Dissertation - A support tool for Boolean networks

William Peckham (Student ID: 130227940)

Computing Science

Supervisor: Dr Jason Steggles

Word Count : 14889

Declaration

"I declare that this dissertation represents my own work, except where otherwise stated."

Acknowledgements

I would like to thank my supervisor, Dr Jason Steggles for his support and guidance throughout this project and for introducing me to the field of Boolean networks.

Abstract

This thesis investigates and develops techniques to produce visualisations of Boolean networks while tackling the problems associated with the state space explosion problem.

Using a state clustering technique and asynchronous priority based networks we researched and developed a tool supporting Synchronous and Asynchronous Boolean networks. Tackling the state based explosion problem and founded in human-computer interaction principles we present the resulting tool and the research produced from its development.

Contents

Chapter 1

Introduction

1.1 Boolean networks

Stuart Kauffman first proposed Boolean networks [34] in 1969. Kauffman's proposal was these resulting Boolean network could be used for studying the dynamical properties of gene regulatory networks [28, 46] and are now widely used in the system biology literature [24, 28, 34, 45].

A Boolean network is described as a state-based network consisting of N nodes, each taking the value of zero or one [28]. A node's state is determined by connections from other nodes (including itself) and a logical function defined with these aforementioned connections as inputs [28]. We depict these Boolean networks in wiring diagrams, such as the example in Figure 1.1 (explained in detail in Section 2.1).

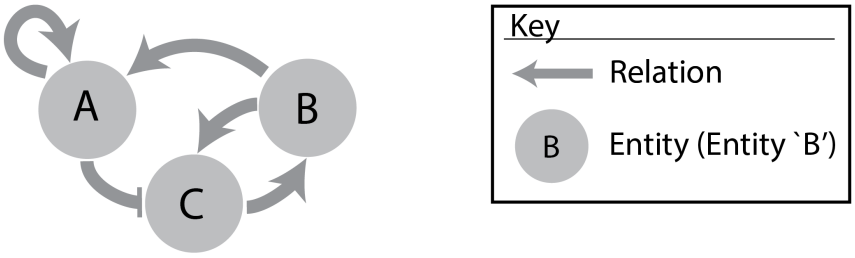


Figure 1.1: Example wiring diagram of a Boolean network

To understand the importance of Boolean networks, we observe how biologists (in particular reference to those studying Systems Biology) use gene regulatory networks to map genetic interactions [45]. After Stuart Kauffman's proposal for using Boolean networks for studying the dynamical properties of gene regulatory networks [34, 46], there have been various studies, such as those done by Thomas [50] studying the application of Boolean networks on Genetic Control Circuits. These studies have focused on the construction of Boolean networks from genetic networks [2, 36, 50]. The main purpose of doing so being the ability to understand the genetic causes behind the characteristics of organisms [24]. The Study of these Boolean networks enables us to discover properties about a set of specific genetic interactions [28] leadings to Boolean networks becoming an important tool in the modelling and analysis of large gene regulatory networks [5, 30, 35, 49, 61].

1.2 Motivation

A problem faced by users of Boolean networks is that tools currently available are often not simple to use; designed for specialists in the subject. Take the example solutions of

GINsim [59] and Cytyoscape; both solutions lack the application of basic Human-Computer Interaction (HCI) [19, 39, 42] and software design principles [11, 27, 43, 47], hindering on the learnability of their systems. Although well designed systems such as Boolesim [8] do exist they often are bespoke one off systems with very little functionality.

Although this projects focus is on bio-informatics, boolean networks has also be used within physics communities [20, 21]. In addition to studies undertaken into the application of boolean network in other areas such as within finance and economics [13, 40, 62]. Showing a broadening from the current use cases (of gene regulatory networks) and therefore an increase of the core user group. By adhering to HCI and software design principles a simpler to use system can be produced helping to broaden the current user group to non-expert users.

This project is also motivated to investigate the state space explosion problem; given a Boolean network with N entities there will be 2^N possible states [4]. This exponential growth causes the process of creating traces difficult with the addition of a single element increasing the total number of states exponentially. This behaviour can be seen clearly in Figure 1.2.

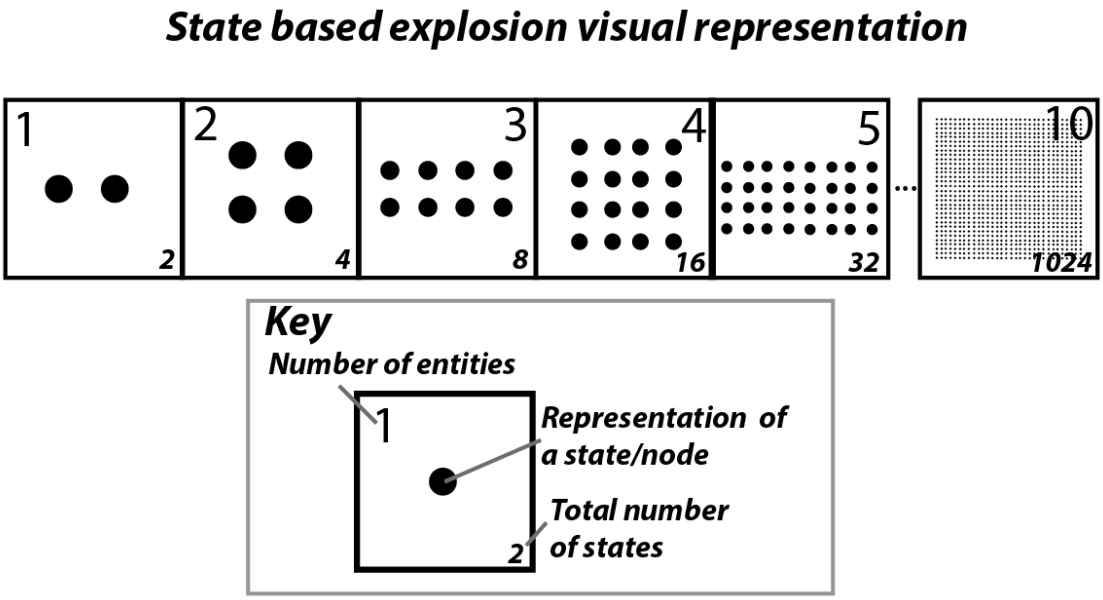


Figure 1.2: Visual representation of the state based explosion problem

In summary, the motivation of this project was to design and build a practical Boolean network tool, by applying principles of usability to the creation of Boolean networks to bring their usage to a larger user group. The project will also investigate and develop techniques to produce visualisations of Boolean networks. With the addition of tackling the problems associated with the state space explosion problem via the application of a set of visualisations techniques.

1.3 Aim and objectives

1.3.1 High Level Aim

To design and develop a support tool for creating and visualising the behaviour of Boolean networks.

1.3.2 List of objectives

Below is a list of objectives, provided with a brief reasoning in italics.

1. To identify and evaluate a selection of at least 3 Boolean network examples.

   To gain an understanding in the working of Boolean networks it is necessary to identify and evaluate their behaviours.

2. To evaluate existing Boolean network modelling support tools and produce a summary report.

   To better understand the pros and cons of the existing systems in order to generate requirements for our system, as well as learning from the mistakes made in the past.

3. To identify techniques for visualising Boolean networks.

   To display information about a given Boolean network to a user it is necessary to have a collection of visualisations to display.

4. To identify a technique(s) for tackling the state space explosion problem.

   If time permits, to reduce the complexity of visualisation, methods need to be produced to help simply them.

5. To identify a list of requirements for a Boolean network support tool.

   Identification of functionality is essential for the development and evaluation of the tool, defining what the system will do rather than how.

6. To develop a Boolean network support tool prototype that incorporates network creation and techniques identified from Objectives 3 and 4.

   Creation of a prototype allows for a view of the system, before the creation of the final system.

7. To evaluate the Boolean network support tool prototype and produce a summary report evaluating each feature within the tool.

   Analysis provides an insight into the issues with the prototype that would be resolved in future development of the tool.

