Software Engineering Report Two

Peter Coetzee, Ismail Gunsaya, Chris Matthews, Fred van den Driessche, Stephen Wray

Introduction

Activities to date have been spread across three main areas of interest; the core of the project, a persistence framework, and the graphical user interface components. As such, this report is structured to reflect the progress in these sections of design and development. The project thus far has kept to its original service-oriented architecture (with some enhancements), which has made it possible to modularise development and make best use of each individual's time. The results of the activities since report one have been outlined below.

Project Core

The core of the project now consists of a few architectural “glue” components, around which the rest of the system is built. While these components were not a part of the original specification per se, they should enable the rest of the project's development to go smoothly. The components currently include a Plugin Manager, a Configuration Manager, and the Neural Network itself. The Plugin Manager service has been designed to permit any section of the project to easily have a set of dynamic plugins for a piece of functionality, without having to worry about re-implementing any logic for Java Reflection, or having to “find” plugins themselves. The Configuration Manager is one example of this in action; it executes a sequence of Configurator plugins at program initialisation, with each plugin responsible for configuring one section of the system; logging, statistics, etc.

The Neural Network portion of the core is built upon a generic Graph API, which permits it to be used directly as the basis for the GUI implementation as well as for network modelling and simulation. It offers a series of factory classes which permit a generic specification to be provided to describe how a given network is to be built. Furthermore, this architecture directly supports embedding of networks within networks, permitting the user to build their networks in small components and later build them up to a larger scale network. Various types of Neurone are also supported within the network, allowing it to exhibit whatever behaviour most interests the user.

In general, implementation of the project core has proved not too problematic. There have been occasional issues with interoperability – the lack of Java 1.6 for 32-bit Mac OSX caused some code to need re-implementing. An unexpected amount of time was lost in the conversion from Izhikevich's Matlab implementation of a particular spiking neurone behaviour to our object-oriented model; highly optimised Matlab code can be significantly challenging to untangle for the inexperienced developer! Despite this, the core is on track and the presence of the base Network implementations meant that the setback did not prevent progress from being made in other sections of the project.

No significant changes are planned for the project core going forward; the only remaining task for it is the implementation of Network Training, as well (potentially) as some currently out-of-scope enhancements in terms of the available library of neurone types. These modifications are planned for a later project iteration, as in its current state most developer activity is shifting to focus instead upon the GUI implementation.

Persistence

Key requirements of the project are the ability to save and load neural networks from the modelling tool and export to an intermediate representation for use with other tools. The implementation of these two goals are similar enough to warrant a general Persistence sub-system to perform both tasks. While the final specification for the intermediate representation had not been formalised we required Java object serialisation tools to allow persistence of networks for development purposes. The core persistence services were therefore created to be as adaptable as possible to reduce the changes needed to add the intermediate representation functionality. The added advantage of this extendable structure is adding new persistence formats is straightforward via plugins through the Plugin Manager.

The intermediate representation has been adapted slightly, although within the scope of their original goal of providing the ability to export the network to an XML file. The selected schema for this has been finalised as NeuroML. NeuroML is an online project which aims to create a standard for computational neuroscience to describe models of neural systems. The XML export feature of the Neural Modelling Tool is required as the XML is to be used as an intermediate language which can then be transformed into Matlab or Cuda for General-Purpose computing on Graphics Processing Units (GPGPU). Integrating the NeuroML schema allows the XML produced to be more freely used in other applications, as well as imported from them, thus increasing the interoperability of the tool.

Discussion with the project supervisor led to XML being selected as the intermediate representation, with options existing to create our own schema, use NeuroML, or using PyNN (another specification for representing neural networks). The suggested path to take was the NeuroML schema, although the final decision rested with the team. After comparison between the three different paths NeuroML was selected, as it was the most mature of the two existing specifications and matched our model closer than PyNN. We discounted creating our own specification because of the lack of interoperability with other software. The project sponsor was happy with this being the best decision, particularly as it represented a potent feature extension on the original specification by facilitating this interoperability.

The metrics of progress for the XML have not changed as the Plugin Manager is already in place. The XML serialisation service is one such plugin, which handles the exporting and importing of networks. The only potential problem that threatens the completion of this module on time is the amount of time that is needed to learn and understand the NeuroML libraries and integrating them into the project, such that they do not leak large dependencies beyond the scope of their package.

Two iterations are planned for the XML service. The first of these will consist of exporting a Neural Network to NeuroML, while the second encompasses NeuroML import. This delineation of activities have been selected as NeuroML export is the feature that is more critical to the success of the project. The current schedule for the module is completion 20th November for demonstration to the project supervisor.

User Interface

Despite initially encountering some difficulties in making the user interface display graphs nicely and having to make several architecture changes accordingly, it has not been necessary to modify the key requirements. The initial plan was to draw the graph using the cross-platform Standard Widget Toolkit (SWT) but it was later discovered there was no native support for non-rectangular canvases, or those with transparent backgrounds. This is unsuitable for displaying networks unless the entire graph is drawn on a single canvas, and this method would require a large amount of time-consuming code to pass mouse events to individual elements, move them around and so on.

The Draw2D library, as later discovered, contains this exact code. Draw2D is a subset of Eclipse's Graph Editing Framework, which lets the user draw several graph elements on a single SWT canvas. Events are automatically passed to the correct element, and the movement of nodes and edges is handled gracefully. This seemed to be the solution to the problem, but a lack of clear documentation caused later problems implementing scrolling, dragging and other common graph functions. We then discovered the Zest project, which builds on top of Draw2D and provides these features. All existing user interface code has been cleanly ported to this system, which should simplify future progress.

Currently, the user interface efficiently imports networks from our system's internal representation, while trying to keep the memory footprint small as possible. These networks are then displayed as a graph on an auto-resizing, scrollable canvas. Graph layout can either be changed manually by dragging nodes with the mouse, or automatically via an extendable graph layout interface.

Measurement of user interface progress is relatively unchanged as despite these setbacks, our initial iteration perfectly supports basic graph rendering and UI interaction as described in report one. Future iterations of the user interface will support zooming, graph structure editing, modular and layered networks, and network training and simulation. Instead of scheduling these features sequentially as milestones, they have been split into separate plugins and will be produced concurrently by different group members for the remainder of the project.