EBRAINS closed-loop AI and robotics workflows: Status at M18 SGA3 - D5.6

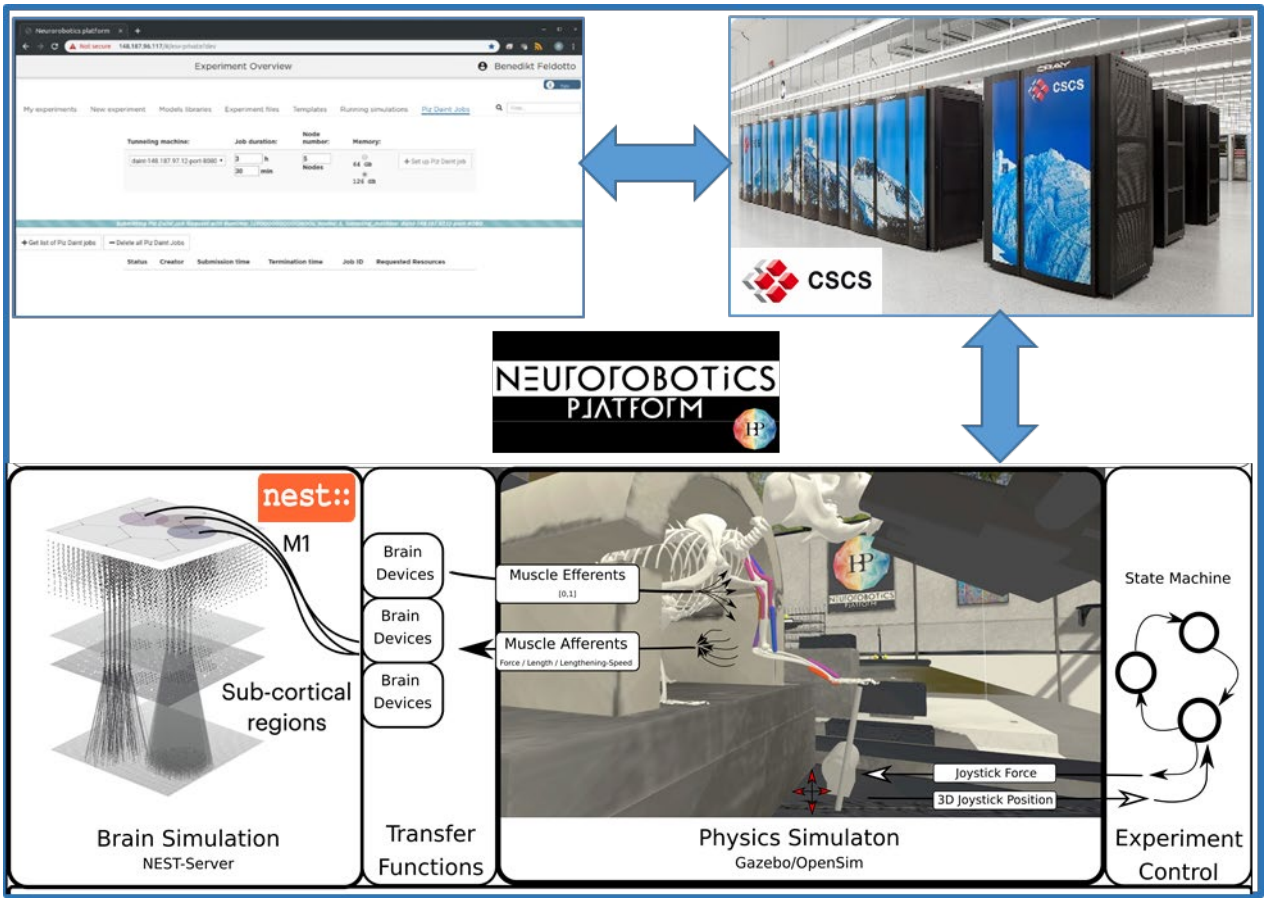

Figure 1: The Neurorobotics Platform enables embodied simulations with large networks of spiking neurons

The joint efforts of the NRP, NEST and CSCS engineering teams have borne fruit. As described herein, a service prototype was developed to enable neuroscientists to link brain models, neuron activity and motor output, even when dealing with networks comprising millions of simulated neurons and billions of synapses.

