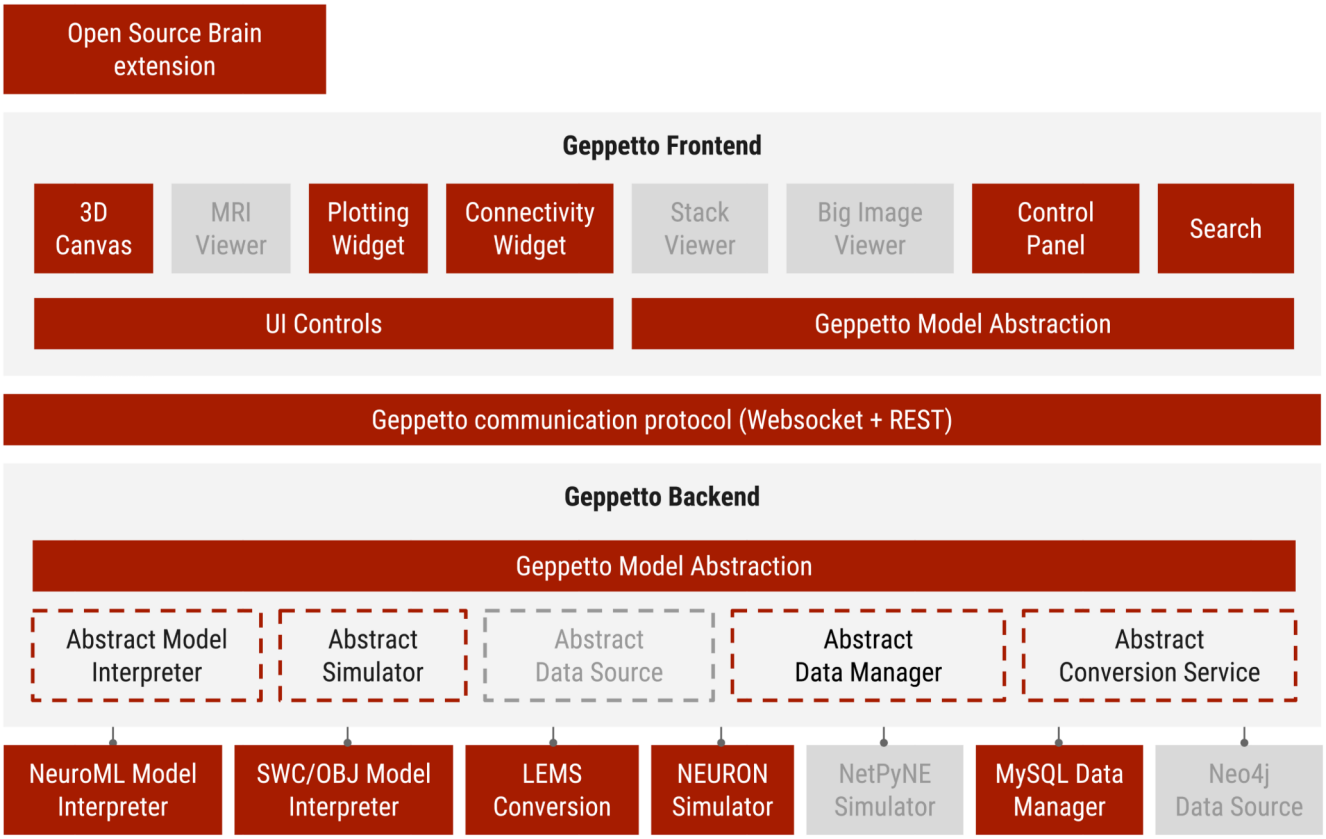

Figure 2. Geppetto OSB configuration.

Graphical representation of the components of Geppetto that are used on the OSB application (red). The ones not used are coloured in grey.