1.4 Project Schedule

1.4.1 Diagrammatic work plan

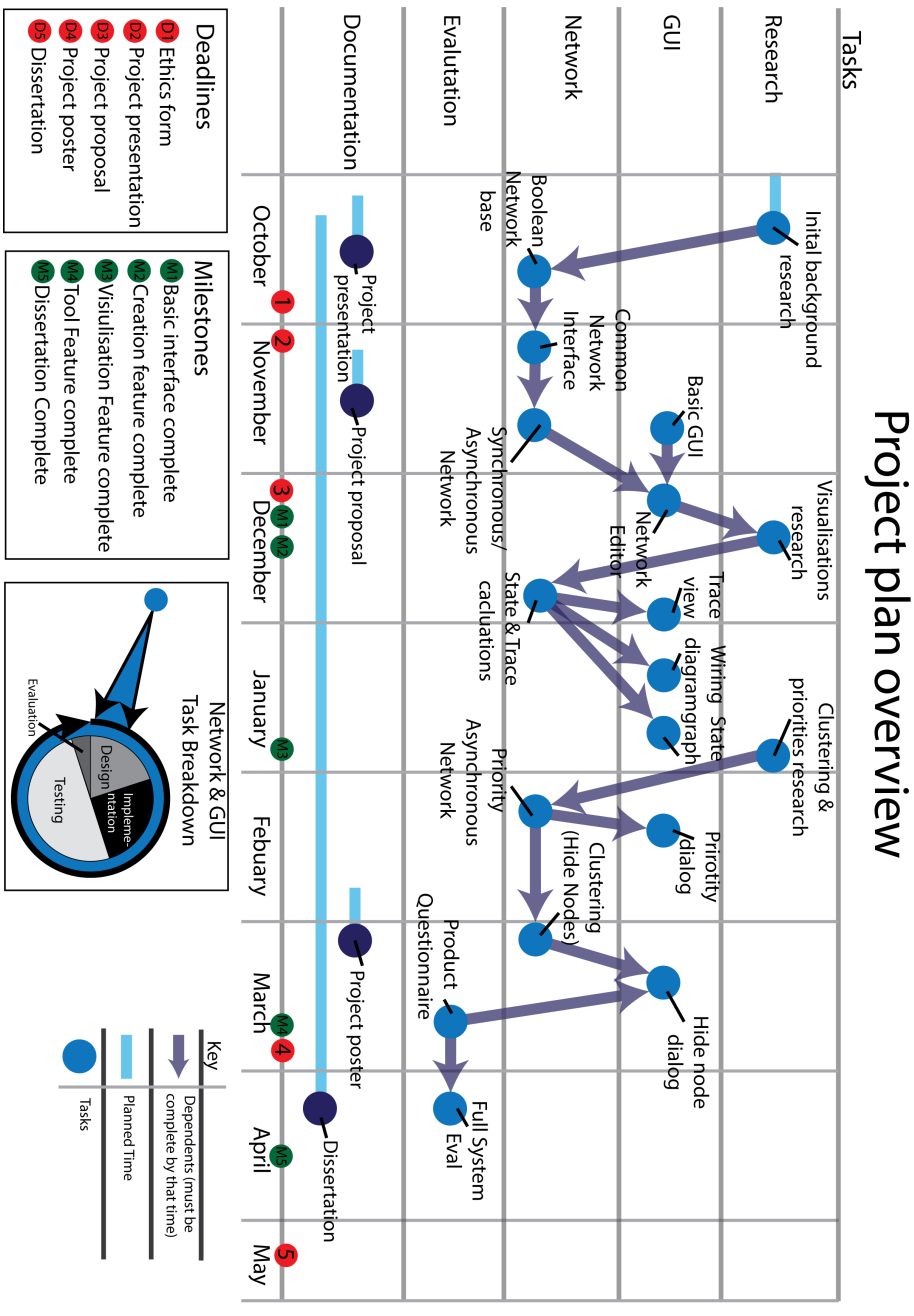


Figure 1.3: Project plan overview

1.4.2 Explanation of schedule

Figure 1.3 displays the overview of the final project plan, maintained throughout the project and runs from October (2016) to May (2017).

Plan structure/layout The plan is split into 5 key areas, 'Research', 'GUI' (Graphical User Interface). 'Network' (Network infrastructure), 'Evaluation' and 'Documentation'. The work to be done in each area is then split into a number of tasks. The planned time for a task is either decided by 'Planned Time' line or by an 'dependency' arrow. A task must be completed before at the end of the planned time (if defined) or before any of the task's dependent tasks' commence.

Summary of each area in the plan is listed below:

- **Graphical User Interface**

  Acting as the basis for the support tool as the primary interface on which all other features will be added.

- **Network infrastructure**

  For each of the features such as network creation and network visualisation the design, implementation and functional testing stages are kept consistent. However given each feature is integrated within the GUI with all other functionality additional time was planned to implement features into the GUI with the creation of dependent GUI tasks. On integration with the GUI automated tests will be run to ensure that the new feature has not created any unexpected behaviour within the tool.

- **Evaluation**

  A basic evaluation after each feature was completed as well as a full evaluation on the completion of the prototype helps ensure quality throughout the project. Full evaluation is performed with user feedback questionnaires and a review of system requirements allowing the production of a summary report.

- **Documentation**

  Throughout the project this report has be written and continual research for the dissertation paper was performed. In addition product documentation is taken into account as part of the implementation stage to each feature.

Development tasks Tasks within the 'GUI' and 'Network' sections are further split into an iterative style development task, where each task is an increment within the incremental model used. On the completion of these tasks there will be a functional subcomponent. As can be seen in Figure 1.3 a great deal of development time is dedicated to testing; automated functional tests will be written and run as each new section of functionality is added. Functional testing is then continued for an additional week to allow for additional testing and for maintenance to be performed if any error is found.

Milestones and Deadlines Milestone depict important steps in the project, mostly representing deliverable stages to the prototype before ending with the conclusion of the project. Deadlines show the hard deadline's provided by the university in regards to this project.

Chapter 2

Background

This chapter covers the background into Boolean networks and tools used to support them. It begins with a detailed outline the various terminologies used with the area of Boolean networks before moving into techniques that can be used to address the state based explosion problem.

2.1 Boolean networks

Within this section we will cover: the components of Boolean networks, the dynamics of their states and the difference network classifications. The section is structured so that we can gradually build up a picture of Boolean networks components and visualisations. So that when we delve into their classifications the terminology and diagrams used is clear. This sections breakdown is as follows:

- Components (make up) of a Boolean network.
- Dynamics of Boolean networks (How their states change over time).
- Methods for visualising Boolean networks and their states.
- Boolean network types and their differences/similarities.

There are a number of different networks used in research of gene regulatory networks. We will be concentrating on synchronous and asynchronous Boolean networks. Both are state-based networks consisting of N nodes each taking the value of zero or one. Where a node's state is determined by connections from other nodes (including itself) and a logical function defined with these aforementioned connections as inputs [28].

To help synthesize a clear picture of Boolean networks we have provided the network in Figure 2.1. We will use the **Entities** and **Boolean Functions** of this network (explained in section 2.1.1) to provide a consistent example throughout this section.

$$\begin{array}{ll} \textit{Entities:} & [\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}] \\ \textit{Functions :} & \left[\begin{array}{lcl} \text{A} & = & \text{A OR NOT C} \\ \text{B} & = & \text{A AND C} \\ \text{C} & = & \text{B} \end{array}\right] \end{array}$$

Figure 2.1: Boolean network Example

2.1.1 Make-up of Boolean networks

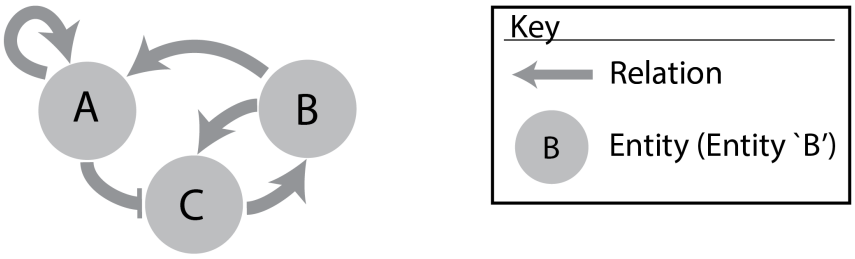


Figure 2.2: Network wire diagram of the Boolean network in Figure 2.1

Entities A Boolean network consists of a sequence of unique entities (nodes). Each entity's state is represented by a Boolean value (1 or 0). In our example network (Figure 2.1 & Figure 2.2) the entities are 'A' , 'B' and 'C' ($[A, B, C]$). [28]

Network State A state of a Boolean network (often refereed to as the *global state*) is a sequence of it's entities states. For example for the Boolean network depicted in Figure 2.2 its state can be depicted by $[State_{\mathrm{A}}, State_{\mathrm{B}}, State_{\mathrm{C}}]$. For example one of many possible states would be $[1, 0, 1]$.[28]

Boolean functions The state of a given entity within a Boolean network is determined by a corresponding Boolean function (command). These Boolean functions take the current state of the network and returns a Boolean value (0 or 1). A Boolean function consists of 6 main components; 'OR' , 'AND' , 'NOT' , '(', ')' and entities.

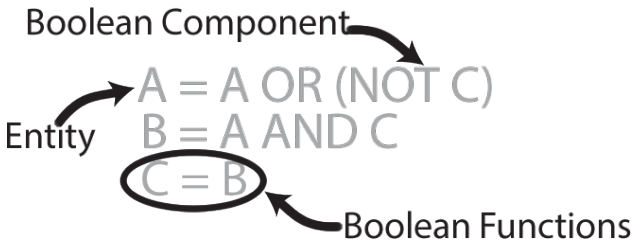


Figure 2.3: Boolean network functions

The example in figure 2.3 shows the functions from our example (Figure 2.1); we denote the next state of an entity with the syntax '*[Entity] = [Command]*'. To elaborate the entity's state to change is on the left of the equation with a Boolean operation on the right. For the example function B = A AND C we say the next state of B is the current state of A and the current state of C [28]

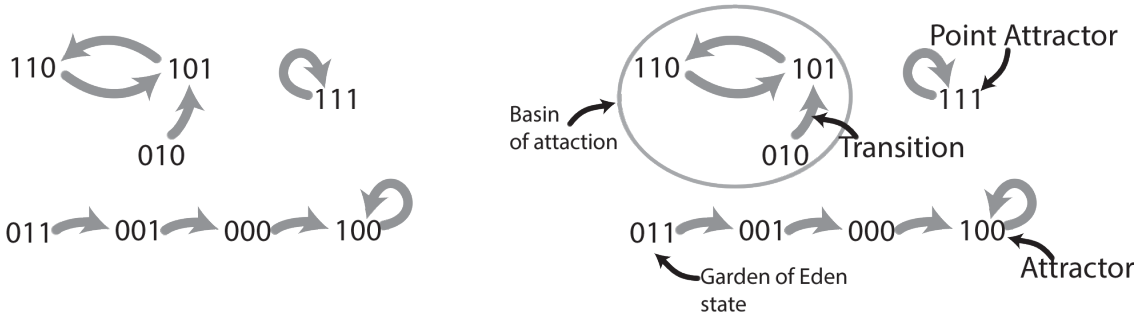
Relations Relations between entities are shown within a wiring diagram with directed arrows pointing from a entity to a entity that effects it's state as determined by its cor-

responding function (can be itself). When a relationship is negated the directed arrow is depicted with a flat end (see figure 2.2).[28]

Updating Schemes The updating scheme of a Boolean network defines the process on which the next state is evaluated.[28].

2.1.2 Network dynamics

The network dynamics of a Boolean network pertains to their states and processes on which they transition between these states.