Project Number:	945539	Project Title:	HBP SGA3

Document Title:	EBRAINS Closed-loop AI and robotics workflows: Status at M18
Document Filename:	D5.6 (D53) SGA3 M18 ACCEPTED 220520.docx
Deliverable Number:	SGA3 D5.6 (D53)
Deliverable Type:	Other: Software
Dissemination Level:	PU = Public
Planned Delivery Date:	SGA3 M18 / 30 Sep 2021
Actual Delivery Date:	SGA3 M18 / 27 Sep 2021; accepted 20 May 2022
Author(s):	Fabrice MORIN, TUM (P56)
Compiled by:	Fabrice MORIN, TUM (P56)
Contributor(s):	Eloy RETAMINO, UGR (P66), contributed to Section 3.1.2 Sandro WEBER, TUM (P56), contributed to Section 3.2 Benedikt FELDOTTO, TUM (P56), contributed to Section 4
WP QC Review:	Victoria NEUMANN, TUM (P56)
WP Leader / Deputy Leader Sign Off:	Fabrice MORIN, TUM (P56)
PCO QC Review:	Martin TELEFONT, Annemieke MICHELS, EBRAINS (P1)
Description in GA:	Neurorobotics Platform release with improved functionality, extended content, and updated inventory of related closed-loop models, tools, and services prepared for, or released through the EBRAINS portal.
Abstract:	This document provides a high-level overview of the evolution of the Neurorobotics Platform (NRP) in SGA3. The latest release (version 3.2) is described, which provides our users with necessary updates in parts of the software stack in terms of both usability and functionality. Barring some unexpected need for major security updates to some components of the software stack, version 3.2 will be the last release of the “legacy” NRP. Indeed, the efforts of the NRP software development team will from now on be entirely focused on delivering the NRP v4.0; the latter represents a fundamental evolution in terms of software architecture and capabilities. In particular, this future version will be fully modular in terms of the simulators that can be orchestrated and integrated into experiments. These efforts – described herein and those planned – provide the foundation necessary to deliver a future-proof NRP that is seamlessly integrated into EBRAINS. Finally, large-scale NEST simulations on the NRP are described that leverage the capabilities for distributed computations of the EBRAINS infrastructure (specifically, the Piz Daint supercomputer). This achievement will underpin in the coming months the instantiation of a service unique to EBRAINS, which will provide neuroscientists with the ability to perform closed-loop simulations of bodies controlled by brains modelled with millions of spiking neurons and billions of synapses.

Keywords:	Neurorobotics, embodied cognition, simulation, closed-loop experiments, neuroscience, embodied AI, sensorimotor loop, spiking neural networks
Target Users/Readers:	Computational neuroscience community, computer scientists, Consortium members, embodied AI researchers, roboticists, general public, neuroscientific community, Platform users, students.

Table of Contents

Table of Tables

Table of Figures

1. Introduction

The present document provides a high-level view of the evolution of the Neurorobotics Platform (NRP) since January 2021. It follows and complements the previously-released Deliverable D5.1 (D48), which focused on the period going from April 2020 to December 2020. The latter document already introduced the notion that the NRP needed to evolve to remain relevant for the foreseeable future. In support of this vision, technical developments were undertaken – and are still ongoing – to change the architecture of the NRP and re-engineer some of its key components. This entails the production of a fundamentally new NRP, which leverages and extends the same concept that drove the development of the NRP since the beginning of the Human Brain Project, but does away with significant parts of the code base of the NRP v3.x (which we will refer to hereafter as the "legacy" NRP).

While the preparation of the NRP v4.0 understandably commanded a significant portion of the resources dedicated to the software development of the NRP, current NRP users were not left to their own devices. Support activities continued uninterrupted to implement various experiments, demonstrators and showcases based on the legacy version of the NRP. Furthermore, improvements were introduced through release 3.2 of the NRP to remove some critical limitations identified by our users and thus improve their experience with the platform.

The present Deliverable thus delves into both these aspects of our software development activities, with an emphasis on what they entail for the users of the NRP. Concretely, it focuses on the new features brought about by the last version of the "legacy" NRP (Section 2), the re-engineering of the software architecture of the NRP for the release v4.0 (Section 3), and finally a proto-service (Section 4): the deployment of large-scale NEST simulations on the NRP that leverage the capabilities for distributed computations of the EBRAINS infrastructure (specifically, the Piz Daint supercomputer in Lugano, Switzerland).

2. The last legacy release: NRP v3.2

Release 3.2 of the NRP is intended to be the final legacy version. As such, the focus of the NRP development team was on software reliability and usability rather than on new features. A lot of work thus went into bug fixing and feature consolidation; this is clearly reflected in the evolution of the NRP software stack in this latest release (see Table 1).

Table 1: Comparison of the main components of the software stack for NRP releases since April 2020

Release 3.0	Release 3.1	Release 3.2
Python 2.7	Python 3.8	Python 3.8
Ubuntu 18.04	Ubuntu 20.04	Ubuntu 20.04
ROS Melodic	ROS Noetic	ROS Noetic
Gazebo 9	Gazebo 11	Gazebo 11.3
NEST 2.12	NEST 2.18	NEST 2.18
PyNN 0.9.4	PyNN 0.9.5	PyNN 0.9.5

The upgrade from Gazebo 11 to Gazebo 11.3 is more important than it may appear. Indeed, Gazebo 11.3 introduces the "-lockstep" run option that, in theory, guarantees the execution of all plugins at their proper specified frequency. Previous Gazebo versions indeed worked on a best-effort basis, resulting in highly variable output rates for every plugin, sometimes way below their target (i.e. configuration-specified) values. This was a major pain point for our users, e.g. those working with simulated sensors requiring high update rates, such as DVS cameras. Using the NRP v3.2 and Gazebo 11.3 will now enable them to get a consistent, predetermined number of samples from their sensors for each timestep of the physics engine.

Among the new features introduced by v3.2, two of them are notable as they were introduced as a result of user requests. First, the Docker-based local installation of the NRP can now use Jupyter notebooks to run the Virtual Coach interactively. Up to v3.1, this was only possible through source

install, as the ports of the NRP Docker container were not adequately mapped. This is now fixed and, with v3.2, Jupyter notebooks can be run as easily from a source install as from a Docker container.