Europe PMC Funders Author Manuscripts

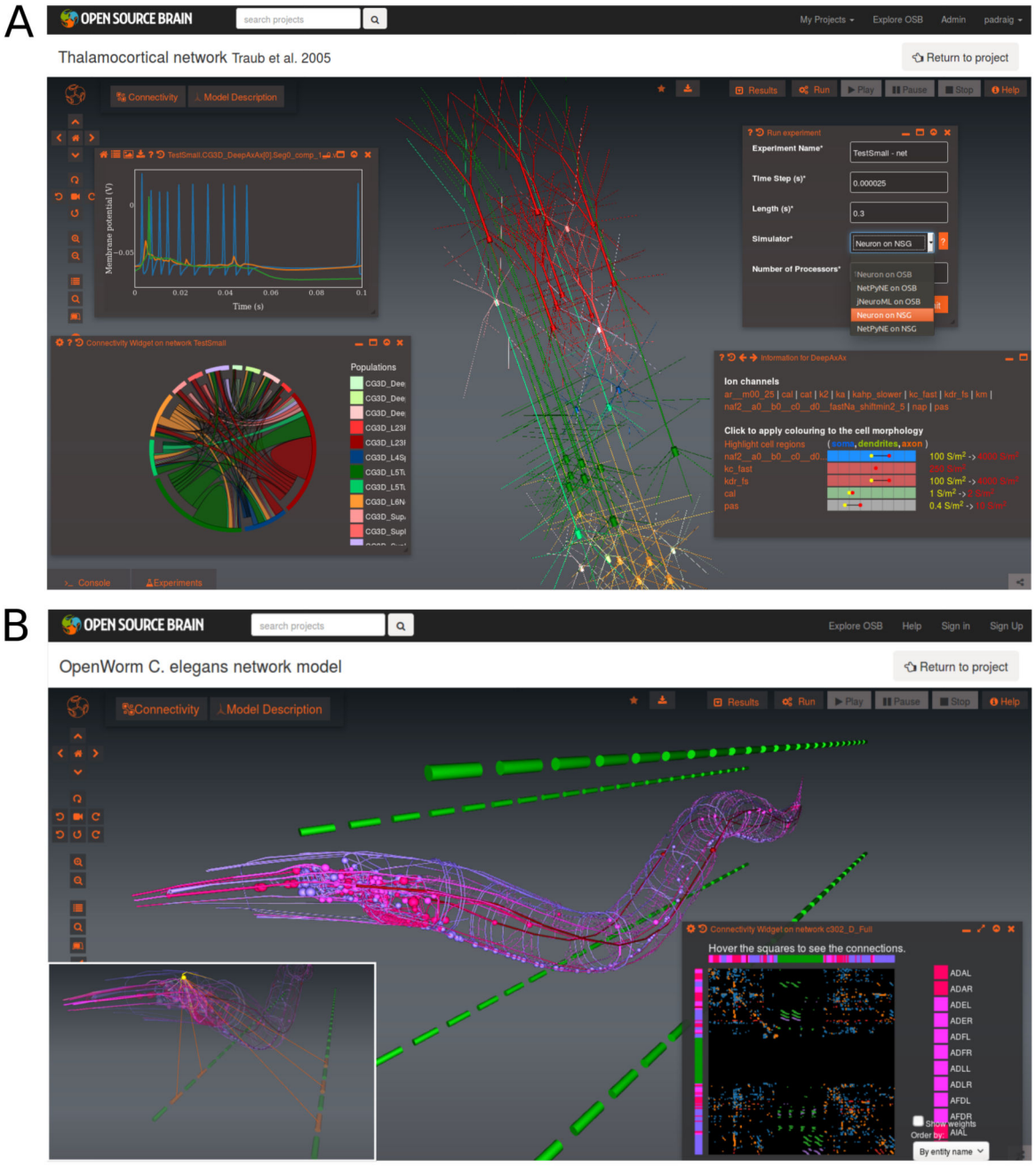

Figure 3.

(a) Screenshot of a reduced thalamocortical network model [46] on OSB showing analysis and simulation widgets provided by Geppetto and the Geppetto Frontend OSB extension. Centre of screen shows 3D rendering of the 12 populations of pyramidal cells and interneurons. Widgets shown are (clockwise from top-left): plot showing recorded membrane potentials from 3 cells of a previously run experiment; Run dialog for selecting simulators and running experiments; widget showing ion channels and their densities for a single cell model; chord diagram showing connectivity between populations. **(b)**

Europe PMC Funders Author Manuscripts

Visualisation of the neuronal network model of *C. elegans* being developed by the OpenWorm project. Centre of screen shows 302 neurons (red: interneurons; pink: sensory; purple: motor neurons) and four quadrants of body wall muscles (green) located away from the body for clarity. Connectivity widget on lower right shows chemical synapses between individual neurons/muscles. Inset on lower left illustrates interactive exploration of network; selecting a single motor neuron (RMED in head) highlights the neurons connected to it, along with 5 muscles in two of the ventral quadrants.