(a) Synchronous state network (b) Annotated synchronous state network

Figure 2.4: Synchronous state diagrams

State transition A state transition is the changing of state; transitions are determined by: the current state and the network's Boolean functions and is dependent on the updating scheme of the network. As can be seen in our example in figure 2.4b (where we are using a synchronous updating scheme) an example transition is from the state 010 to the state 101:

$$[0, 1, 0] \longrightarrow [1, 0, 1].$$

To elaborate in the given example we have:

$$\begin{array}{ll} \textit{Current State :} & \mathbf{[0, 1, 0]} \\ \textit{Functions :} & \left[\begin{array}{lcl} \text{A} & = & \text{A OR NOT C} \\ \text{B} & = & \text{A AND C} \\ \text{C} & = & \text{B} \end{array}\right] \\ \textit{Updating scheme :} & \textbf{Synchronous} \end{array}$$

We can then use these to calculate the next state as follows:

$$\begin{array}{lcl} \text{A} & \longrightarrow & 0 \text{ OR NOT } 0 \\ \text{B} & \longrightarrow & 0 \text{ AND } 0 \\ \text{C} & \longrightarrow & 1 \end{array}$$

$\therefore$ the next state will be $[1, 0, 1]$

Trajectories A trajectory is a sequence of state transitions, for example in In figure 2.4b an example trajectory would be :

$$[0,1,1] \longrightarrow [0,0,1] \longrightarrow [0,0,0] \longrightarrow [1,0,0]$$

State attractors An attractor is a repeating trajectory seen as a cycle within a state transition. An example from figure 2.4b is :

$$[1,0,1] \longrightarrow [1,1,0] \longrightarrow [1,0,1]$$

Point attractors A point attractor is an attractor with only one element; this is the case where a state's next state is itself. In the example in figure 2.4 there are 2 point attractors $[1,0,0]$ and $[1,1,1]$:

$$[1,0,0] \longrightarrow [1,0,0]$$

$$[1,1,1] \longrightarrow [1,1,1]$$

Basin of Attraction A basin of attraction is all of the states that lead to a particular attractor, for example for the attractor $[1,0,0]$ the basin of attraction is :

$$[0,1,1] \longrightarrow [0,0,1] \longrightarrow [0,0,0] \longrightarrow [1,0,0]$$

Garden of Eden states A garden of Eden state is any state that no over state transitions to, or in other words is a state that cannot reached from any other state. Examples from figure 2.4 are:

$$[0,1,0] \text{ and } [0,1,1]$$

State space explosion problem Also know as combinatorial explosion the state space explosion problem refers to the exponential growth of possible states. Given a simple Synchronous Boolean network with N nodes the possible states can be calculated with 2^N given a Boolean entity has 2 values (0 and 1). For a multi valued network the possible states would be determined by X^N where N is the number of nodes and X is the number of possible states an entity can be in [6].

2.1.3 Visualizations

There are a number of visualisations used to depict Boolean networks and their behaviour; we will cover wiring diagrams, state transition graphs and network traces.

Wiring diagram The wiring diagram is a visual representation of the relations between elements within a Boolean network; using directional edges to show relations between it's entities. A simple example of this is shown in figure 2.5.

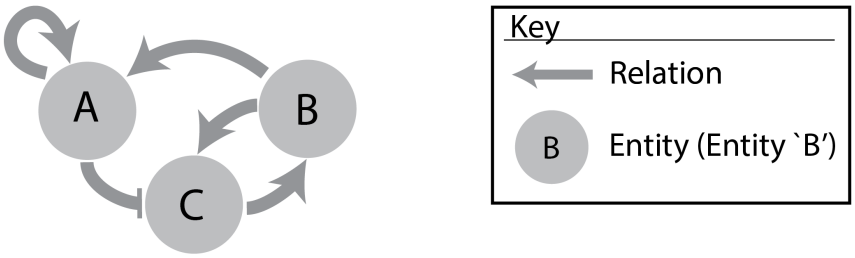


Figure 2.5: Wiring diagram

Wiring diagrams depend solely upon the function list of a network. The wiring diagram does not change for synchronous and asynchronous networks given it does not rely upon updating scheme. Leading to the composition of a wiring diagram to being a simple process. We take the function list and then map the relations within it to create a network. Let us take our example from figure 2.1, with the function list:

$$\textit{Functions :} \quad \begin{bmatrix} A & = & A \text{ OR NOT } C \\ B & = & A \text{ AND } C \\ C & = & B \end{bmatrix}$$

To generate the wiring digram as seen in Figure 2.5 we do the following:

(a) (b) (c) (d)

Figure 2.6: Wiring diagram creation example

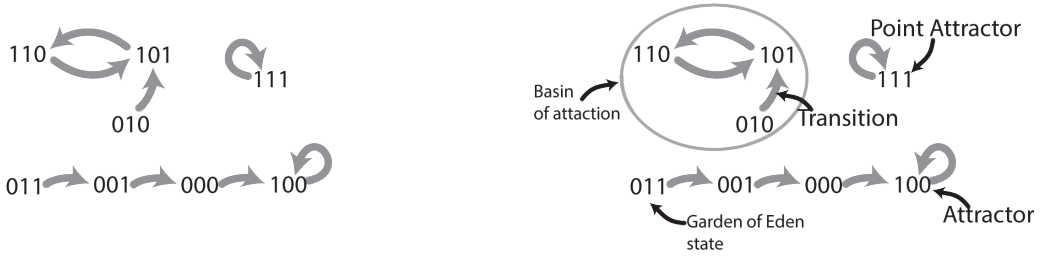
- Add all the entities as nodes to the graph. (see Figure 2.6a).
- Add relations for each entity:
  - Add relations for entity A using : **A** OR **NOT C** (see Figure 2.6b):
    - add directed arrow from entity A's node to itself.
    - add directed arrow from entity A's node to entity C's node.
      - *Note that Given C is negated within the Boolean operation we use a flat line as the end arrow (), these edges can be seen within figure 2.5.*
  - Add relations for entity B using :**A** AND **C** (see Figure 2.6c):
    - add directed arrow from entity B's node to entity A's node.

* add directed arrow from entity B's node to entity C's node.

– Add relations for entity C using :**B** (see Figure 2.6d):

* add directed arrow from entity C's node to entity B's node.

State transition graph The state transition graph is a representation of the transitions between states within a network. An example of a synchronous Boolean network's state transition graph is depicted in figure 2.7, it shows us the relations between all of the possible states.



(a) Synchronous state network (b) Annotated synchronous state network

Figure 2.7: Synchronous state diagrams

The composition of a state transition diagram is dependent on the updating scheme and the given network's functions. We have previously covered the dynamics of the state network within the '*Network dynamics*' section (section 2.1.2). For a given network there are number of approaches one might use to generate the state network graph. The method we will use is to find all possible states, then for each possible state we find the next possible state (synchronous) or states (asynchronous).

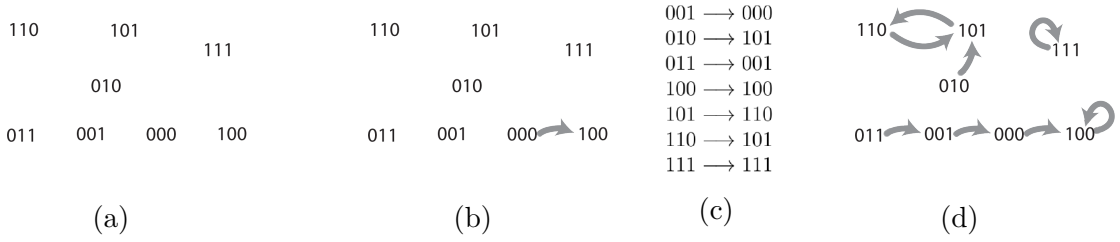


Figure 2.8: Synchronous state graph creation example

Figure 2.8 shows how we would generate a state graph for our example network. Using a synchronous updating scheme and our functions list in Figure 2.1.

- Find and add all the possible states as nodes to the graph (see Figure 2.8a).
- Starting with the state 000, find the next state and add a directional arrow to this state from the original state (see Figure 2.8b):
  - next state : 100

– add directed arrow from 000's node to 100's node.

- repeat for all possible states, finding each state's next state (see Figure 2.8c) and adding a direction arrow between the states (see Figure 2.8d).

Network Traces Network Traces are similar to state transition graphs and show the transitions between states. However traces are often shown in a text based form and a trace continues until an attractor (cycle in state transitions) is found.

$000 \rightarrow 1\ 0\ 0 \rightarrow 1\ 0\ 0$
$001 \rightarrow 0\ 0\ 0 \rightarrow 1\ 0\ 0 \rightarrow 1\ 0\ 0$
$010 \rightarrow 1\ 0\ 1 \rightarrow 1\ 1\ 0 \rightarrow 1\ 0\ 1$
$011 \rightarrow 0\ 0\ 1 \rightarrow 0\ 0\ 0 \rightarrow 1\ 0\ 0 \rightarrow 1\ 0\ 0$
$100 \rightarrow 1\ 0\ 0$
$101 \rightarrow 1\ 1\ 0 \rightarrow 1\ 0\ 1$
$110 \rightarrow 1\ 0\ 1 \rightarrow 1\ 1\ 0$
$111 \rightarrow 1\ 1\ 1$

Figure 2.9: synchronous Boolean network trace

Figure 2.9 shows the synchronous traces from our example network from Figure 2.1. To generate the traces we generate all the possible states, then for each state we calculate the next state until we find an attractor (See '*Network dynamics*' section 2.1.2).

Take the first trace of '$000 \rightarrow 100 \rightarrow 100$' starting with the first possible state of 000 we find the next state. Using the network's function list:

$$\textit{Functions :} \quad \begin{bmatrix} \text{A} & = & \text{A OR NOT C} \\ \text{B} & = & \text{A AND C} \\ \text{C} & = & \text{B} \end{bmatrix}$$

Given in our example we are using a synchronous updating scheme we perform the calculation as follows:

$$\begin{array}{lcl} \text{A} & \longrightarrow & 0 \text{ OR NOT } 0 \\ \text{B} & \longrightarrow & 0 \text{ AND } 0 \\ \text{C} & \longrightarrow & 0 \end{array}$$

$\therefore$ the next state will be $[1, 0, 0]$ (giving a trace of '$000 \rightarrow 100$')

$$\begin{array}{lcl} \text{A} & \longrightarrow & 1 \text{ OR NOT } 0 \\ \text{B} & \longrightarrow & 1 \text{ AND } 0 \\ \text{C} & \longrightarrow & 0 \end{array}$$

$\therefore$ the next state will be $[1, 0, 0]$ (giving a trace of '$000 \rightarrow 100 \rightarrow 100$')

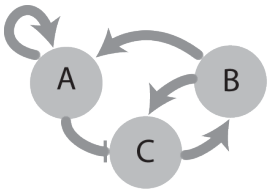
Here we have an attractor given '100 → 100' is a repeating trajectory (a cycle within a state transition) (See '*Network dynamics*' section 2.1.2). So we stop the trace and move on to the next possible state and repeat the process till all state traces have been performed.

2.1.4 Network types

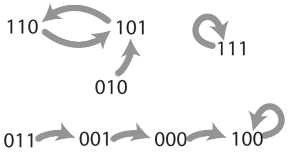
There are a number of networks that are used within gene regulatory network analysis. As we previously mentioned our concentration will be on the synchronous and asynchronous Boolean networks. However to allow for possible expansion of our tool we have considered a multitude of network types that we could support in the future.

