NEURONSCAPE SYSTEM SPECIFICATION

VERSION 0.01 DRAFT

Julian Bailey

School of Electronics & Computer Science

University of Southampton, UK

jab@ecs.soton.ac.uk

17th June 2010

1 Neuronscape System Structure

The Neuronscape system is made up of three different parts, each performing different function.

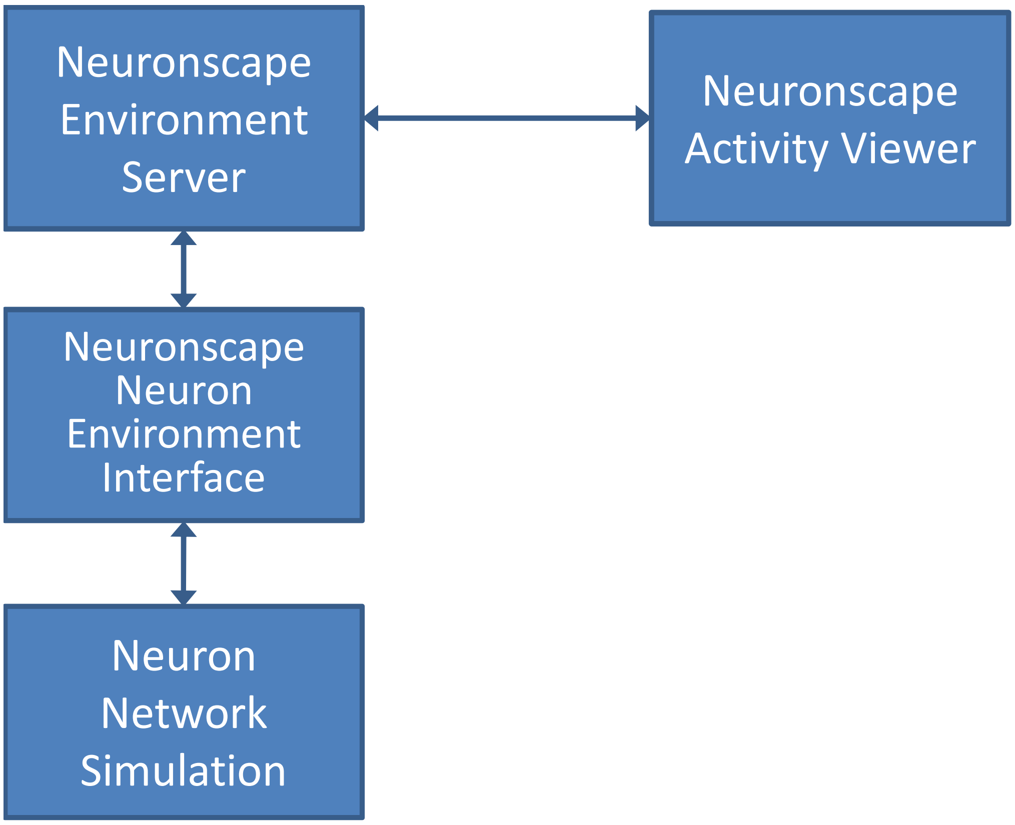
Neuronscape Environment Server
Neuronscape Activity Viewer
Neuronscape Neuron Environment Interface
Neuron Network Simulation

Figure 1.1: An overall view of the neuronscape system

The overall system view is shown in Figure 1.1, the Environment server (ES) is responsible for tracking all the objects in the environment, handling tasks such as environmental physics. The Neuron Environment interface (NEI) is responsible for muscle and receptor modelling, in other words, it is responsible for interfacing between the neuron network simulation and the virtual physical envrionment. Finally the activity viewer (AV) allows a person to peek into the virtual world, altering parameters and allowing external interactions.

At a minimum a system requires the Server module and the Neuron Environment interface module therefore these are considered "core components" of the system.

The purpose of dividing these components is to allow processing on separate machines, with many virtual animals in the environment many calculations may have to be performed in order to model, environment physics, reception behaviour and muscle behaviour. By dividing the load less pressure is put on one processing device so processing can be divided easily between machines that are part of a processing cluster.