Europe PMC Funders Author Manuscripts

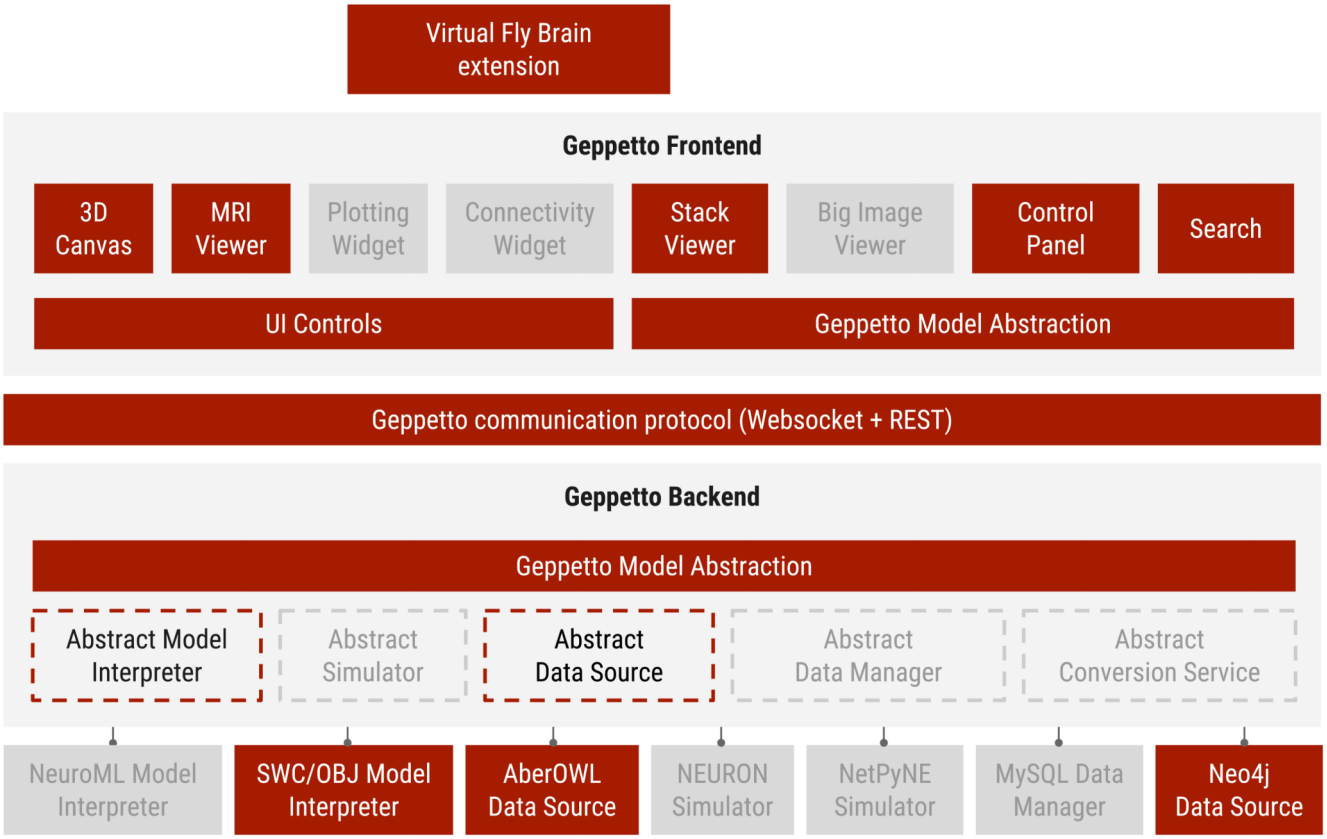

Figure 4. Geppetto VFB configuration.

Graphical representation of the components of Geppetto that are used on the VFB application (red). The ones not used are coloured in grey.