Synchronous Boolean networks The synchronous Boolean network (SBN) has a simple network structure using a Synchronous update scheme, making one of the most common forms of the network used in the modelling of gene regulatory networks. [28]

Asynchronous network visualisations Below we have created the wiring diagram (Figure 2.10a), state transition graph ((Figure 2.10b)) and state trace (Figure 2.10c) for our simple Boolean network example (see Figure 2.1) using a synchronous updating scheme.



(a) Synchronous wiring diagram



(b) Synchronous state network

$000 \rightarrow 100 \rightarrow 100$
$001 \rightarrow 000 \rightarrow 100 \rightarrow 100$
$010 \rightarrow 101 \rightarrow 110 \rightarrow 101$
$011 \rightarrow 001 \rightarrow 000 \rightarrow 100 \rightarrow 100$
$100 \rightarrow 100$
$101 \rightarrow 110 \rightarrow 101$
$110 \rightarrow 101 \rightarrow 110$
$111 \rightarrow 111$

(c) Synchronous state trace

Figure 2.10: Synchronous figures

Synchronous updating scheme Within the synchronous updating scheme the next state is computed by finding the next state of all the entities at once. This updating scheme is deterministic; each state can lead to only one other possible state. This can clearly be seen from Figure 2.8b where each node in the graph connects to one (and only one) other node.

Using our example network from (Figure 2.1) with an starting state of 010 we would work out the next state in the following manner:

We calculate:

A $\longrightarrow$ 0 OR NOT 0
B $\longrightarrow$ 0 AND 0
C $\longrightarrow$ 1
$\therefore$ the next state will be $[1, 0, 1]$

Given:

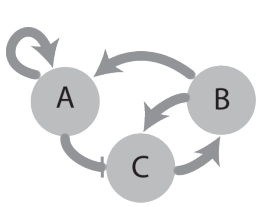
Current State : **[0, 1, 0]**

Functions :
$$\begin{bmatrix} A & = & A \text{ OR NOT } C \\ B & = & A \text{ AND } C \\ C & = & B \end{bmatrix}$$

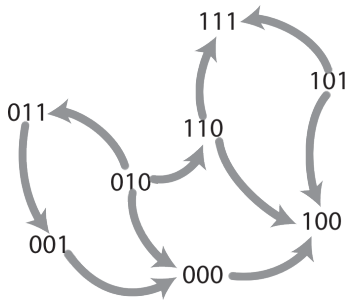
Updating scheme : **Synchronous**

Asynchronous Boolean networks Some argue that the Asynchronous form of Boolean networks are closer to that of biological occurrences; a key argument is that within biological processes genes do not change there states at the same time. However given their complexity, often the concentration is on synchronous Boolean networks [28, 32].

Asynchronous network visualisations Below we have created the wiring diagram (Figure 2.11a), state transition graph ((Figure 2.11b)) and state trace (Figure 2.11c) for our simple Boolean network example (see Figure 2.1) using a asynchronous updating scheme.



(a) Asynchronous Wiring Diagram



(b) Asynchronous state network

$010 \rightarrow 110$
$010 \rightarrow 000$
$010 \rightarrow 011$
$000 \rightarrow 100$
$011 \rightarrow 001$
$101 \rightarrow 111$
$101 \rightarrow 100$
$110 \rightarrow 100$
$110 \rightarrow 111$
$001 \rightarrow 000$

(c) Asynchronous state trace (partial)

Figure 2.11: Asynchronous figures

Asynchronous updating scheme There are a number of updating schemes for asynchronous networks. The approach we shall be using is to determine the state by taking an random entity and updating it (provided the state changes on update). An additional scheme is to introduce a time delay for each node whereby a node is only updated at the end of the delay. Unlike SBNs the asynchronous updating scheme is non-deterministic; each state can lead to a multitude of different states; clearly seen in Figure 2.11b [28].

Using our example network from (Figure 2.1) with an starting state of 010 we would work out the next states in the following manner:

We calculate:

A $\longrightarrow$ 0 OR NOT 0 $\longrightarrow$ 1
B $\longrightarrow$ 0 AND 0 $\longrightarrow$ 0
C $\longrightarrow$ 1 $\longrightarrow$ 1

A's state changed (0 to 1)
$\therefore$ $[\mathbf{1}, 1, 0]$ is a next state
B's state changed (1 to 0)
$\therefore [0, \mathbf{0}, 0]$ is a next state
C's state changed (0 to 1)
$\therefore$ $[0, 1, \mathbf{1}]$ is a next state

$\therefore$ the next states will be $[1, 1, 0]$, $[0, 0, 0]$ and $[0, 1, 1]$.

Given:

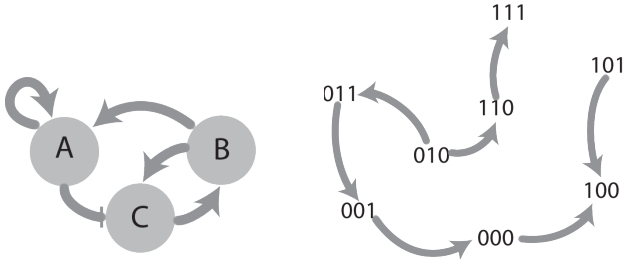
Current State : $\mathbf{[0, 1, 0]}$

Functions :
$$\begin{bmatrix} A & = & A \text{ OR NOT } C \\ B & = & A \text{ AND } C \\ C & = & B \end{bmatrix}$$

Updating scheme : **Asynchronous**

Priority Asynchronous Boolean networks As we have mentioned asynchronous network's non-deterministic behaviour leads to problems because of it's complexity. One way we can reduce the complexity of the asynchronous networks is to assign priorities to different entities. If we comparing the state network in Figure 2.12b with that of the default asynchronous network Figure 2.11b can see a clear reduce in connections and therefore the complexity.

Priority-based asynchronous network visualisations Below we have created the wiring diagram (Figure 2.12a), state transition graph ((Figure 2.12b)) and state trace (Figure 2.12c) for our simple Boolean network example in Figure 2.1) with the priorities '$A > B, C > B$'.



(a) priority asynchronous Wiring Diagram

(b) priority asynchronous state network

101 → 100
110 → 111
001 → 000
010 → 110
010 → 011
000 → 100
011 → 001

(c) priority asynchronous state trace

Figure 2.12: priority asynchronous figures

Priority-based asynchronous updating scheme The priority-based asynchronous updating scheme uses a list of priority rules to determine the next state and the normal asynchronous updating scheme. If a entity A has priority over another entity B then if on a given global state A's state can change B is ignored. In terms of gene regularity network this

could be when a gene will change the state quicker than another, for example if the gene A triggers gene B every time then B will never effect the state when A triggers, therefore we could add the priority '$A > B$'.

Using our example network from (Figure 2.1) with an starting state of 010 we would work out the next states in the following manner:

We calculate:

A $\longrightarrow$ 0 OR NOT 0 $\longrightarrow$ 1
B $\longrightarrow$ 0 AND 0 $\longrightarrow$ 0
C $\longrightarrow$ 1 $\longrightarrow$ 1

A's state changed (0 to 1) & there are no entities of a higher priority $\therefore$ $[\mathbf{1}, 1, 0]$ is a next state

B's state changed (1 to 0) however as state A/C changed and has priority over B $\therefore$ $[0, \mathbf{0}, 0]$ is **not** a next state

C's state changed (0 to 1) $\therefore$ $[0, 1, \mathbf{1}]$ is a next state

$\therefore$ the next states will be $[1, 1, 0]$ and $[0, 1, 1]$.

Given:

Current State : $\mathbf{[0, 1, 0]}$

Functions :
$$\begin{bmatrix} \text{A} & = & \text{A OR NOT C} \\ \text{B} & = & \text{A AND C} \\ \text{C} & = & \text{B} \end{bmatrix}$$

Updating scheme : **Priority-based asynchronous**

Priorities : $A > B$, $C > B$

Other networks The following are networks that we will not be developing the support tool for but have been researched to allow for considerations to be taken into account for the future expansion of the support tool.

Multi valued networks Although we will not be concentrating on Multi valued Boolean networks (MVNs) it is important to consider the requirements of MVNs to allow for MVNs to be integrated within the support tool with relative ease. A MVN is similar to a Boolean network, however rather than an entities state being determined by a Boolean value a state can have multiple difference states. MVNs can be updated using the same synchronous and a asynchronous updating schemes. [5, 6, 28, 32, 48]

Deterministic asynchronous Boolean network The deterministic asynchronous Boolean network approach to a asynchronous Boolean network involves the addition of two parameters per entity we will call P and Q. Both P and Q are natutal numbers ($P, Q \in \mathbb{N}$) and P will always be greater than Q. For any given entity will only update if it's P and Q values satisfy '$T \mod P \equiv Q$' (Current time step modulus P is equal to Q). Therefore we say that P is the period of the entity update and Q is the translation of the entity update

[28, 32]. Similar to the priority based asynchronous networks it reduces states and therefore the complexity when compared to the general asynchronous Boolean networks.

Mixed-context Boolean network The mixed-content Boolean networks uses the principles of a deterministic asynchronous Boolean network, however instead of a entities P and Q value being set values they change. The network is designed with a sequence of P and a sequence of Q values which relate to an entity. We call these P and Q values the context of the network; for each time step the context of the network will change, this could be cyclic or random in nature. The need for such networks comes from the need to map the fact that external factors can affect a biological system such as temperature and pressure of the environment. [28, 32]

2.1.5 State clustering

Node merging In order to reduce the complexity of the network visualisations we can hide specific nodes or entities from the graph. By hiding a node we merge states together significantly reducing the states from 2^n states to $2^{n\text{-}1}$ states whist additionally reducing the state connections as they are merged together. We can see this in the example below where we have hidden the node C.

010
010
010
000
011
101
101
110
110
001

(a) Possible states before hiding

010 → 110
010 → 000
010 → 011
000 → 100
011 → 001
101 → 111
101 → 100
110 → 100
110 → 111
001 → 000

(b) Before hiding

01
00
10
11

(c) Possible states after hiding node C

01 → 01
01 → 00
01 → 11
00 → 00
00 → 10
10 → 10
10 → 11
11 → 11
11 → 01

(d) After hiding node C

Figure 2.13: Node cluster (node hiding) figures

How these nodes are merged is depicted below in Figure 2.14 where a second entity is hidden. Here 00 and 10 both change to state 0, therefore merging the two states.