Additionally, NRP users can now run the Intel Loihi neuromorphic chip (configured with Nengo scripts) as a backend for simulating spiking neural networks on the NRP. This provides NRP users with the option to work with one of the most popular neuromorphic chips available.

The source code of the NRP is open source and available on the following Bitbucket repository: https://bitbucket.org/hbpneurorobotics/neurorobotics-platform/src/master.

The Docker installer for local install is available at https://neurorobotics.net/local_install.html.

The NRP documentation is available at https://neurorobotics.net/Documentation/nrp/index.html.

3. NRP v4.0: a new software architecture for a new philosophy

The software development of the NRP started in 2013, at a time when concepts such as process isolation and containerisation had not yet become widespread (the Docker Engine itself only appeared in 2013). It also followed a co-design path with the HBP, meaning that the technical specifications of the NRP emerged only progressively and evolved across the project lifetime. In the early years, a fairly monolithic architecture was therefore chosen, consisting in a synchronisation component (the so called closed-loop engine, or CLE) orchestrating communications between two simulators tightly integrated into the NRP code: Gazebo and NEST. Another foundational choice was the use of ROS as a communication layer between multiple components of the software architecture. On this basis, multiple features were added to the legacy NRP over the course of the HBP, with the final iteration being described in section 2.

This process, in spite of the many useful features that it introduced into the Platform, revealed the limits of the software architecture initially chosen. For example, the latter is poorly adapted to distribution over the nodes of a supercomputer for high-performance computing (HPC) applications. Similarly, many features were added at the cost of an NRP-specific fork of the open source software making up the NRP software stack. With the decision to build EBRAINS, a research infrastructure expected to outlive the HBP by at least a decade, the static nature of the legacy architecture thus compromised both long-term performance and sustainability of the NRP insofar as continuously evolving domain, user and technology requirements were bound to erode the usefulness of the Platform. Over time, this would eventually require its complete redesign.

3.1 Modularity and advantages thereof

To remediate this situation, it was decided to proactively proceed with this complete redesign, with a view to make the NRP intrinsically capable of evolution. This entailed refactoring the legacy CLE into a set of new components, referred to as NRP-core, fulfilling the same functional role (orchestration of multiple simulators) but with a key difference: NRP-core was indeed designed to make the NRP modular in terms of the simulators and control modules that can be integrated into a single simulation. NRP-core is thus the hub element in a spoke-hub architecture. The communication interfaces between simulators/modules and NRP-core are custom-made, although most of them can be derived from generic classes that are provided together with the rest of the NRP-core code. The combination of one given simulator / module and its communication interface to NRP-core is referred to as an “engine”.

NRP-core keeps the essence of the transfer function (TF) framework offered by the legacy NRP[1]. This allows users to decouple the computational features of individual modules from the definition of the connections that exist between one another. Because these functions are implemented in Python, the new NRP retains the ability to pause a simulation, modify any TF, and restart the simulation with the modification being immediately accounted for. This usability feature from the legacy versions of the NRP is therefore fully preserved.

NRP-core also implements a simulation manager, NRPCoreSim, which itself can run in two different modes: one is for timestep-driven simulations (see section 3.1.1), the other for event-driven simulations (see Section 3.1.2). These two modes implement different mechanisms for orchestration and communications between simulators, each implemented in a different software component (FixedTimeIncrementLoop – or FTILoop for brevity – and EventLoop, respectively). Finally, NRPCoreSim can run as a stand-alone application, or be controlled through a new component (NRP-server) connected to user interfaces (Frontend Graphical User Interface or Virtual Coach).

For NRP users, the main advantages of such an approach are as follows:

1) The NRP-core can orchestrate any number of time-step-based engines (simulators or control modules) running concurrently, this number being only limited by the availability of compute resources. This is especially interesting for users that create modular cognitive architectures.
2) New simulators can easily be added, thus providing the NRP with the capability to keep up to date with technological evolution and providing users with options to incorporate their own tools through a well-documented application programming interface (API).
3) The communication protocols between NRP-core and engines are not prescribed: an API is provided to extend the number of supported protocols. Furthermore, these communication protocols can be mixed. For example, it is easy to run one engine communicating with NRP-core through gRPC with protobufs, together with another engine communicating through JSON over REST. NRP users who need specific simulators that are not supported out-of-the-box can thus easily integrate the former into the NRP ecosystem as new engines.
4) The maintainability of the NRP code base is considerably increased, as no fork of any simulator is required; in the worst-case scenario, updates to the client-server interfaces can be necessary, but by and large, updates to individual simulators can be leveraged by NRP users without significant resource investment.

Refactoring the NRP software architecture is a massive undertaking, especially considering the already significant maturity level of the latest legacy version. However, it had to be done to provide users with an NRP ecosystem that is more open, more versatile, more HPC-ready and more forward-looking.

3.1.1 Timestep-driven simulations

NRP-core running with FTILoop is designed specifically for supporting simulations that progress through time in well-defined increments (time steps). Unlike event-driven simulations, such timestep-driven ones can achieve full reproducibility, and thus hold a special place in the toolbox of computational (neuro)scientists. The FTILoop establishes synchronous communications with engines (the spokes in the aforementioned hub-spoke architecture) through a client-server paradigm, and ensures that new data is shared across modules at exactly the right simulation timestep. Figure 2: summarises this new architecture for the NRP with timestep-driven simulators that have already been integrated as engines (e.g. Unity, OpenSim) or for which integration is in the works (e.g. The Virtual Brain).