The next three sections describe the operation of each of the components in greater detail.

1.1 Environment Server

The environment server holds the *master list* of all the objects in the system, an object can be an animal controlled by an NEI or an inanimate object such as food any other environmental "furniture".

The absolute position of the object is tracked by the server as sets of Cartesian coordinates along with the orientation of the object in radians and the velocity of the object. The magnitude and direction of any muscle force and external force acting on the object is also held since it is used to calculate future position and velocity.

The physics in the environment is modelled according to classical mechanics; as such the environment obeys the conservation of momentum in a closed system and Newton's laws of motion.

Time progresses in the environment through *ticks*, the period of which can be defined by the user. Each *tick* the forces acting on each object are evaluated and the position and velocity of each object can change. Collisions between objects are also calculated during the *tick*.

1.2 Neuron Environment Interface

The Neuron Environment Interface provides a method by which a network of neurons can interact with the environment, providing sensory input and output via forces generated by muscles. It "owns" an object that exists in the Environment server, which means it is responsible for generating the forces which will move the object and reading the sensory data from the environment, processing it and passing it to the neuronal network simulation.

The NEI holds a local copy of the various objects in the environment, this is important for rendering the vision sensory input.

The NEI component has an internal *tick* system like the server which allows complex muscle and receptor models to be used.

1.2.1 Sensory Input

Initially sensory input will be offered through sight and touch. For sight the objects in front of the animal will be rendered and a bitmap of what the animal can see will be generated allowing this visual plane to be used to drive light receptor models which in turn will provide input to the neuronal network.

The default size of the viewing area will be 320 x 240 pixels; this can be translated on to a smaller number of light receptors if a 1:1 pixel receptor mapping is not required. Stereoscopic vision is possible by rendering two views of the environment offset by a user defined distance.

Sensory input such as touch so the neuronal network can know if a collision with an object has occurred will also be present as required.

1.2.2 Muscle Output

Movement is achieved by the action of muscles; the force vector derived from the muscle action is fed into the environment which calculates the resultant motion.

Initially if several muscles are acting at the same time the NEI must calculate the resultant vector and feed it to the environment.

1.3 Activity Viewer

The activity viewer is the way that the user of the system can interact with the environment, seeing the environment through the eyes of an animal or a god type view showing the positions of all the objects. Like the NEI the viewer holds a local copy of the list of objects which is updated by the server when changes occur.

The view is rendered locally on the machine the viewer program is run on so causes no significant load on the server.

The end goal is to allow the user to push and touch objects in the system and watch how they respond or change the physical environment.

2 Network Communication

Communication in the neuronscape system is achieved using the TCP/IP protocol, and as such, the communications protocol sits in the application layer in the TCP/IP reference model.

2.1 Basic Packet Structure

The basic packet structure (shown in Table 2-1) consists of a 2 byte packet ID, a 2 Byte Length and the payload. The value specified by *Length* is the length of only the payload of the packet, this means all packets should be a minimum of 4 bytes long if they do not include a payload. This does not include the overhead of the TCP/IP protocol.

All multi-byte values are transmitted in Little-endian format.

Table 2-1: Generic Packet Format

Description	**Packet ID**	**Length**	**Payload**
Field Length	**2 Bytes**	**2 Bytes**	**=< 1496 Bytes**
Value	**3 (0x0003)**	**=< 1496 Bytes**	

There are three broad types of packet, Control packet for indicating responses to data that has been sent, Server packets which are used for by a client when transmitting data to a server and Client packets, used for a server transmitting to clients.

2.2 Control Packets

Control packets are sent to indicate the response of the receiver to the packet sender. These come in three forms, Acknowledge, Negative Acknowledge and Error. With the exception of the Error packet the control packets have no payload.

2.2.1 Acknowledge (ACK)

The acknowledge packet is sent to indicate successful processing of a command or a positive response to a command.

For example, in response to a client setup packet, the server should send an acknowledge indicating that the client has been accepted by the server.

The format of the ACK packet is shown in Table 2-2.