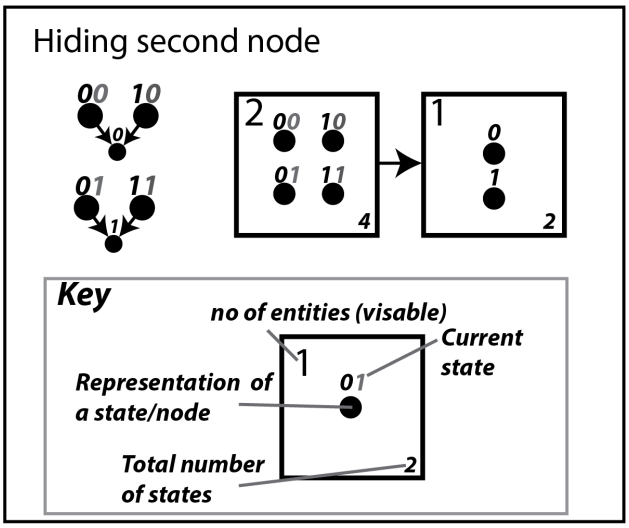


Figure 2.14: State clustering visual example

2.2 Existing systems

There a number of existing systems for the analysis of Boolean networks; this section provided a quick overview of these systems.

2.2.1 Boolesim

Introduction Boolesim is web based Boolean network simulator developed specificity for the modelling of gene registry and signal transduction networks [8].

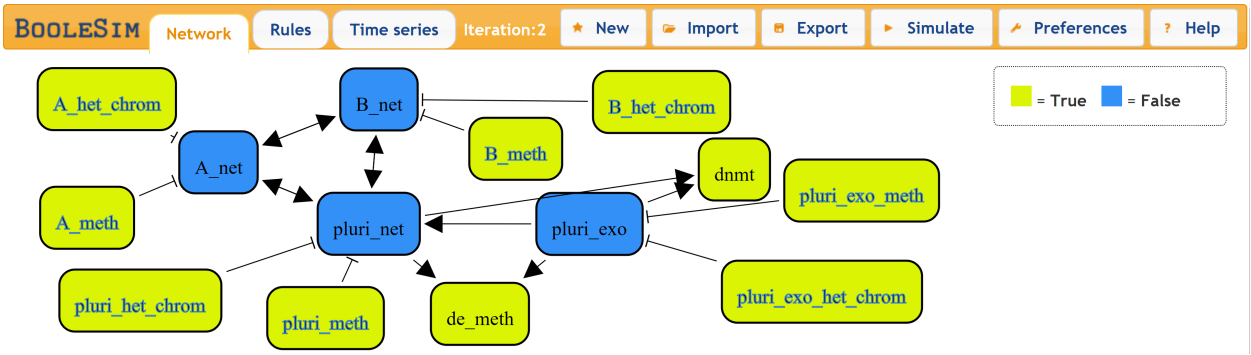


Figure 2.15: Boolesim with example network [8]

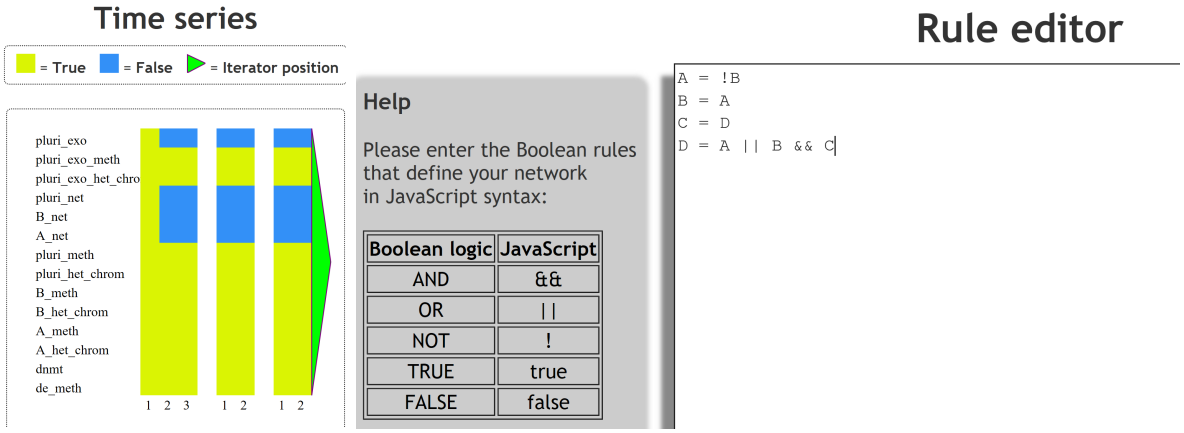
Key features The key features provided by BooleSim are it:

1. uses a number of different web technologies such as HTML5 and JavaScript (such as the rule editor (Figure 2.16b)). [8].

2. Supports network creation and modification. (as seen in Figure 2.15 and 2.16b)

3. Supports additional tools through Import and export functionality; providing a user with an easy way to import an existing network or export one to a image or another file format.

4. has a time series visualisation feature (Figure 2.16a)



(a) Boolesim's Time Series (b) Boolesim's rule editor

Figure 2.16: Boolesim's Figures [8]

2.2.2 GINsim

Introduction GINsim is an open source tool that utilises Java technologies and describes itself as a tool for the modelling and simulation of genetic regulatory networks'. It supports a large range of different Boolean network classifications and has been used to compare the behaviours of the different networks.

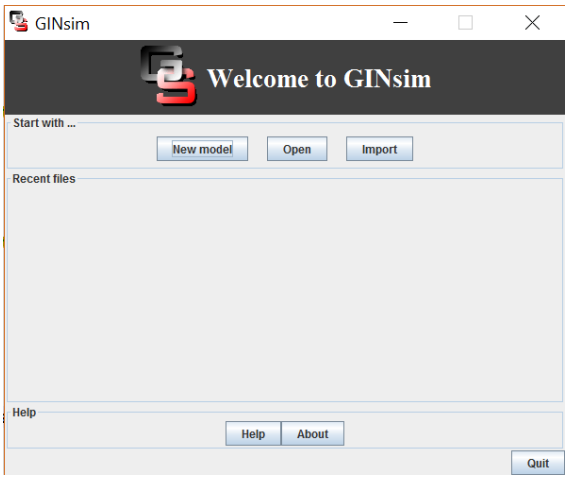


Figure 2.17: GINsim's home page [59]

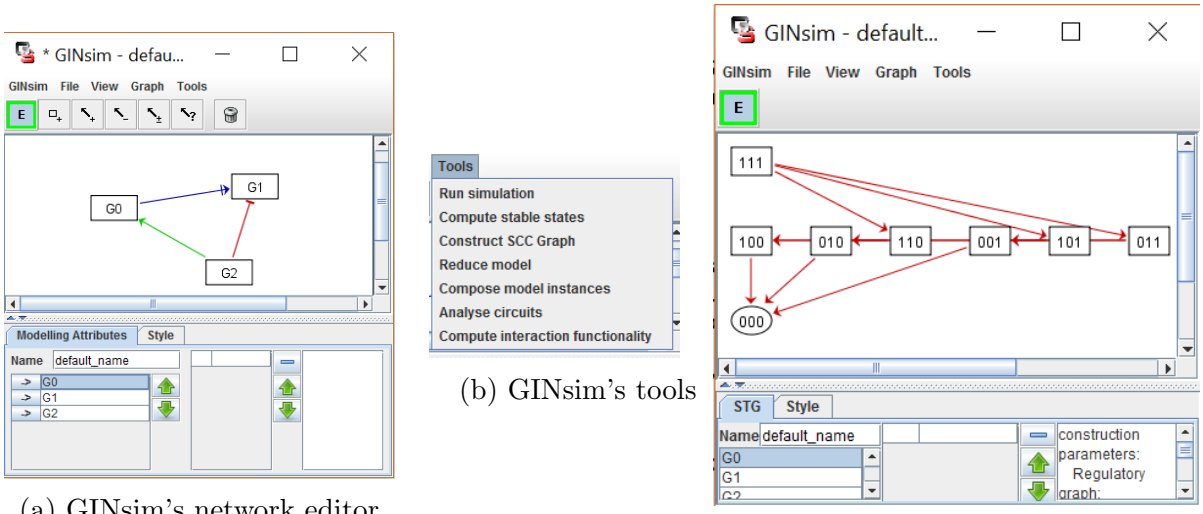
Key features The key features provided by GINsim are it:

1. uses the Java based visualisation library 'JGraphX' (See wiring Diagram in Figure 2.18a)

2. Supports network creation and modification. (see Figure 2.18a)

3. Supports additional tools through Import and export functionality; providing a user with an easy way to import an existing network or export one to a image or another file format.

4. has number of tools for running analysis on the network (Figure 2.16a & 2.16b)



(a) GINsim's network editor

(b) GINsim's tools

(c) GINsim's tools

Figure 2.18: Boolesim's Figures [59]

2.2.3 Cytoscape

Introduction Cytoscape is a Java based tool for visulazing molecular interaction networks. Although its base lacks in a lot a features it provides functionality through a number of Apps.

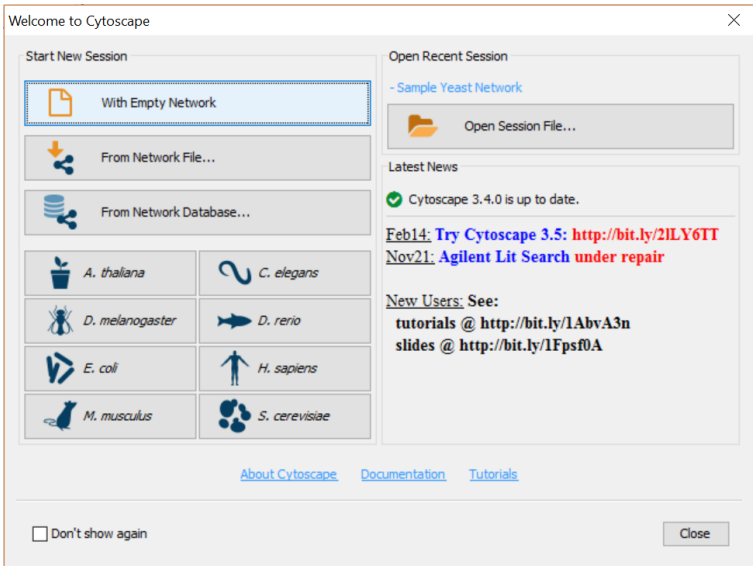


Figure 2.19: Cytoscape's home page [8]

Key features The key features provided by Cytoscape are it:

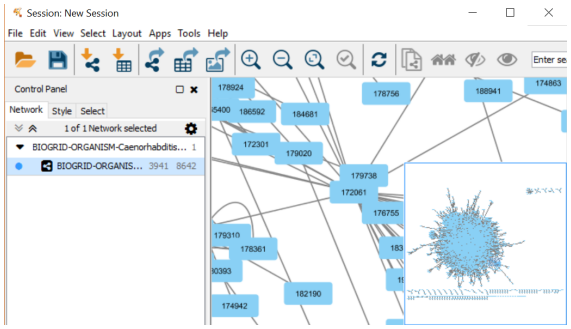
1. Supports additional tools through Import and export functionality; providing a user with an easy way to import an existing network or export one to a image or another file format.