For NRP users interested in fully deterministic time-driven simulations, the main advantages of such an approach are manifold:

[1] These functions were renamed as “Transceiver Functions” (also abbreviated as TFs) in the new NRP, to emphasise the conceptual proximity as well as the small differences in syntax between the two frameworks, and to remove any ambiguity in the documentation.

1) The removal of ROS as a communication layer, as well as its replacement by fully synchronous client-server communications, removes a major source of non-determinism that was so far intrinsic in every NRP simulation, thereby increasing the usefulness of the platform as a scientific tool.
2) The NRP can more easily be distributed over a given HPC architecture by adapting the containerising configuration of NRP-core and simulators to the requirements of such distribution. This is essential for the development of some of the NRP-based EBRAINS services (see for example Section 4).
3) The ability to run NRP-core with FTILoop as a standalone application (i.e. headless execution of NRP experiments) considerably increases code reusability. It also enables the use of NRP-core-based simulations into learning frameworks (e.g. the L2L framework developed within HBP) by simplifying containerisation of experiments.

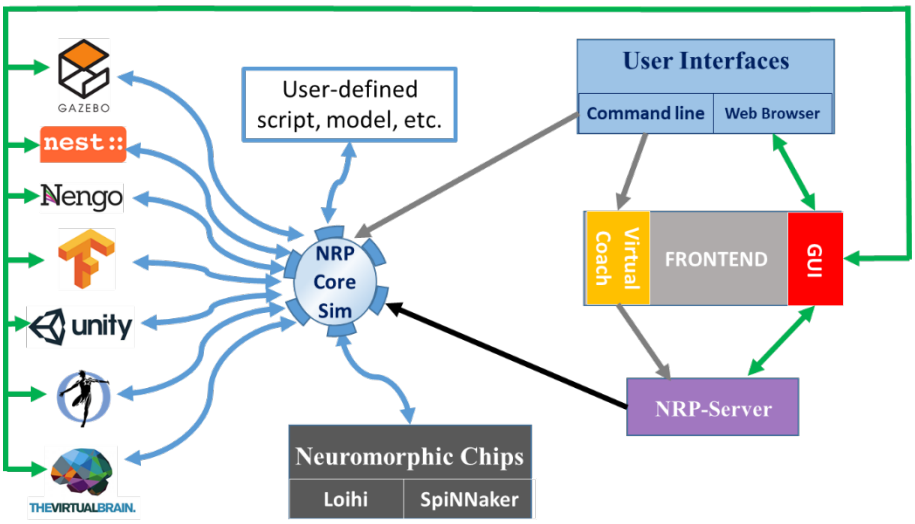

Figure 2: Components of the upcoming NRP v4.0 and their relation to timestep-driven simulators

All the code related to NRP-core and the FTILoop can be found here:
https://bitbucket.org/hbpneurorobotics/nrp-core

3.1.2 Event-driven simulations

NRP-core with FTILoop provides determinism and reproducibility, but it is not adequate for event-based communications and processes. In particular, it is not suitable for the inclusion of interactions with cyber-physical systems, such as hardware in the loop, or virtual reality tools supporting real-time man-machine interfaces with the simulations. As such, the capabilities provided by ROS-based legacy versions of the NRP cannot be carried over to the new NRP by FTILoop alone. For use cases requiring such capabilities, we thus decided to create EventLoop, an orchestration component that can replace FTILoop in NRP-core for any simulation requiring event-driven capabilities.

The main characteristics of EventLoop are asynchronous communications and loose interactions between NRPCoreSim and Engines. This required a series of components to be specifically developed in NRP-core: the input and output nodes. Indeed, NRPCoreSim operating synchronously with FTILoop uses a class of objects called EngineClients to interact with Engines, for both control and data exchange. The most significant feature of these EngineClients is that they are written in a way that blocks execution of the NRP simulation until data is properly produced or processed. In the asynchronous mode, they are therefore substituted by input and output nodes that have a much looser connection to NRPCoreSim (i.e. are non-blocking). They implement the connection of inputs and outputs of TFs to Engines via asynchronous data channels. EventLoop thus runs at a fixed frequency (in actual time, *not* in simulation time) and implements the following steps:

- processing new messages received through Input nodes and rely them to connected TFs.

- execution of TFs.
- sending output of TFs through the corresponding output nodes to apposite engines.

The TFs must also be adapted to the execution mode of NRPCoreSim. As they are Python functions, the connections between TFs and Input and Output nodes can be specified declaratively by users with the use of dedicated decorators in the definition of TFs. The syntax, in this case, is very similar to that used to connect TFs to Engines in the synchronous case, so as to ease the process of porting experiments in NRP-core from time-step-driven to event-driven operation modes. This path should be very attractive to scientists in the neuro-robotics domain, which will be able to safely train modular neuro-controllers in a simulated environment, and afterwards port them to real robots.