Europe PMC Funders Author Manuscripts

Europe PMC Funders Author Manuscripts

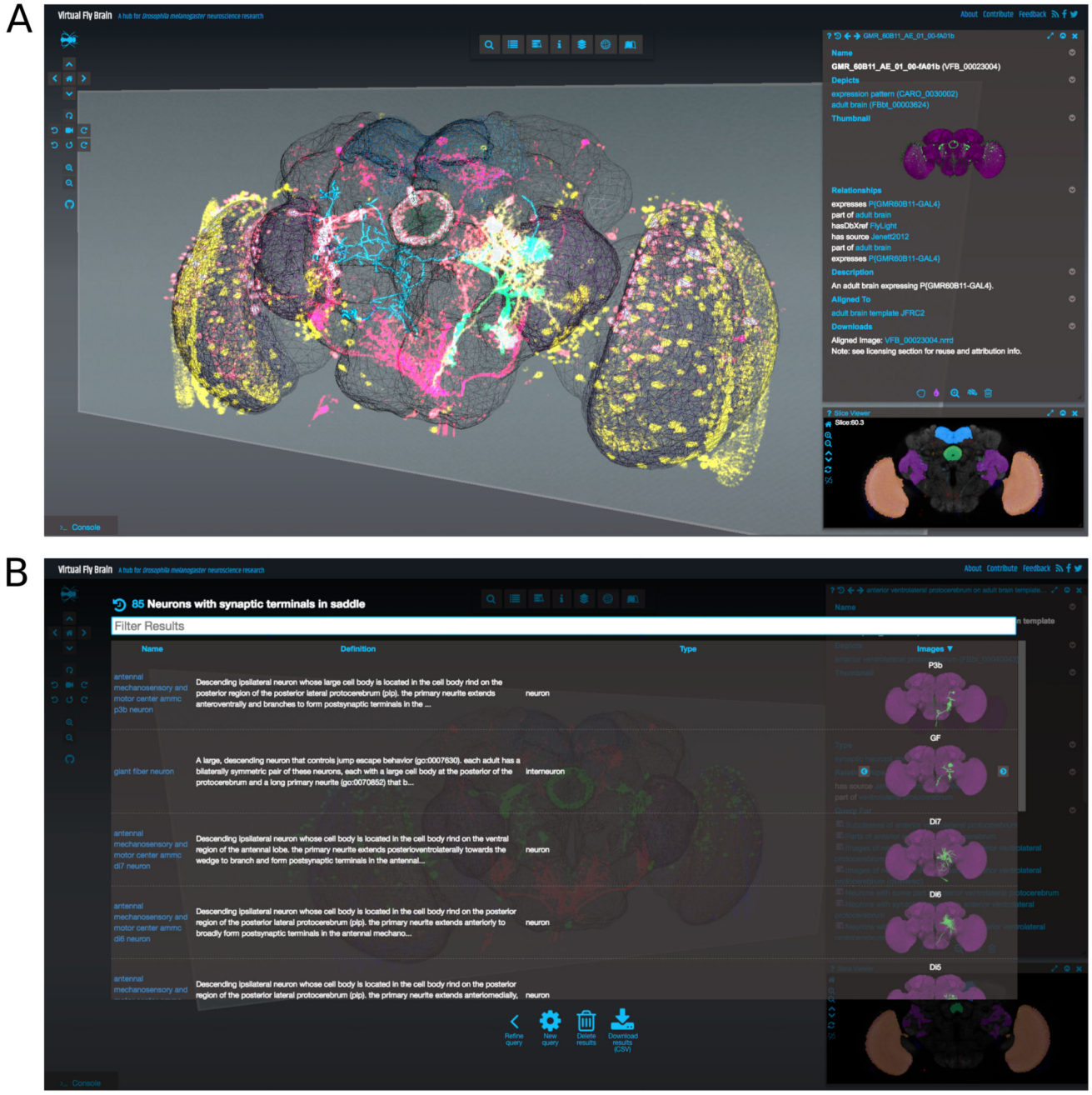

Figure 5.

(a) Virtual Fly Brain main view shows a reference template for the *Drosophila melanogaster* from Janelia Research Campus using the *3D Canvas*. Superposed on the template are various gene expression patterns visualized as point clouds, reconstructed neurons and segmented neuropil regions. At the bottom right corner the *Stack Viewer* shows a frontal slice through the superposed confocal microscopy images. The Stack Widget is fully synchronized with the 3D Canvas and a moving 3D plane indicates the specific slice currently displayed. On top of the Stack Widget a Geppetto viewer is used to display the ontological information

associated with the current selection. **(b)** Geppetto's Query component is used to display the results of queries that can be executed from the UI, in this case the user interface shows all the neurons with synaptic terminals in the saddle. By clicking on the thumbnails the selected neuron is loaded on demand and visualized in the *3D Canvas*, the *Stack Viewer* and the textual definition.

