Closed Loop Engine Architecture

State: Draft

Last Edited: 04.09.2014 by Georg Hinkel

1. Purpose

The Closed Loop Engine (CLE) is responsible for transferring data from the neuronal brain simulator to the world simulation engine (WSE). This includes the synchronization of the simulations and the coordination of the data transfer.

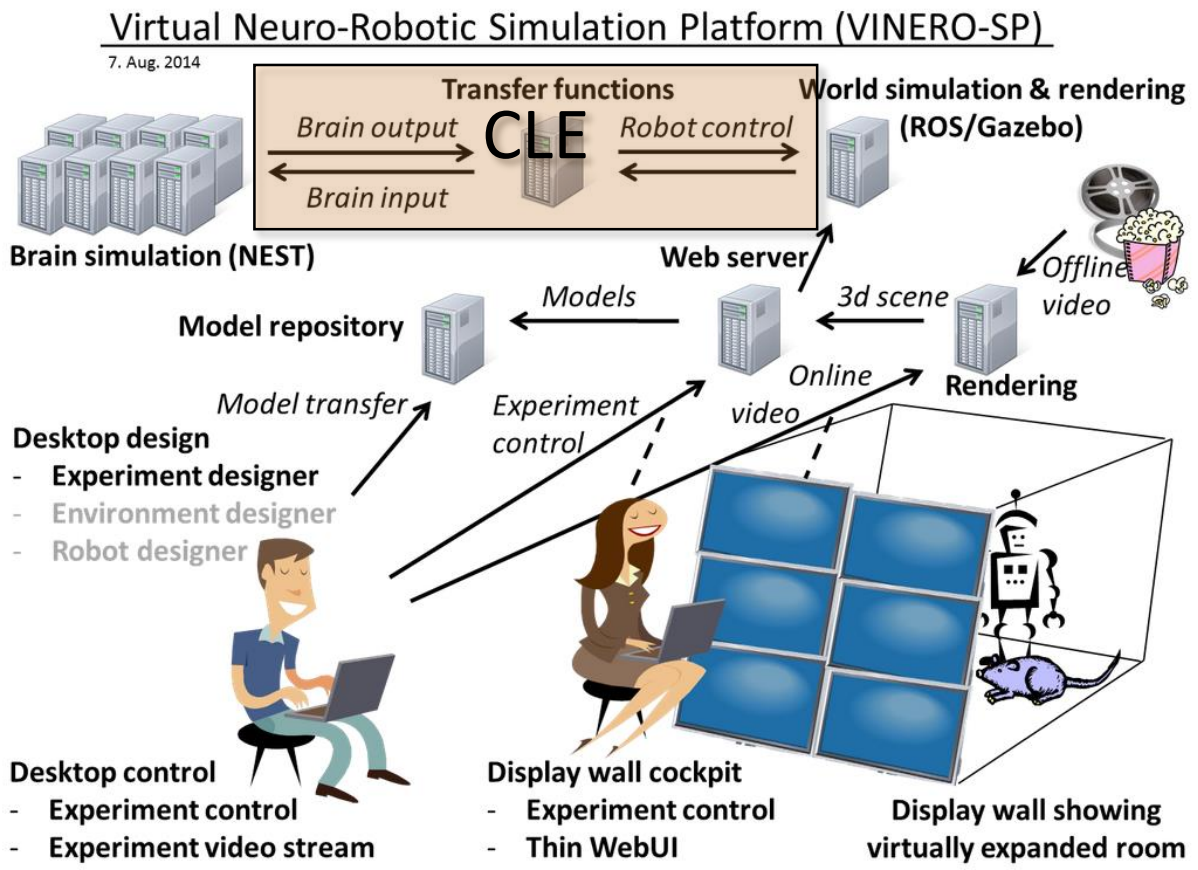

Figure 1: Overview of HBP SP10

Figure 1 shows the overview of the Neurobotics subproject SP10 of the Human Brain Project (HBP). The Closed Loop Engine (CLE) is the mediator between the world simulation and the brain simulation. The CLE is highlighted in the figure. The architecture of this component is the purpose of this document.

2. Functional Requirements

The goal of the CLE is to couple a brain simulation to a world simulation simulating a robot, such that the brain simulation can control the robots behavior. The transmission of the sensor data to the brain and in the other way from the neurons spikes to actual robot commands is done through a communication component, the transfer functions. These transfer functions are specified from the user of the platform, which are neuroscientists.

The management of transferring data in one direction, i.e. either from the world simulation (robot simulation) to the brain or vice versa is referred to as an open loop. The management of both directions is called the closed loop as it enables information to loop through the system.

The purpose of the CLE is now to control when which data is sent between the simulation and when which of the simulations is running.