Table 2-2: Acknowledge (ACK) Packet Format

Description	Packet ID	Length
Field Length	2 Bytes	2 Bytes
Value	0 (0x0000)	0

2.2.2 Negative Acknowledge (NAK)

The negative acknowledge packet is sent to indicate unsuccessful processing of a command or a negative response to a command. In the case of a command which expects an ACK in response the NACK packet should be immediately followed by an Error (ERR) packet indicating why the previous command failed.

The format of the NAK packet is shown in Table 2-3.

Table 2-3: Negative Acknowledge Packet Format

Description	Packet ID	Length
Field Length	2 Bytes	2 Bytes
Value	1 (0x0001)	0

2.2.3 Error (ERR)

The error packet is sent to indicate that an error has occurred and the error code associated with the error. Under a typical scenario it is sent immediately following a NAK packet to indicate why the previous command failed.

The format of the error (ERR) packet is shown in Table 2-4 and a table of error codes is given in Section 2.7 on page 19.

Table 2-4: Error Packet Format

Description	Packet ID	Length	Error Code	Other Data
Field Length	2 Bytes	2 Bytes	2 Bytes	64 Bytes Max
Value	2 (0x0002)	2 (0x0002)	See Section 2.7 on page 19	See Section 2.7 on page 19

2.3 Server Packets

Server packets are defined as packet which flow in the direction from client to server to be processed by the server. The structure of the basic server packet is shown in Table 2-5.

Table 2-5: Generic Format of a Server Packet

Description	Packet ID	Length	Packet Sub-Type ID	Payload
Field Length	2 Bytes	2 Bytes	2 Bytes	=< 1494 Bytes
Value	3 (0x0003)	=< 2 or 0x0002	See table below	Dependent on packet type

The packet ID and Length perform the same function as in the generic packet shown in Table 2-1. The packet sub-types have a unique set of meanings for the contents of the server packet, therefore a Packet ID of 3 indicates the packet is of the server type and that the sub-type belongs to the server packets group. The packet length is at least 2 because this type of packet must have a sub-type identifier. The various sub-types are shown in Table 2-6.

Table 2-6: Sub-Types of Server Packet

Packet Sub-Type ID	Name	Short Description
0 (0x0000)	Client Enumeration	Tells the server the clients role in the system and the command set it supports
1 (0x0001)	Request Disconnect	Asks the server to disconnect a particular client
2 (0x0002)	Request Reconnect	If an unexpected disconnect occurs then the client can request a reconnect. This is done instead of re-enumerating
3 (0x0003)	Forces Packet	Tells the server about the various locomotive forces acting on an object

Each of the types shown in the table are described in the following sub sections.

2.3.1 Client Enumeration

The *client enumeration* packet gives the server information about the client, for example, the role performed by the connecting client and the maximum command set version supported. It is expected that command sets should be backwards compatible. The Client Enumeration packet format is shown in Table 2-7.

Table 2-7: Client Enumerate Packet Format

Description	Packet ID	Length	Packet Sub-Type ID	Device Type	Command Set Version
Field Length	2 Bytes	2 Bytes	2 Bytes	1 Byte	4 Bytes
Value	3 (0x0003)	0x0007	0x0000	See Table 2-8	e.g. 0x0001 (version 00.01)

The sub-type ID for the client enumeration packet is 0x0000 (0), the device type is specified as a single byte which tells the server what function the client performs, the valid types are shown in Table 2-8. The command set version is used by the server to determine the most recent command set the client supports. The server can then tell the client to disconnect if the command set is incompatible (thought command set are intended to be backward compatible). The command set version should be interpreted as two 16 bit unsigned integers with the lowest byte being the minor revision number and the highest byte being the major version number.

Table 2-8: Device Types

Device Types	Type Description
0 (0x00)	Undefined Role
1 (0x01)	Server
2 (0x02)	Neuron Environment Interface
3 (0x03)	Viewer

The different roles supported in this version of the command set are shown in Table 2-8, the table itself is quite straightforward. The only point to note is that a client reporting an undefined role will not be allowed to participate since this is the default state set by the server when I client first connects. The server type is included since in future versions the protocol will allow two servers to work in parallel and load balance.