2. Supports network creation and modification. (see Figure 2.20a)

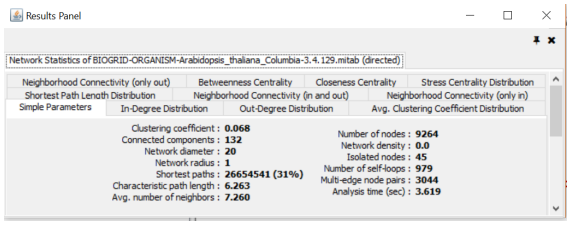
3. runs analysis on the network (Figure 2.20b)

4. supports extensions on itself via 'Apps' which supports further visualisations and analysis.

5. connects to public database clients for access of large networks.



(a) Cytoscape's graph view [8]



(b) Cytoscape's statistics [8]

Figure 2.20: Cytoscape's Figures [8]

2.3 Existing systems evaluation

2.3.1 desirable qualities comparison table

The following table gives an outline of the desirable features within each system.

Support tool	Boolesim	GINSim	Cytoscape (Base)	Cytoscape with Plugins
Network Editor tool	✓	✓	✓	✓
Generates Traces	×	✓	×	✓
Generates State Graph	×	✓	×	✓
Finds attractors	×	✓	×	✓
synchronous	✓	✓	×	✓
asynchronous	×	✓	×	✓
priority classes	×	✓	×	✓
Multivalued	×	✓	×	✓
Hiding Nodes	×	✓	×	✓
Time series	✓	✓	×	×
Visual simulation of state change	✓	✓	×	×
external public database support	×	×	✓	✓
interchangeable graph layout	✓(manual)	✓(manual)	✓(auto)	✓
custom graph styles	×	✓	✓	✓
network filter	×	×	✓	✓
Finding Clusters (subnetworks and highly interconnected regions)	×	×	×	✓
Plugin/App support	×	×	✓	✓
Import	✓	✓	✓	✓
Export	✓	✓	✓	✓
Aesthetics	✓	×	×	×

Table 2.1: Desirable features comparison table

Chapter 3

Software process

Before the development the system could commence, it was important to outline the series of predictable steps called a "software process"[43]. The key difference between methodologies being the flow between each activity (communication, planning, modelling, construction and deployment [43]), called the 'process flow'. The primary process flow models are the linear, iterative and evolutionary process flow models [11, 47].

3.1 Linear process

The linear process outlined in Figure 3.1 is used when given clear set of defined requirements. Used when producing well-defined enhancements or adaptations to an existing system and occasionally in new development projects provided with stable, clear requirements [43].

The Waterfall model [44] is common formation of the linear process model [9, 11, 43, 47]. The model allows for the movement back to a previous step, however is a difficult process that was elegantly put by Alen Denis with the phrase - 'Imagine yourself a salmon trying to swim upstream in a waterfall' [16]. The waterfall model has the advantage of definition of requirements early on the process allowing for limited changes to design and therefore reducing costs in development[16]. Its key disadvantage is the lack of flexibility, given the design must be completely specified before implementation [16]. Therefore making it unsuitable for this project.

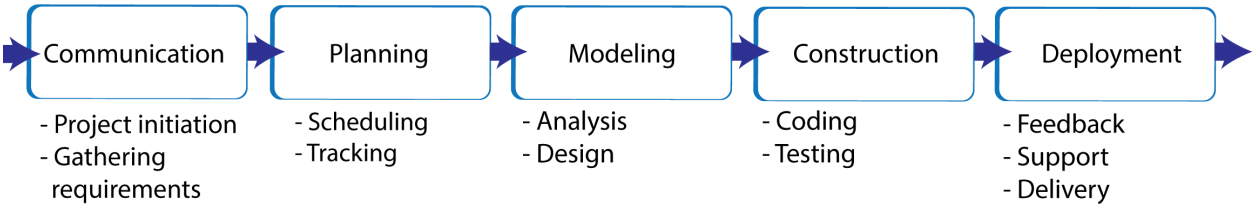


Figure 3.1: The linear model [43]

3.2 Iterative process

The iterative process outlined in Figure 3.2 uses the principles of the linear model, but repeats one or more activities before moving to the next. The incremental process is used to quickly provide core functionality before being expanded in future iterations [11, 47]. The key advantage being the flexibility provided, with the ability top add and remove requirements as we go. However this also acts as a disadvantage as a clear view of the system is not formed at the start and has a danger of continually adding features without a clear end.

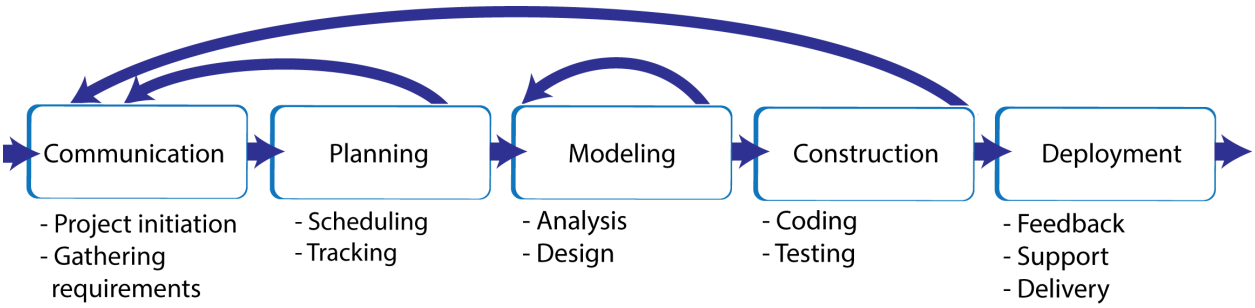


Figure 3.2: The iterative model [43]

3.3 Evolutionary process

The evolutionary process outlined in Figure 3.3 executes each activity one after each other in a cyclic manner. A evolutionary process is chosen when wanting to introduce the product to a user base as quickly as possible, deploying a limited version of the software before enhancing in further releases [47]. This advantage of being able to release an incomplete, but stable product attracts software development involving a degree of 'pioneering' [11, 26].

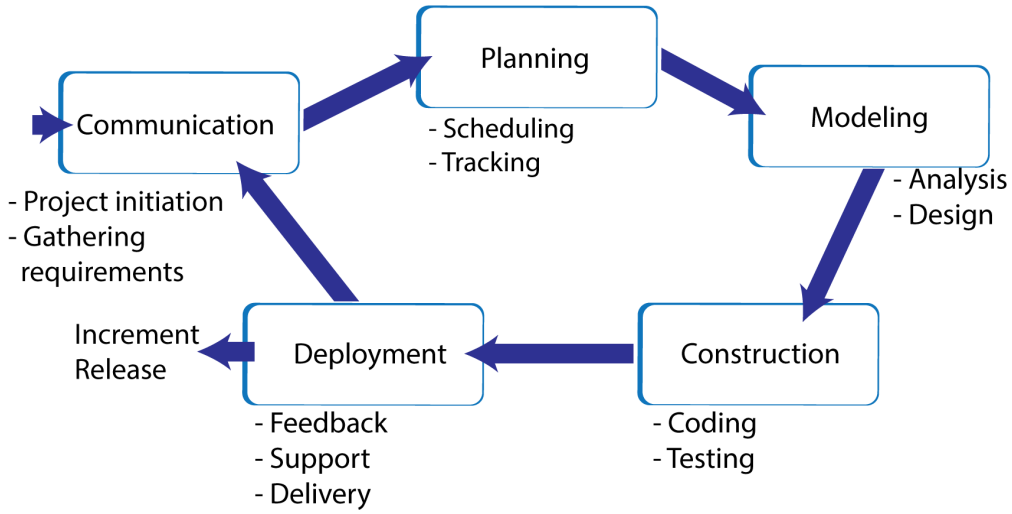


Figure 3.3: The evolutionary model [43]

3.4 Chosen process

Given knowledge of a fixed deadline and the possibility of additional requirements, the choice of process becomes between the iterative and evolutionary processes because of there adaptive natures. Given the system will have a single point of release (at the conclusion of the project) the project uses the iterative process flow model, more specifically the incremental process model shown in Figure 3.4.

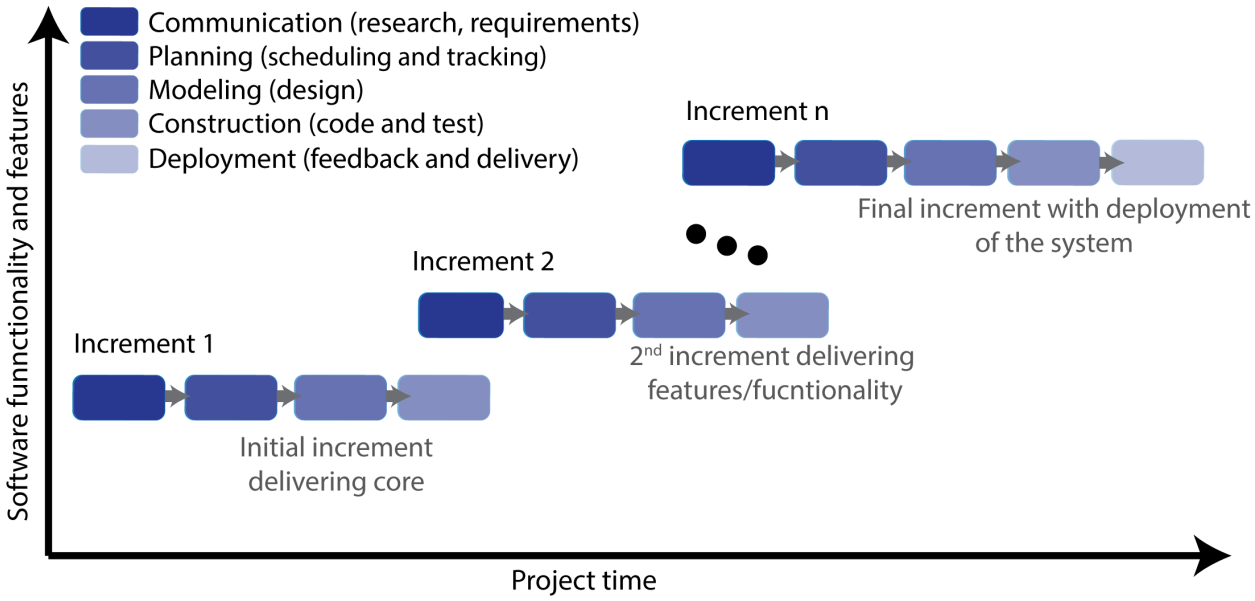


Figure 3.4: Iterative project schedule (based of Figure 2.5 in [43])

The chosen incremental model introduces parallelism to the iterative methodology; allowing for the design of future iterations to begin before the end of the previous ones. The completion of each iteration delivers a specific feature or functionality, incrementally getting closer to a complete system. On the completion of each increment research, requirements, plans and designs are reviewed, allowing for adaptation to be made throughout the software development progress. The initial iteration providing core functionality, on which all other iterations build upon.