Finally, NRPCoreSim with EventLoop can run under real-time constraints (as defined here: https://en.wikipedia.org/wiki/Real-time_computing#Criteria_for_real-time_computing), provided that certain requirements are met. These requirements are summarised below:

- All TFs in the experiment must be implemented in C/C++. The possibility to use precompiled C/C++ functions as TFs (instead of Python) has been added to NRP-core to support use-cases where performance is necessary. This is indeed the case if a user wants to execute NRPCoreSim under real-time constraints.
- In the implementation of TFs, the user must avoid operations that prevent a deterministic schedule in the execution of the loop. A guide for best practices in implementing real-time code can be found here: https://design.ros2.org/articles/realtime_background.html.
- The OS and transport layer used in the experiment must be real-time ready

In future iterations of NRP-core, the addition of monitoring tools for assessing real-time performance and violations of real-time constraints is planned so that users can be confident in the performance observed with EventLoop.

All the code related to the EventLoop currently resides in the following branch of NRP-core: https://bitbucket.org/hbpneurorobotics/nrp-core/branch/event-loop

3.2 New NRP frontend

The NRP frontend of the legacy version of the NRP is based on AngularJS – a very popular framework for web applications back when the development of the NRP was initiated within the HBP. Released in 2010 by Google, AngularJS, however, is meeting its end of life: by the end of 2021, it will not be further supported. New frameworks like Angular, VueJS and React have since long replaced it for developing progressive web applications. To provide a future-proof platform and address concerns of familiarity for developers, as well as browser support, updates and security, it is therefore mandatory to make a switch to more modern technologies.

Additionally, the original design of the legacy web frontend was also heavily geared towards the initial conception of the NRP with Gazebo and NEST as its main simulators. With the change of philosophy embodied through the redesign of the NRP in version 4, now is thus an excellent point in time to adjust the frontend accordingly. As the core of the NRP moves away from a specific bundle of simulators towards a more modular infrastructure, so the web frontend will move towards a collection of tools and services communicating with the NRP-core that are designed to be used and extended in the future by community developers. This will allow users of the NRP to quickly adopt existing web interfaces (see Figure 3) and libraries designed by the individual simulators' own communities or write their own tools if necessary.

If the Platform is supposed to be extendable in the future by the community, then the technology used must enticing, or at the very least familiar, to the potential users concerned. As such, having to learn the inner workings of an outdated technology heavily discourages any effort. For this reason, React (https://reactjs.org/) was chosen as the user interface (UI) framework; indeed, it currently enjoys a high popularity while also being one of the most future-proof options, as it is supported by Facebook. Most importantly, it is a lightweight and versatile framework that focuses on reusable components; different parts of the web frontend can therefore remain self-contained and flexible,

thus avoiding a monolithic and closely intertwined architecture that would be hard to change in the future. The services used to communicate with, and steer execution of, simulations through NRP-core are rewritten to be: a) pure Javascript without any ties to other frameworks and b) obvious in function and location in the code so that, later, community developers can easily find what they need to make new tools run in conjunction with the NRP. Finally, further integration with EBRAINS tools services, present and future, will be made considerably easier through this framework (e.g. integration of NEST-desktop depicted in Figure 3).

All the code related to the new frontend for the upcoming NRP v4.0 will be merged with the NRP master repositories upon release. Until then, it can be found here for evaluation: https://bitbucket.org/hbpneurorobotics/nrp-frontend/src/development/

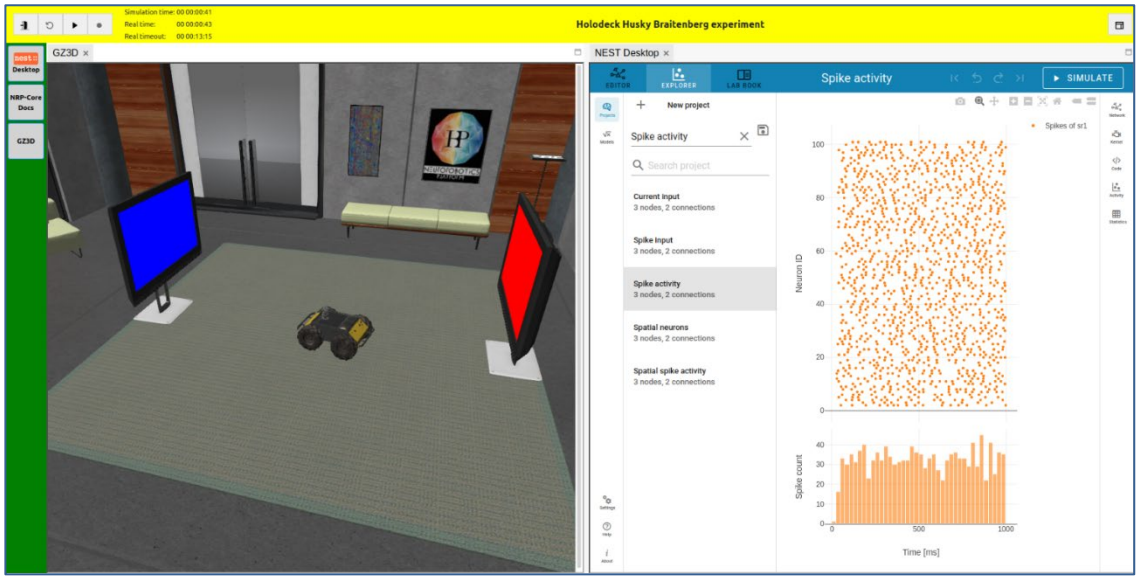

Figure 3: GZ3D environment and spike raster plot from NEST-desktop in the new NRP frontend.