<u>SERVER RESPONSE:</u> The server should respond with an ACK if the packet was ok, otherwise the server should send a NAK followed by an ERR packet. In this command version if a server

connects to another server then the response of the first server should be NAK followed by an ERR packet.

2.3.2 Request Disconnection

The *request disconnection* packet is sent to the server when a client indicates that it wishes to disconnect from the system. The server **MUST** acknowledge this packet since the client will always be allowed to disconnect. If the client disconnecting is an NEI then the object associated (controlled) by the NEI will be removed from the system. The packet format is shown in Table 2-9.

Table 2-9: Request Disconnect Packet Format

Description	Packet ID	Length	Packet Sub-Type ID
Field Length	2 Bytes	2 Bytes	2 Bytes
Value	3 (0x0003)	0x0001	0x0001

SERVER RESPONSE: The server must always respond with an ACK to a disconnection request. The server will then remove any object associated with the client.

2.3.3 Request Reconnection

The *request reconnection* packet is sent by a client to the server when the client wishes to reconnect to the system. This typically happens when an external network problem causes the TCP link to fail; this allows the client to resume as if the connection never happened. This type of packet would only be sent by a client in the role of a Neuron-Environment interface, otherwise the system will end up with an orphaned object. The format of the packet is shown in Table 2-10.

Table 2-10: Request Reconnect Packet

Description	Packet ID	Length	Packet Sub-Type ID	Object ID
Field Length	2 Bytes	2 Bytes	2 Bytes	1 Byte
Value	3 (0x0003)	0x0002	0x0002	(see description below)

SERVER RESPONSE: The server must look for the object referenced by the Object ID given in the packet. If the object ID does not exist or if the object is already attached to a client other than the current client the server must issue a NAK followed by an error packet with the appropriate error code, otherwise the server should ACK the reconnection request and associate the client with the object reference by the Object ID.

2.3.4 Forces Packet

The *forces* packet is sent to the server in order to update the locomotive forces acting on an object. No acknowledgement should be sent in reply to this packet since the number of forces packets would result in a considerable overhead in the system. The format of the packet is shown in Table 2-11

Table 2-11: Forces Packet

Description	**Packet ID**	**Length**	**Packet Sub-Type ID**	**Locomotive Force Magnitude**	**Locomotive Force Direction**	**Rotational Force**
Field Length	**2 Bytes**	**2 Bytes**	**2 Bytes**	**4 bytes**	**4 bytes**	**4 bytes**
Value	**3 (0x0003)**	**8 0x0008**	**3 0x0003**	**(see description below)**	**(see description below)**	**(see description below)**

The locomotive force magnitude is a 32 bit floating point value representing the driving force acting on an object.

The locomotive force direction is a 32 bit floating point value representing the direction in radians that the force is acting on the object.

The rotational force is a 32 bit floating point value representing the torque acting on the object, a negative value represents an anti-clockwise rotation whilst a positive value represents a clockwise rotation.

SERVER RESPONSE: No Response should be sent.

2.4 Client Packets

Client packets are defined as packet which flow in the direction from server to clients to be processed by the client.

Table 2-12: Generic Format of a Client Packet

Description	Packet ID	Length	Packet Sub-Type ID	Payload
Field Length	2 Bytes	2 Bytes	2 Bytes	=< 1494 Bytes
Value	4 (0x0004)	Dependent on packet type	See table below	Dependent on packet type

The packet ID and Length perform the same function as in the generic packet shown in Table 2-1. The packet sub-types have a unique set of meanings for the contents of the client packet, therefore a Packet ID of 4 indicates the packet is of the client type and that the sub-type belongs to the client packets group. The packet length is at least 2 because this type of packet must have a sub-type identifier. The various sub-types are shown in Table 2-13.