Europe PMC Funders Author Manuscripts

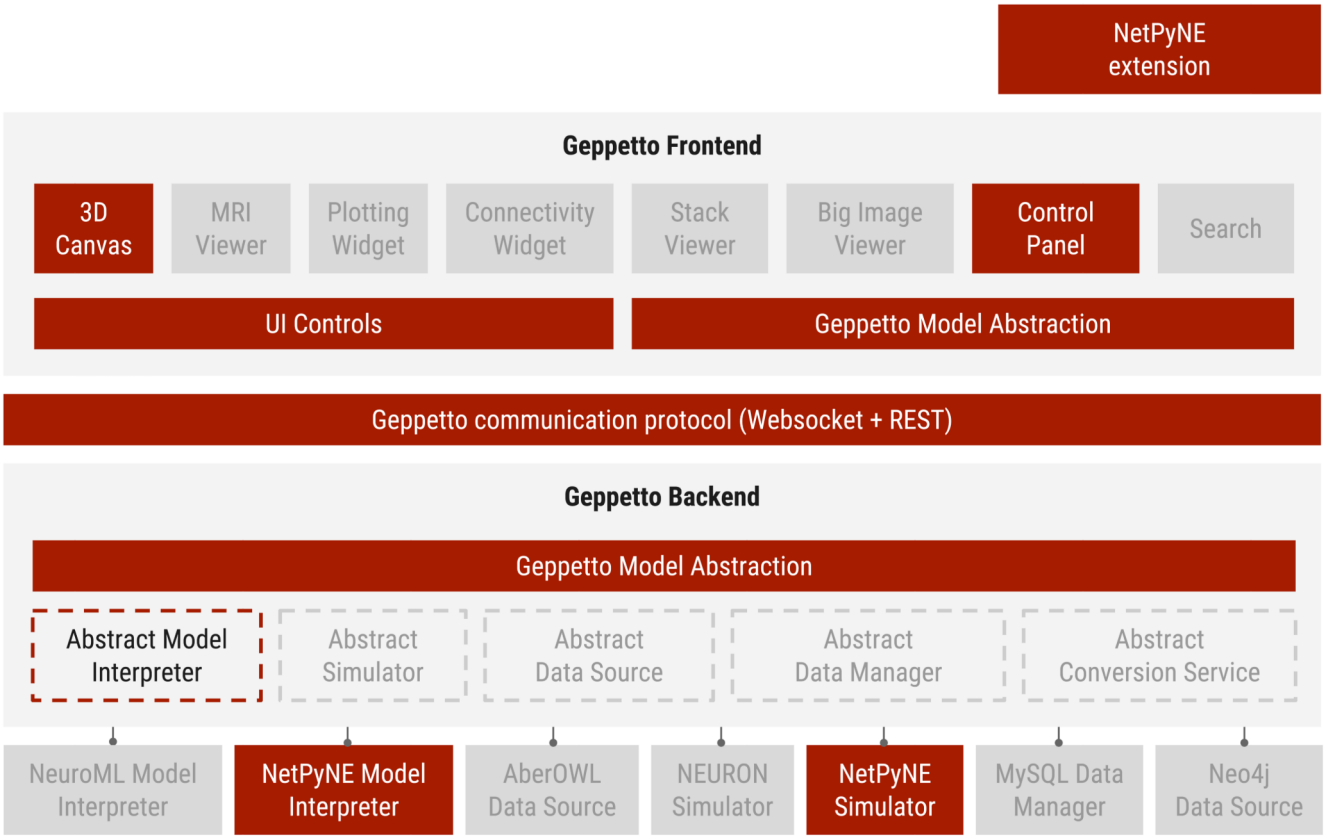

Figure 6. Geppetto NetPyNE-UI configuration.
Graphical representation of the components of Geppetto that are used on the NetPyNE-UI application (red). The ones not used are coloured in grey.