The latter will retain the look and feel of the legacy frontend but will fundamentally differ in that any existing web interface can be integrated in a matter of minutes, provided it is ready to be deployed on its own. For example, the plotting capabilities of NEST-desktop featured herein cannot be leveraged by the legacy frontend.

4. Large-scale NEST simulations on the NRP: towards a unique EBRAINS service

A prototype EBRAINS service was created to enable users to run custom-embodied large-scale brain simulations inside the NRP. This service is intended to run large-scale, multi-area networks of spiking neurons simulated in NEST and connected through the NRP with the physically realistic simulation of a musculoskeletal system in Gazebo. This allows neuroscientists to explore *in silico* the complex relationships that exist between network topology, neuronal electrophysiology and expressed behaviour, without the size of the simulated network being a limiting factor (see also section 6.2). This prototype was made possible by using the client-server paradigm described in section 3, as well as the distribution of NEST (as a server) over multiple nodes of the supercomputer Piz Daint (CSCS, Lugano, Switzerland).

As a demonstrator, we implemented an experiment that features a multi-region rodent brain model (1,089,137 spiking neurons interconnected through 1,588,469,795 Synapses in NEST, developed by multiple Japanese teams[2]) controlling the movements of a musculoskeletal system (a rodent forelimb) with 8 muscles, in conditioning tasks that involve pressing a lever (also modelled inside the physical simulation). While the brain-body connection is currently simplistic (activation of

[2] Joint publication in preparation.

neuronal populations in the motor cortex directly causes muscle contraction), it already provides a complete view of what the service will look like for EBRAINS users.

Access to the service prototype is given through the NRP frontend. On a new bespoke user interface (Figure 4, left), the user can request supercomputing resources on Piz Daint, launch new NRP instances, and manage running jobs. Once a simulation has been launched, the user can control and interact with the experiment through the same graphical interface as in a local installation (Figure 4, right), but with the compute power of a supercomputer actually running NEST.

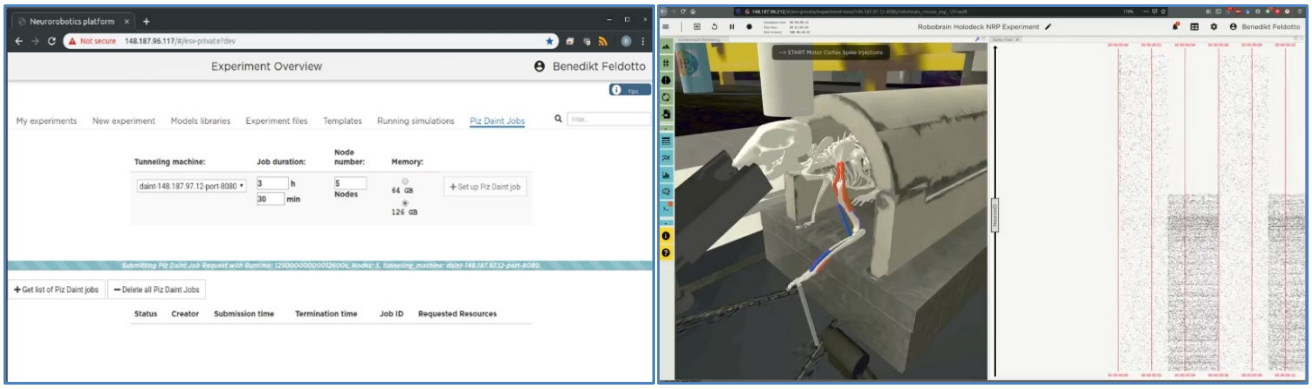

Figure 4: User interface for resource allocation and experiment view during simulation

For the more technically inclined readers, the following paragraph details the software architecture for the proposed service. This architecture spans across Virtual Machines (VMs) on Pollux (a cluster of VMs at CSCS, Lugano, Switzerland) and Piz Daint nodes. The NRP frontend runs on a Pollux VM and routes the connections from the user's browser to the instantiated NRP backend on a Piz Daint node via a dedicated tunnelling machine, itself set up on a different Pollux VM (see Figure 5). A UNICORE (Uniform Interface to Computing Resources) interface was implemented inside the NRP proxy that sends requests of user resources and NRP control scripts to Piz Daint, where SLURM (the job scheduler) manages node allocation. The first requested node runs the NRP backend, while all the others run a distributed NEST simulation, made possible by the new implementation of a NEST server in NEST 3.0. A corresponding NEST client was created as a device inside the NRP, which connects with the NEST server instances on the various compute nodes. Finally, MPI distribution is leveraged to balance brain simulation traffic of NEST across multiple nodes.