Table 2-13: Sub-Types of the Client Packet

Packet Sub-Type ID	Name	Short Description
0 (0x0000)	Client Setup	Allows the server to tell the client how large the environment is.
1 (0x0001)		
2 (0x0002)	Force Disconnect	Client must disconnect
3 (0x0003)	Add Object	A new object has been added to the environment
4 (0x0004)	Update Object	The position of an object needs to be updated
5 (0x0005)	Delete Object	An object needs to be removed from the environment

Each of the types shown in the table are described in the following sub sections.

2.4.1 Client Setup

The *client setup* packet allows the server to communicate the various environmental parameters to the client. This allows the client to accurately render the environment and all object contained within it. The format of the packet is shown in Table 2-14.

Table 2-14: Format of a Client Setup Packet

Description	Packet ID	Length	Packet Sub-Type ID	Environment Length	Environment Width
Field Length	2 Bytes	2 Bytes	2 Bytes	2 Bytes	2 Bytes
Value	4(0x0004)	6 (0x0006)	0x0000	See description below	See description below

The environment length and environment width set the dimensions of the environment in the client. These are unsigned 16-bit integers.

<u>CLIENT RESPONSE:</u> The client should response with an ACK or NAK if an error occurred. It may send an ERR packet after the NAK to indicate the error that occurred.

2.4.2 Client Setup Complete

The *client setup complete* packet indicates to the client that the setup phase is complete and the initial setup is complete. It should now expect to participate as part of the Neuronscape network.

Table 2-15: Format of a Client Setup Complete packet

Description	Packet ID	Length	Packet Sub-Type ID
Field Length	2 Bytes	2 Bytes	2 Bytes
Value	4 (0x0004)	0x0003	0x0001

<u>CLIENT RESPONSE:</u> The client should response with an ACK or NAK if an error occurred during the setup phase which was not picked up during the processing of one of the previous commands. It may send an ERR packet after the NAK to indicate the error that occurred.

2.4.3 Force Disconnect

The *force disconnection* packet allows the server to command the client to disconnect immediately. The format of this packet is shown in Table 2-15.

Table 2-16: Format of a force disconnect packet

Description	Packet ID	Length	Packet Sub-Type ID	Disconnect Code
Field Length	2 Bytes	2 Bytes	2 Bytes	1 Byte
Value	4 (0x0004)	0x0003	0x0002	See section 2.8 on page 19

The disconnect code indicates the reason for the disconnection, e.g. server shutting down. See section 2.8 on page 19 for the list of codes.

CLIENT RESPONSE: The client may or may not log the disconnection code. No response will be sent, the client must just disconnect. The server will wait for all clients to disconnect and close the connections, after 30 seconds the server will close all connections.

2.4.4 Add Object

The *add object* packet tells the client to add a new object to its local copy of the object list. This packet is sent during the client setup phase to initialise the list AND during normal operation if a new NEI connects while the system is already running. The format of the packet is shown in Table 2-17.

Table 2-17: Format of an add object packet

Description	Packet ID	Length	Packet Sub-Type ID	Object ID	Object X Position	Object Y Position	Object Orientation
Field Length	2 Bytes	2 Bytes	2 Bytes	1 Byte	8 Bytes	8 Bytes	4 Bytes
Value	4(0x0004)	23 (0x0017)	0x0003	See Below	See Below	See Below	See Below

The object ID is a single byte, this byte is used to reference the object in update and delete commands. The object x, and y position are double precision floating point numbers indicating the X & Y positions. Co-ordinate (0,0) is the top-left corner of the screen. Object orientation is a single precision floating point number indicating the direction the object is facing in radians.

CLIENT RESPONSE: The client must ACK or NAK the add object command to indicate success or failure. An ERR packet is expect to follow the NAK in the case of an error.

2.4.5 Update Object

The *update object* is sent once the client has been setup successfully (after the client setup complete packet) and an objects position or orientation needs updating. The format of the update object packet is shown in Table 2-18.