3. Technical requirements

The world simulation currently runs on the middleware Robot Operating System (ROS), which has a platform dependency to Ubuntu Linux. Furthermore, we are using Nest as our brain simulation, which also has a platform dependency to Linux, although the simulator can be controlled remotely using PyNN, which is a Python interface.

The transfer functions have a strict requirement that users must be able to specify them in Python.

Overall, it should be possible to run the CLE in realtime mode, such that the simulated time does not differ too much from the real time. This can be possible by running simulations at a lower time resolution.

4. Architecture of the CLE

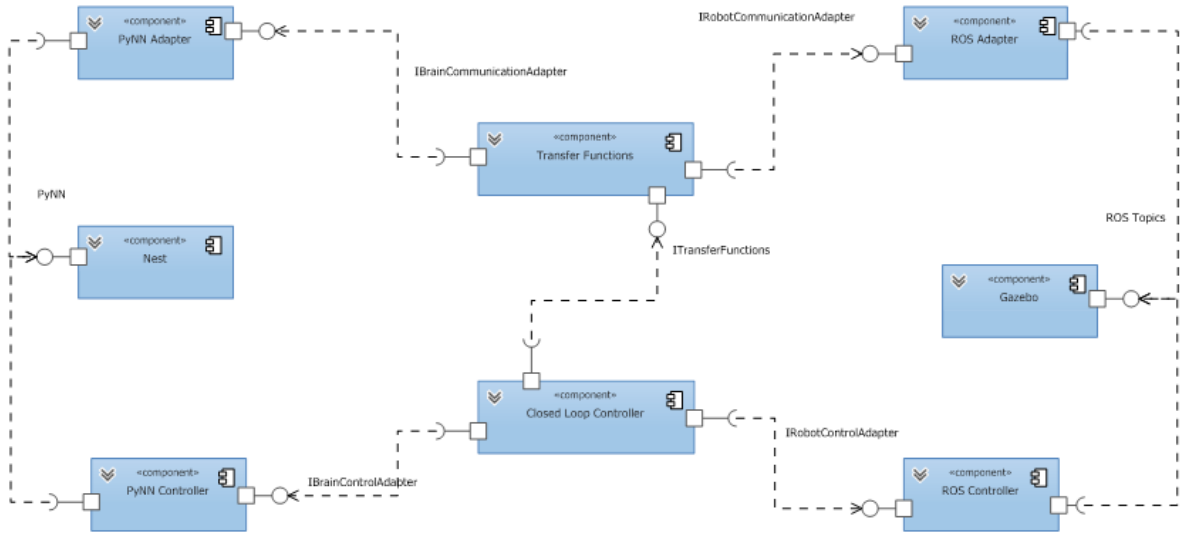

Figure 2: Architecture of the Closed Loop Engine

Figure 2 shows the architecture of the CLE in a UML component diagram. The data transfer is managed from the world simulation (currently Gazebo and unlikely to change) to the neuronal brain simulator (currently Nest, but we should also support other simulators such as SpiNNaker). The CLE basically consists of two lanes. The upper lane consisting of the ROS Adapter, the Transfer Functions and the PyNN Adapter are responsible for the data transfer, whereas the lower lane consisting of ROS Controller, Closed Loop Controller and PyNN Controller is responsible for the control management.

ROS Adapter and ROS Controller communicate with Gazebo through ROS topics, whereas PyNN Adapter and PyNN Controller communicate with Nest via the PyNN interface, which allows us to switch from Nest to other neuronal simulators such as Neuron or SpiNNaker. The interfaces between the components of the CLE are manifested in interfaces (see Figure 3). The communication between these components is currently meant as in-process communication as they should operate non-distributed. However, the architecture allows to distribute the system at a later stage by simply hiding e.g. the Transfer Functions behind a proxy that accesses a remote component through remote procedure calls.