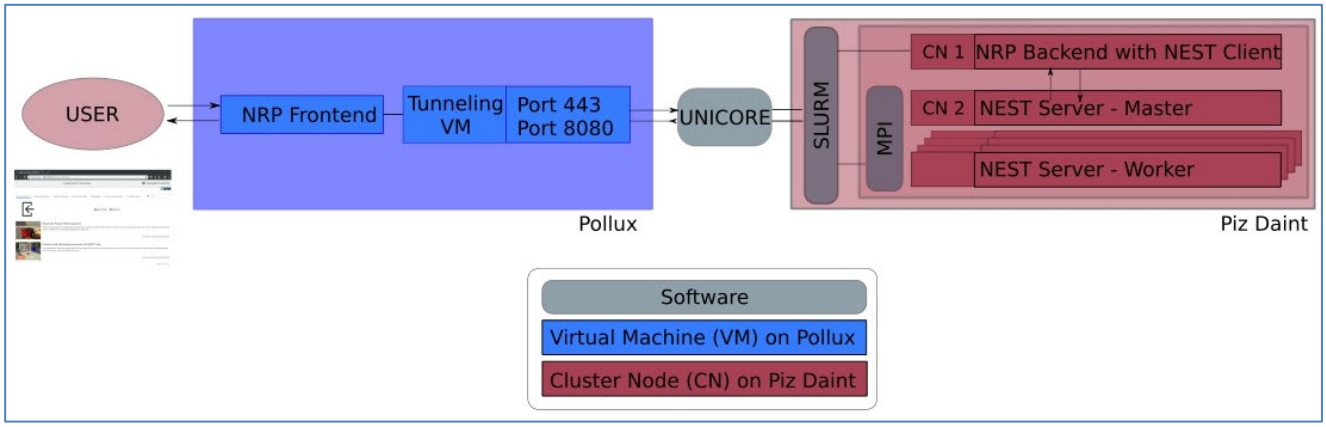

Figure 5: Architecture of the service prototype

This service prototype is not available to the general public through EBRAINS yet, as it still needs to be connected to a service account and to the EBRAINS federated user resource management system. However, it is a prime example of collaboration between multiple engineering teams (NRP, NEST, infrastructure) to eventually deliver unique added value to users through EBRAINS. To the best of our knowledge, there is no equivalent to this service anywhere in the world, and we are therefore looking forward to its official introduction as an EBRAINS offering.

The code corresponding to this work will be merged to the NRP repositories upon release of the service itself. Until then, it is available for evaluation on dedicated branches:

- For the adapted NEST client: https://bitbucket.org/hbpneurorobotics/cle/src/NRRPLT-7915-integrate-nest-client-server/
- For the new graphical user interface to Piz Daint in the frontend: https://bitbucket.org/hbpneurorobotics/exdfrontend/src/NRRPLT-8277_daint_from_nrp_frontend/
- For the new UNICORE interface to Piz Daint in the NRP proxy: https://bitbucket.org/hbpneurorobotics/nrpbackendproxy/src/NRRPLT-8277_daint_from_nrp_frontend/

5. Conclusion

The NRP evolves to better serve its users. Concretely, version 4.0 will propose a modular architecture supported by the ongoing refactoring of both its frontend and backend components. This will make the NRP future-proof and greatly enhances the maintainability of its codebase, which are essential characteristics in the perspective of the integration of the Platform into services of the EBRAINS infrastructure. Although v4.0 is still a few months away (see Annex), the client-server architecture that underpins the refactoring of the old CLE into the new NRP-core has already enabled a unique technical achievement. Large-scale NEST simulations on the NRP are indeed a unique value proposition of EBRAINS to neuroscientists that want to bridge the gap between brain structure, activity and behaviour.

Finally, while the present document delves into multiple topics that are of direct interest to the users of the NRP, it does not constitute an exhaustive list of the efforts that are underway. Indeed, many activities of the NRP software development team are as essential as they are transparent to NRP users, and they are not described in detail herein. These include e.g. the deployment of a proper workflow for continuous integration / continuous deployment (CI/CD), the ongoing migration of all Virtual Machines (VMs) from the Pollux cluster to Castor cluster (both at CSCS in Lugano; Pollux will be shut down at the end of September 2021), the refactoring of the authentication workflow of the online NRP to accommodate the new Keycloak technology adopted by EBRAINS, etc. Thus, besides the various activities reported hereinbefore, NRP users can rest assured that there is a lot going on behind the scenes to make the NRP (and, of course, the EBRAINS services based thereupon) as usable and robust as possible.

6. Annex: Roadmap of upcoming activities

The following is a tentative timeline of the software development activities of the NRP, provided for information purposes only. It does not contain information of relevance to the NRP users about the current functionalities of the NRP but is meant to provide more concrete information regarding what they can expect in the coming months and years.