Table 2-18: Format of an update object packet

Description	Packet ID	Length	Packet Sub-Type ID	Object ID	Object X Position	Object Y Position	Object Orientation
Field Length	2 Bytes	2 Bytes	2 Bytes	1 Byte	8 Bytes	8 Bytes	4 Bytes
Value	4 (0x0004)	23 (0x0017)	0x0004	See Below	See Below	See Below	See Below

The object ID identifies the object whose details should be updated. The object x, and y position are double precision floating point numbers indicating the new X & Y positions. Co-ordinate (0,0) is the top-left corner of the screen. Object orientation is a single precision floating point number indicating the new direction the object is facing in radians.

CLIENT RESPONSE: On success the client will not send any response. If the client cannot find the object referenced by the given Object ID an ERR packet MUST be sent to the server so corrective action can be taken.

2.4.6 Delete object

The *delete object* is sent once the client has been setup successfully (after the client setup complete packet) and an objects needs removing from the system. The format of the delete object packet is shown in Table 2-19.

Table 2-19: Format of a delete object packet

Description	Packet ID	Length	Packet Sub-Type ID	Object ID
Field Length	2 Bytes	2 Bytes	2 Bytes	1 Byte
Value	4 (0x0004)	3 (0x0003)	0x0005	See Below

The object ID indicates which object should be deleted.

CLIENT RESPONSE: The client should ACK if the command completed successfully. If the object with the given ID is not found the client should NAK.

2.5 Communication Examples

2.5.1 Client Connects to the Neuronscape Server

1. A TCP Connection is established between the programs using the server address and the port number the server is listening on.
2. Once the connection is established the client must send a *Client Enumerate* packet to the server indicating the *role* the client fulfils and the *command set version* that the client supports.
3. In response the Server will either accept (ACK) or reject (NAK) the *Client Enumerate* packet.
4. On rejection of the Client Enumerate packet, the server will immediately send an ERR packet indicating the error which caused the client to be disconnected. The server will then close the connection.
5. If the client's connection is accepted by the server then the server must send a *Client Setup* packet to the client. This informs the client of the various environmental parameters.
6. The client must ACK the client setup packet.
7. The server should now send an *add object* command to the new client for each object that currently exists in the environment. This synchronizes the lists and positions of the objects in the environment with the local list held in the client.
8. The client must report if it managed to add the object to the local object list with an ACK or an ERR packet.
9. The server ends the connection phase by sending a *Client setup complete* packet to indicate that the client can now participate in the neuronscape network.
10. The client must now transmit ACK or transmit NAK and ERR packets to indicate any problem.
11. On receipt of the ACK the server may transmit any object manipulation commands to the client.

2.6 Summary of Network Protocol

The following sections summarise the control, server and client packets.

2.6.1 Control Packets

Description	Packet ID	Length	Payload
Acknowledge (ACK)	0x0000	0x0000	N/A
Negative Acknowledge (NAK)	0x0001	0x0000	N/A
Error (ERR)	0x0002	0x0001	Error Code (1)

2.6.2 Server Packet

Description	Packet ID	Length	Packet Sub-Type	Payload
Client Enum.	0x0003	0x0007	0x0000	Device Type (1) Command Set Version (4)
Request Disconnect		0x0002	0x0001	N/A
Request Reconnect		0x0003	0x0002	Obj. ID (1)
Forces		0x000E	0x0003	Force Mag. (4) Force Dir. (4) Rot. Force (4)

2.6.3 Client Packet

Description	Packet ID	Length	Packet Sub-Type	Payload
Client Setup	0x0004	0x0006	0x0000	Env. Length (2) Env. Width (2)
Force Disconnect		0x0002	0x0001	N/A
Add Object		0x0017	0x0002	Obj. ID (1)
Update Object		0x0017	0x0003	Object Pos.x (8) Obj. Pos.y (8) Obj. Orient. (4)
Delete Object		0x0003	0x0004	Obj. ID (1)

2.7 Error Codes (for ERR Packet)

Error Code	Error Description
0x0000	No Error
0xFFFF	Undefined Error

2.8 Disconnection Codes (for Force Disconnection packet)

These codes are used to indicate why the server is forcing disconnection a client.

Disconnection	Error Description
0x0000	Server Shutting Down (User requested server shut down)
0xFFFF	Undefined Unrecoverable Server Error, Server Quitting