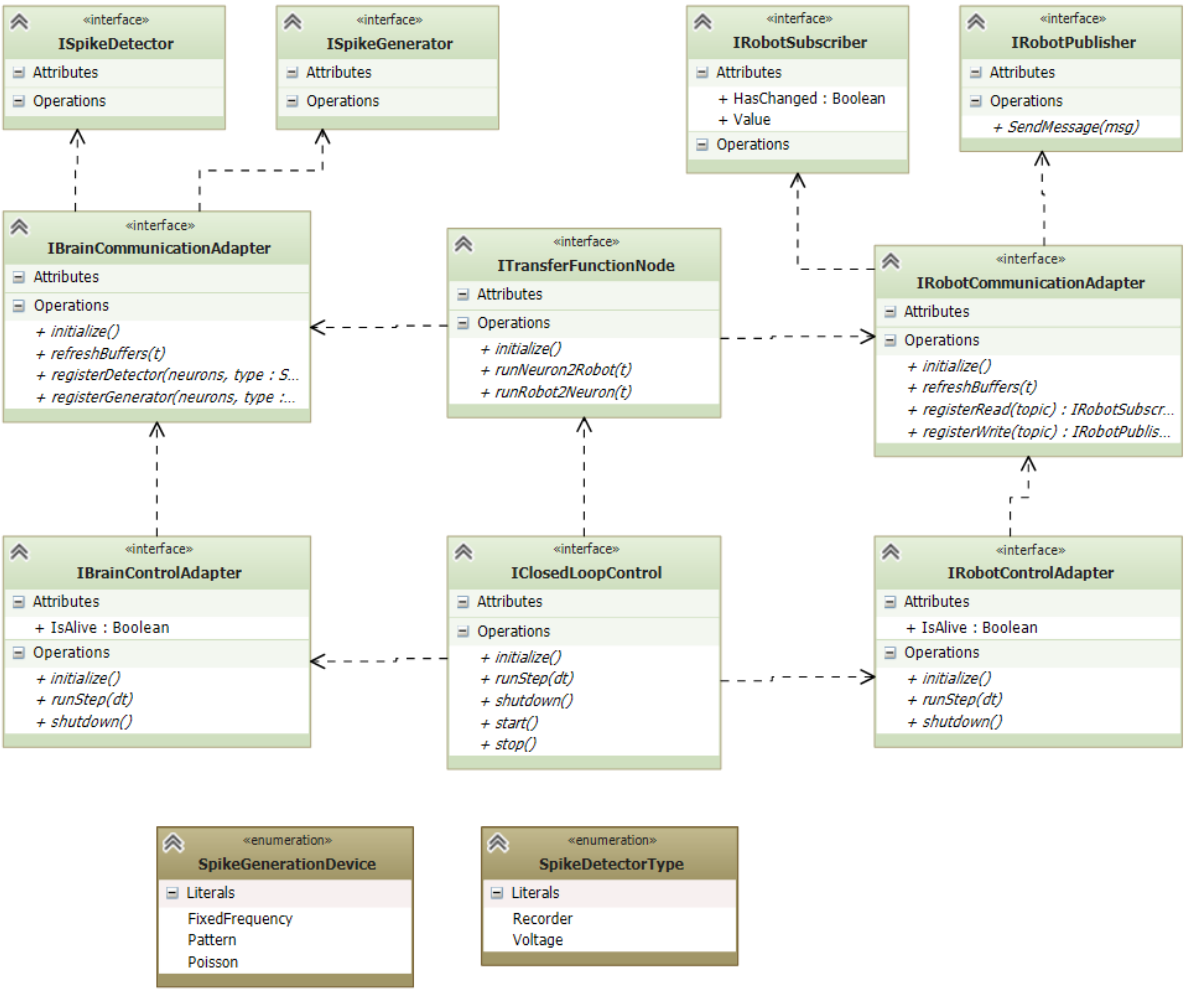

Figure 3: Interfaces of the Closed Loop Engine

The interfaces relevant for the above architecture of the CLE are shown in Figure 3. The transfer functions provide three methods that initialize them and call the transfer functions in either direction.