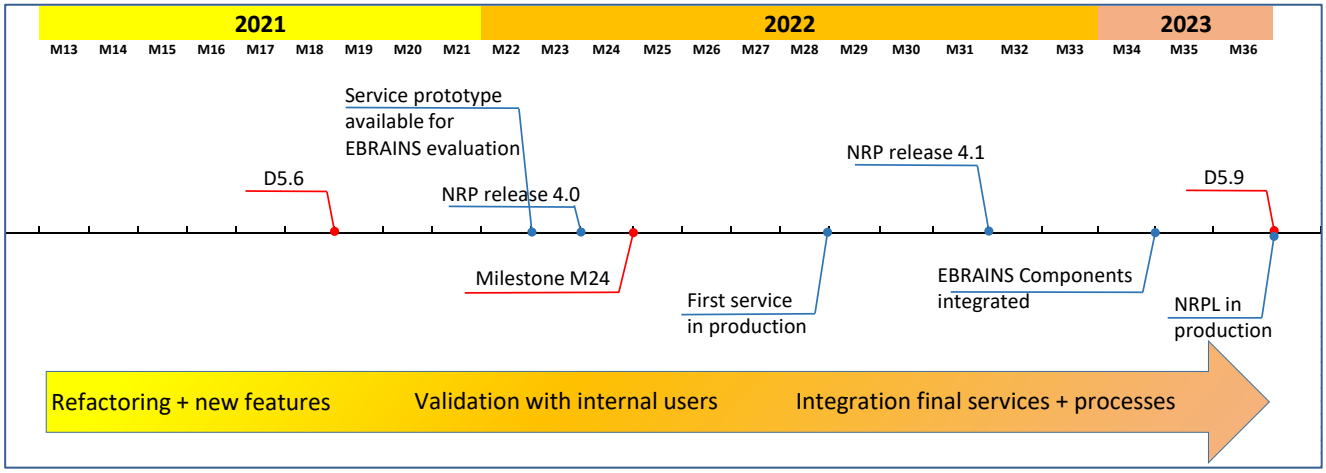

Figure 6: Tentative timeline of software development activities of the NRP

The numbered months are related to the timeline of the Human Brain Project. Blue lines indicate internal deadlines set by the software development team of the NRP, red lines indicate Deliverables and Milestones included in the Description of Action of the Human Brain Project of relevance to the NRP users. Details for each element are provided in section 6.1.

6.1 Significant activities and deliveries

- M22: One service prototype (large-scale NRP-NEST simulations; see Section 4) available for evaluation by EBRAINS leadership, with all frontend-related features and infrastructure-related functions (e.g. authentication, user resource management through service account, etc.) required from a minimally viable product.
- M23: Release of NRP v4.0 (see Section 3).
- M24 - HBP Milestone: Provision of integrated dynamical closed-loop models developed within the HBP, useable on the NRP.
- M28: One NRP-based service moves into production (large-scale NRP-NEST simulations; see Section 6.2), supported by apposite GUIs, deployed on HPC infrastructure. The second NRP-based service (NR Public Library; see Section 6.2) is presented to EBRAINS leadership for evaluation.
- M31: Release of NRP v4.1, including new OLE and related demonstrators, as well as consolidated client-server interfaces between the CLE for an updated set (to be confirmed) of simulations engines (OpenSim and, e.g. PyBullet, TVB, etc.).
- M34: Complete integration into the NRP of all EBRAINS-level components:
  - Service accounts established and tested
  - Connection to EBRAINS data and compute proxies (and thus to the federated user resource management system) operational and tested.
  - Data provenance tracking operational and tested.
  - User roles properly inherited from EBRAINS Collab.
  - CI/CD process fully operational and in line with final EBRAINS specifications.
  - Repositories containing NRP code fully migrated in line with EBRAINS specifications.

- All quality control procedures specified by EBRAINS passed.

- M36: HBP Deliverable D5.9 (D56): Public report detailing the latest features available on the NRP and the full list of EBRAINS services based thereupon.
- M36: second NRP-based service (NRPL; see Section 6.2 below) in production, compatible with all existing NRP versions.

6.2 Description of the proposed EBRAINS services based on the NRP

The NRP itself can be considered both as a tool and a service (which has been online for many years). However, we intend to provide more value to our users by offering additional services based on the NRP with specific purposes. The first two of such new EBRAINS services will thus be (pending approval by EBRAINS management): 1) the NeuroRobotics Public Library (NRPL); and 2) embodied simulations of very large-scale spiking neural networks in NEST (based on the proof of concept described in section 4).

The NRPL is to be a service offered to both the Robotics and Neuroscience communities, intended to run on an Openstack park of Virtual Machines (e.g. the Castor cluster at CSCS). The NRP Public Library is a service where experiments can be shared by an initial “owner” / author and offered to NRP users through a graphical web interface. The NRP Public Library is intended to store the experiment files and (where necessary) actual datasets produced by the corresponding experiment in long-term archival storage. From their browser, the users should be able to browse these experiments, clone experiment files to their private space, and finally run and modify them. This service is intended to support lectures in the academic world and to offer both code and simulation engines as supporting information for scientific publications. While this functionality is closely related to currently available functions of the online NRP (e.g. cloning and sharing experiments), it needs to go beyond as there must be mechanisms for: 1) keeping the saved experiments and data safe (from, e.g. hard drive failure) by leveraging an EBRAINS solution for long-term safe data storage (e.g. SWIFT storage); 2) handling the evolution of the NRP itself (e.g. versioning and interactive deployment). Indeed, such a publishing service cannot be sustainable if the experiments available online are broken by updates to the NRP itself. This entails keeping a copy of every Docker image deployed online across time (e.g. in the EBRAINS Docker registry, Harbor) and also ensuring that the Openstack infrastructure (i.e. Virtual Machines) remains compatible with every one of those images.

Provision of embodied simulations of very large-scale spiking neural networks in NEST is a service intended for the Neuroscience and Neuromorphic Computing communities. Intended to run on the Piz Daint supercomputer at CSCS (at least until the end of the HBP) for the NRP backend and simulation engines, and with a virtual machine (e.g. on Castor at CSCS) running the NRP frontend, this service addresses the need of researchers studying the computational properties of brain models or robotic controllers built upon large-scale spiking neural networks. It will be unique in its ability to connect simulated bodies to large (millions of neurons, billions of synapses) brains, and let them interact in a closed loop within physically realistic environments.