Chapter 4

Requirements

Requirements are built on the idea that 'we have no need for an elegant solution that solves the wrong problem' - [43]. To develop our system correctly we required an understanding of the user's and the system's needs, achived through the creation of a set of requirements. A requirement being a statement of what the system must do or a trait is must have [10, 16, 27].

4.1 Inception and Elicitation

We begin with the task that defines the scope and nature of the project; the inception [47]. For this project the inception is defined by the aim, 'To design and develop a support tool for creating and visualising the behaviour of Boolean networks'. Using this basis we moved to requirements elicitation; the process on which we gather the user and system requirements [47]. The first step of this approach was to gain a domain understanding, what the system will do. The tool's primary purpose will be to perform tasks to aid with the study and understanding of regulatory networks. We then categorised functionality from basic to advanced features, lists of core functions (elicited from research) are summarised in the bullet point list below:

- **Basic functionality:**
  - Create and save synchronous and asynchronous networks.
  - Modify existing synchronous and asynchronous networks (Create, edit and delete network functions).
  - Create a wiring diagram from a network
- **Intermediate functionality:**
  - Create state traces for both synchronous and asynchronous networks.
  - Create a state graph for both synchronous and asynchronous networks.
- **Advanced functionality:**
  - Implement priority based scheme for asynchronous Boolean networks (PABN).
  - Implement node clustering (*using the 'hiding nodes' technique*)

With this functionality defined we elaborate expanding and separating into a concise set of requirements, each requirement categorised by it's type and given a priority. Traditionally requirements are split into functional and non-functional requirements [16, 27, 43, 47] and therefore is the method chosen.

4.2 Functional requirements

A functional requirement defines what the system should do, providing functionality or the behaviour to the system. Below is the set of functional requirements for the system.

Table 4.1: Functional requirements summary table

Requirement identifier	Name	Description	Priority
F1	New network creation	User must be able to create a new network from a choice of the available networks.	5
F2	Save networks to file	Users must be able to save a network that they are working on to file.	5
F3	Open existing files	Users must be able to open existing Boolean network files they they previously save (using the tool).	5
F4	Edit existing files	Users must be able to edit new or existing networks.	5
F5	Function manipulation	Users must be able to add, edit and remove functions.	5
F6	Wiring diagrams	The tool must provide a a user of a wire diagram view of a given network.	5
F7	State traces	The tool must be able to produce and show visually to a user the state traces for a given network.	4
F8	State graph	The tool must be able to generate and display state graphs for a given network.	5
F9	Export networks	The tool must be able to export a network into an image format.	3
F10	Import networks	The tool must be able to import networks from other tool layouts.	3
F11	Hiding nodes	The tool must be able to hide specific entities within a stage graph.	3
F12	Priority networks	The tool must be able to set priorities for a asynchronous network.	3

4.2.1 Non-Functional requirements

A non-functional requirement defines a property of how the system must work. Below is the set of non-functional requirements for the system.

Data requirements

Requirement identifier	Name	Description	Priority
D1	Long term storage	It must be possible to store a network long term in a file	4
D2	Network type	The system must be able to store the type of the network.	4
D3	Network entities	The system must be able to store a entity with its name.	5
D4	Network functions	The system must be able to store an entity's function/command	5
D5	State	The system must be able to store a networks state (short term)	5
D6	Traces	The system must be able to store a networks state (short term)	5

Table 4.2: Non-Functional data requirements summary table

Environmental requirements

Organizational/social environmental requirements

Requirement identifier	Name	Description	Priority
E1	User guide	User must be provided with support via a user guide	4

Table 4.3: Non-Functional data requirements summary table

Technical environmental requirements

Requirement identifier	Name	Description	Priority
E2	Platform Compatibility	System must be runnable on a desktop machine in particular a Windows PC	5

Table 4.4: Non-Functional data requirements summary table

Chapter 5

Design

5.1 Design model

The design model provides us with detail on software architecture, data structures, interfaces and components required to implement the system [43]. Here requirements and technical considerations came together in the formation of the system. The goal was to produce a model that establishes the quality of software [43].

5.1.1 Design concepts

Principles and concepts have been developed over the history of software engineering, by applying these techniques we can develop our solution effectively, avoiding the issued faced by similar tools in the past. The first such concept is 'separation of concerns' [17], is founded on the idea that any complex problems can be more easily handled when divided into independent pieces to be solved independently [43]. A concern is defined as a feature/behaviour of the system, which is split into smaller more manageable pieces. This is similar to the concept of 'divide-and-conquer; it is easier to solve a complex problem when it is broken into manageable pieces.

Modularity is how we apply these principles in our design. Modularity is a process whereby software is split into clear uniquely named components also called modules. Each module in turn provides abstraction, designed so information only required by itself is hidden from other modules that might access it. This act of abstraction provides numerous benefits in future testing and maintenance, but most importantly helps ensure a fault tolerant system [3]. Given most data and procedural details are hidden, errors introduced to one module are unlikely to propagate to other modules [3, 43].

5.1.2 Architectural design

Once the key concepts were understood, they where applied to generate an architecture for the system. The architecture provides us with a representation of the design decisions, overall structural view and communication between subcomponents. The architecture provides a way of analysing the effectiveness our design, allowing for us to modify the design with lower risks than if modifications were performed later down the line [43].

To design the architecture we could use a number of architectural styles, including 'data-centred' and 'layered' architectures. Below we discus how we might apply these styles to this project as well as the final architectural design.

Given the projects centralization over Boolean networks, we can produce a data-centred design whereby sub components update, delete, add and read from a centralised data source, the Boolean network (Depicted in Figure 5.1). An important factor that must be carried in all architectural designs is abstraction of the Boolean network; to provide space for future network additions (such as the MVN) we must abstract to use high-level view of a network.

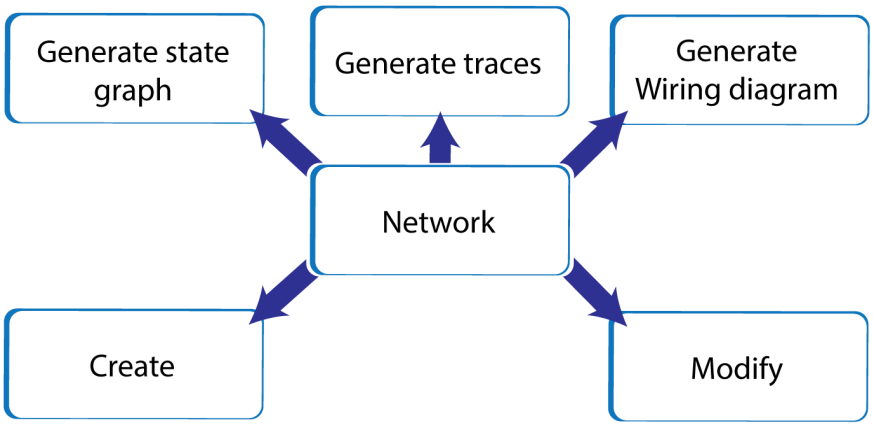


Figure 5.1: Data-centred architecture

A simple three tiered layered model could be represented as shown in Figure 5.2, here the system is separated by the Presentation, logic and data tiers. As we progress up the tiers we start with the data tier, here information is stored in this case the entities and functions of a network. Next we have the Logic layer where operations are run modifying or accessing the data. Finally the presentation layer provides the view of the system, in our case the GUI that the user will be presented with.

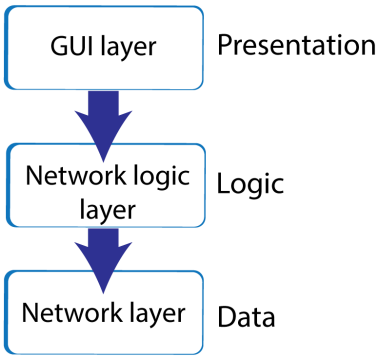


Figure 5.2: Data-centred architecture

Both architectures have their advantages, data-centred allows for components to be modified/added without effecting the others (as they work independently) and the layered architecture providing clear definitions between the users and developers view of the system. However given the portability and changeability provided by the layered system, allowing for modules to be replaced (providing the interface does not change) with ease. In addition to the downside with the data-centred approach that logical sub components such as creating

and modify a network would share some repeated functionality the layered architecture is the best for this system.

With the layered architectural type chosen the design was then refined into the architectural digram shown in Figure 5.3; Layers have been elaborated providing additional functional information of the layers, providing a number of sub components within each layer (applying the principle of modularity). The main designs being the introduction of the utility layer; providing application wide functionality such as logging, properties and common functionality shared over multiple layers. As well the change from the logic layer to the 'common network interface'. Furthermore given our incremental model and how on the modelling stage the architecture is reviewed, the architecture was re-factored to include the 'priority asynchronous' sub component.