Europe PMC Funders Author Manuscripts

Europe PMC Funders Author Manuscripts

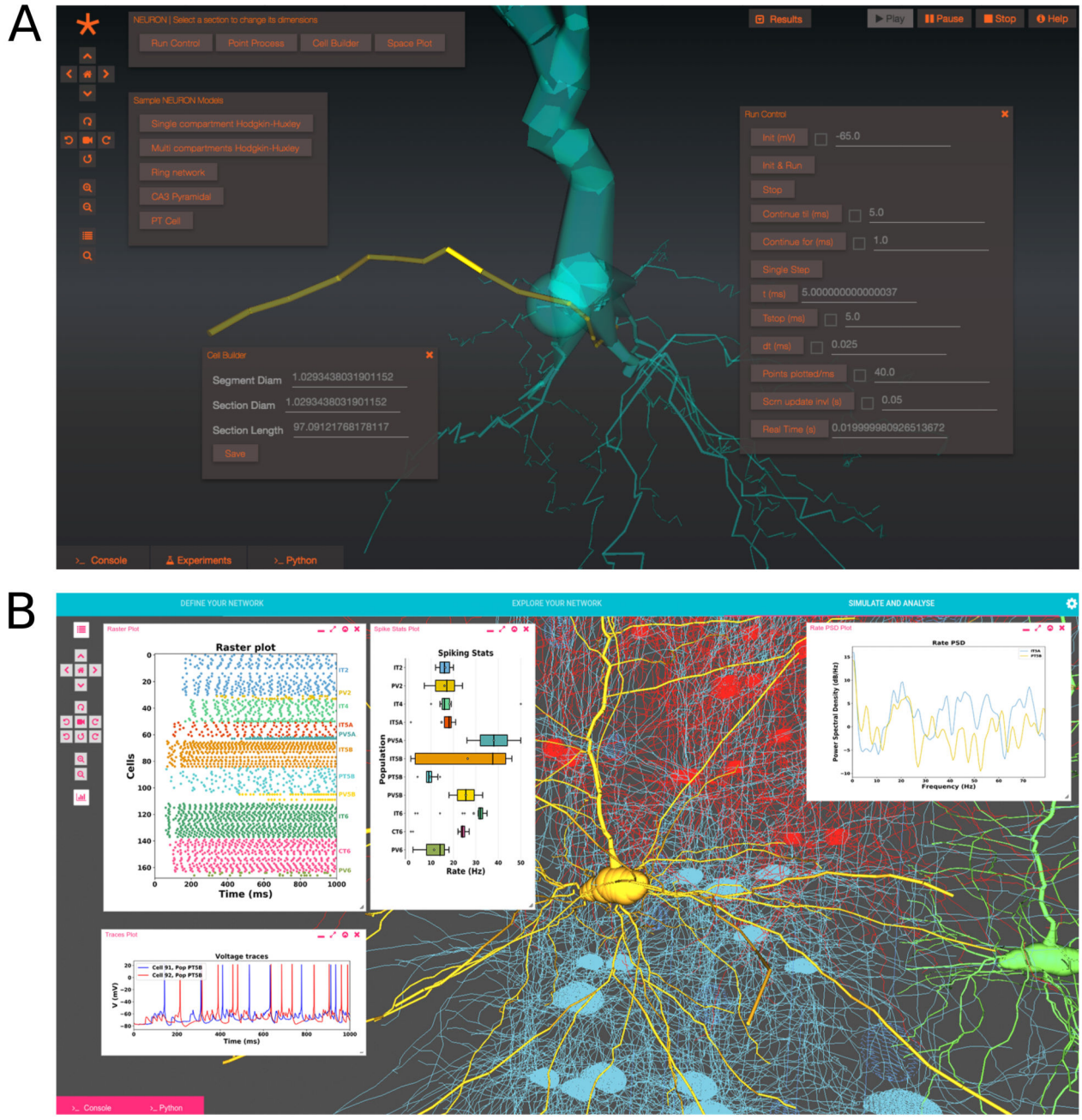

Figure 7.

(a) Screenshot of NEURON-UI while in edit mode, a simplified cell builder (bottom left) lets the user edit any selected section (in yellow) while the Run control panel (right) is used to control the simulation **(b)** NetPyNE-UI showing the result of a simulation of a large-scale M1 microcircuit model with widgets showing a rasterplot (top left), individual cell membrane potentials (bottom left), population spiking statistics (middle) and the power spectral densities for two populations (right).