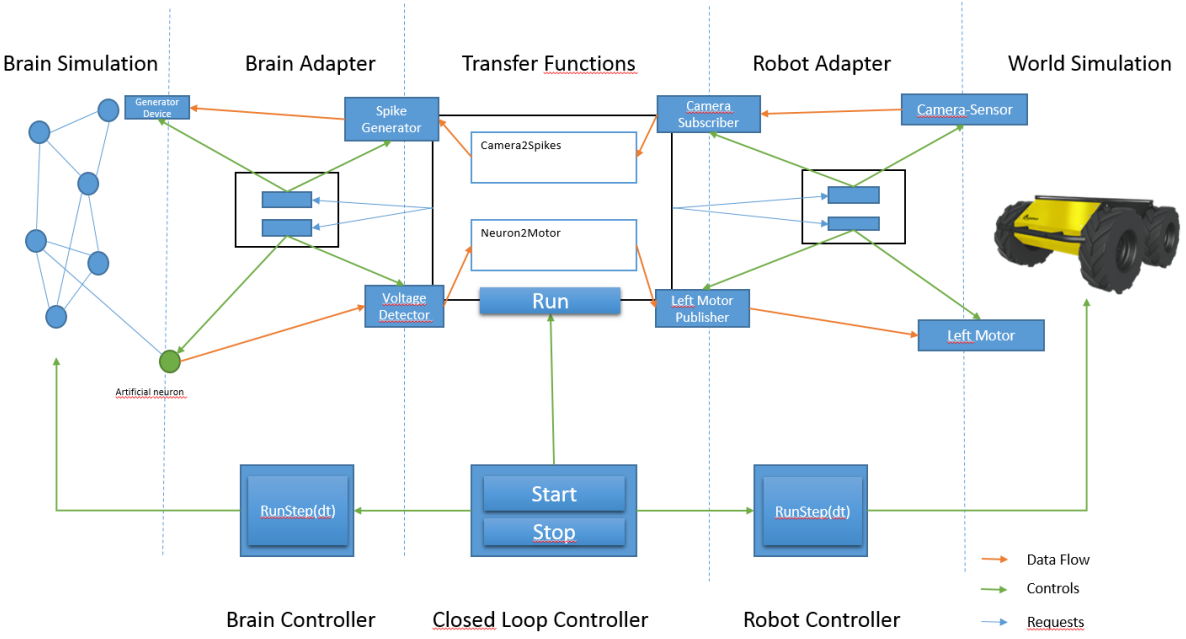

Figure 4: Interactions within the CLE

The interaction of the components within the CLE are shown in Figure 4. The data transfer is handled in the upper lane, whereas the control of the simulation is done in the lower lane. The dotted lines mark the component boundaries within the lanes. Objects drawn on the dotted lines are interfaces between the components.

The center of the upper lane is the Transfer Functions Component that organizes the data transfer. It consists of a set of transfer functions and knows which transfer functions require which communication objects to send or receive data and requests these communication objects from the respective adapters. These adapters then map these communication objects to artifacts in the simulation or instrument the simulation with new artifacts. In Figure 4, the Camera Sensor and the Left Motor exist and just need to be connected by the Robot Adapter, whereas the artificial neuron implementing the voltmeter requested by the transfer functions is injected to the brain simulation by the brain adapter. Similar, devices such as the spike generator in the upper left corner are also instrumented in the neuronal simulation by the brain adapter. The Transfer Functions connect these communication objects to the transfer functions that need them and executes them if needed.

The main work for the transfer function node is completed after the initialization, when all transfer functions are connected to their communication objects that they need. The communication during the simulation (indicated in Figure 4 with the orange arrows) is then only done between the communication adapters, which forward or receive data from the technical objects realizing their functionality.

The need for the Tranfer Functions Component to run its transfer functions is determined by the Closed Loop Controller that is responsible to orchestrate the closed loop, including both world simulation and brain simulation.

The data flow in Figure 4 is as follows: Sensors from the robot are captured in a robots sensor. In our simulation, this sensor is a virtual camera that captures the simulated room, but may also look through physical cameras in a physical room. The sensor data is exposed through a ROS topic. The robot control adapter has created a ROS Subscriber as the Transfer Functions has informed it that it will need the camera topic. The robot adapter fetches the data and provides it via a communication object.

The Transfer Function Component takes the data from the communication object and forwards it to the transfer function that actually converts the data to spikes or more precisely reconfigures spike generators accordingly. To do this, the transfer function has access to a spike generating communication object. This object has been created by the brain communication adapter after the Transfer Functions have told the brain adapter that such a communication is necessary. The brain communication adapter knows the connection of the spike generating communication object to the physical spike generator device in the neuronal simulator and forwards the configuration change to the device through the PyNN interface.

Meanwhile the organization of the data flow is organized by the Transfer Functions Component, the orchestration (controlling when the data is actually transferred), is handled by the Closed Loop Controller.

In Figure 4, the following objects conform to the following interfaces:

- **Closed Loop Controller:** IClosedLoopControl
- **Brain Controller:** IBrainControlAdapter
- **Robot Controller:** IRobotControlAdapter
- **Transfer Functions:** ITransferFunctionsNode
- **Brain Adapter:** IBrainCommunicationAdapter

- **Robot Adapter:** IRobotCommunicationAdapter
- **Spike Generator:** ISpikeGenerator. The exact interface is determined by the spike generation device type
- **Voltage Detector:** ISpikeDetector. The exact interface is determined by the spike detector type
- **Camera Subscriber:** IRobotSubscriber
- **Left Motor Publisher:** IRobotPublisher

The implementations for all of these objects is subject of the CLE, although the implementation may forward requests to some library such as PyNN. For all of these interfaces, mock implementations are also available in order to test as much as possible from the architecture in a separate unit test.

The following other objects do not belong to core CLE, but are either reused from other components or internal implementation details:

- **Generator device:** This is a PyNN device. Most likely, this will be one of the devices already implemented in PyNN, we just reference it.
- **Artifical neuron:** This has to be supported directly by the neuronal simulator and is thus also reused from PyNN.
- **Camera-**Sensor: The camera sensor is the virtual camera sensor of the simulated robot, in Figure 4 a husky robot.
- **Left Motor:** Same as before, this is an existing sensor of the virtual robot.