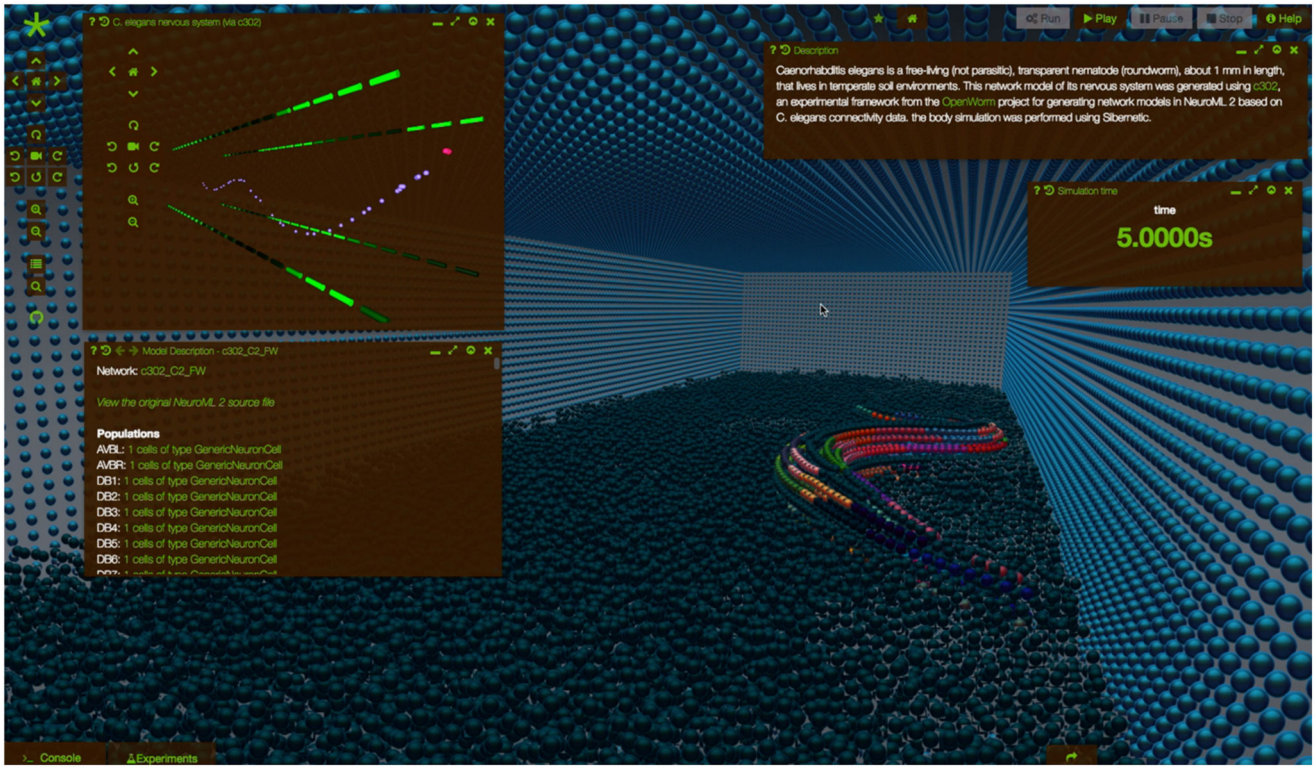

Figure 8.

Prototype of the integration between a nervous system model (top left widget) and a fluid mechanics based simulation of a worm body (background *3D Canvas*) within Geppetto, currently under development. The mechanical model of the body of the worm, which includes musculature, is shown immersed in a simulated fluid environment. Both worm body and fluid are made of particles. Different colors on the worm body highlight different groups of particles (e.g. elastic particles for each of the worm muscles, liquid particles for the surrounding fluid, etc.). All around the fluid and the worm is the experiment bounding box, made of an impermeable layer of particles. The calcium concentrations in the muscles (four rows separated from the main body cells in top left widget) simulated by the model are translated into activation signals for the muscles cells in Sibernetic ultimately driving the locomotion of the worm.

Europe PMC Funders Author Manuscripts