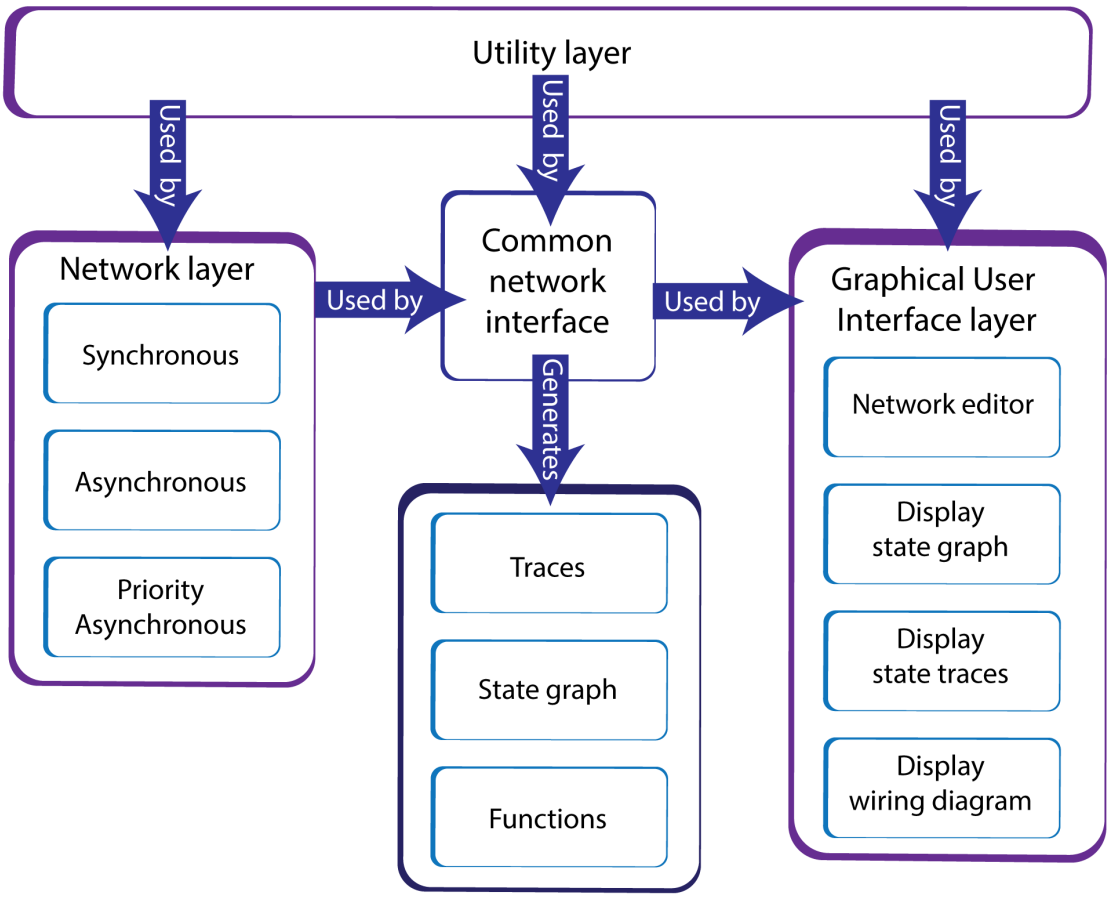


Figure 5.3: System architecture

With the 'common network interface' any network type can be used, the common network interfaces implementing the required logic (functionality) for the GUI. The decision made to use an interface to abstract networks from the GUI, in essence developing a 'dumb' GUI. That is to say that the GUI knows a little as possible about how the inner workings of the network logic. For example rather than calculate directional edges for a given network the

GUI simply requests a list of edges that it then draws. Creating more distinct components allowing for a concentration of the GUI to be aesthetics, learnability and usability. Given the GUI is the users first view of the system and that although we are taught 'not to judge a book by it's cover' in the case of UIs we often do, it is therefore essential to present a quality first view of the system. This links well with the following Section 5.2 where the process of designing the GUI it outlined.

5.2 Graphical User Interface (GUI) design

5.2.1 Introduction

The interface between the user and software the design of a GUI in founded in the principles of HCI. Systems with poor interfaces end up in users making mistakes, unable to access functionality making them feel the system hinders rather than helps them to achieve their goal [19, 42, 43, 47]. Whereas the aim of a well designed GUI is to provide a usable system founded in principles of Human-Computer-Interaction (HCI) such as the golden rules and heuristics [19, 42, 43].

5.2.2 Design process

To design the system's UI a series of low over high fidelity prototypes, the fidelity of a prototype is described as the accuracy of the prototype in detail and quality compared to that of the final system [52]. Essentially the higher the fidelity the closer we get to a complete system. A low fidelity prototype does not look like the final product but does provide the same functionality; rather than perform functions they often just represent them. For example rather than producing a graph it may just be a sketch an example graph containing its required components.

The Low-fidelity prototypes are useful given their simplicity and are quick to produce and therefore quick and simple to modify. Partially important during the early stages of design given ideas should be exploratory and not set in stone.

For each feature of the GUI we started with a number of low fidelity sketches of each design. Each sketch's concentration was of the design of the interactions rather than with the quality or styling. As is common practice sketches were drawn by hand allowing for the production of a large number of sketches in a short period time. Using a multitude of sketches concentrating on single features of the system we gain a broader range of ideas that sticking to a single idea.

Throw-away incremental paper prototype

On choosing the basic design of the system a number of sketch's where compiled into a visual representation of how the system might look (see Figure 5.4). So called incremental given the prototype was developed for each increment of our software process model. These set of sketches are named 'Throw-away' given on completion of the project the prototype can

be discarded. Unlike the final system which on completion will be maintained and operated. By using this low fidelity prototype one can find design problems before translating over to the final prototype. Given that if issues arise on a low fidelity prototype it can be quickly discarded and re-designed [19, 42].

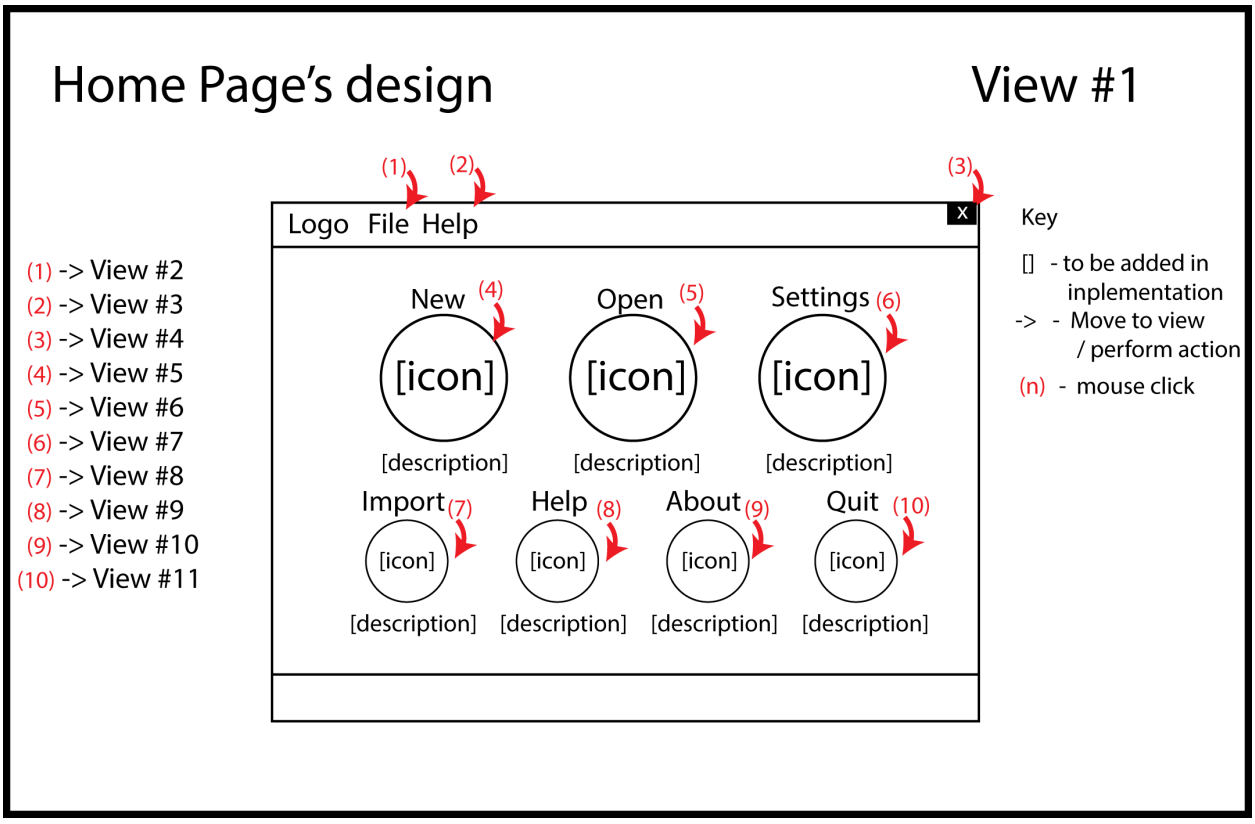


Figure 5.4: Example view of the low fidelity prototype

5.2.3 Design Rationale

The design rationale has been particularity important within HCI, providing both reflection and documentation on interface design taken place throughout the entire product life cycle[19]. Here we outline design decisions alongside the (HCI) principles the where based upon.

Learnability

To design a system to be used effectively; providing a user with the capabilities to get the most out of the interface we applied the principle of 'Learnability'. We split Learnability into Predictability, Synthesizability , Feedback, Familiarity , Consistency and viability.

Predictability By a common styling and interactions the user can predict future interactions in our interface. One key example of this is within out buttons where a common

styling is used. This allows a user to predict future interactions with the system assisting them in getting familiar with the interface.

Synthesizability Also known as the 'principle of honest' here we provide a user with a clear view of when the system changes state. The key place where this can be seen is where we disable the current view but do not hide it to show the user what view they are currently using.

Feedback we provide a user with clear visual feedback wherever possible. We have designed the system to provide buttons providing visual feedback on hover and on click.

Familiarity and Consistency As the majority of tools use navigation bars for navigation to adhere to the principle of familiarity we have included a navigation bar. We could have avoided this as our design allows a user to access all functionality from the buttons seen on screen. As mentioned in Predictability we have adhered to a consistent styling in our designs which also applies to the principles of Familiarity and Consistency.

Viability It is important for a user to be able to predict how the interface will function. Partially this has been achieved through the use of styling; A user can visually see a button will navigate to a specific functionality. However to make this clear to a user we have used a selection of Icons to represent functionality Providing a user with a visual representation of what is going to happen when they press the button. Each button is then additional provided with text to help explain to a user it'so purpose.

Chapter 6

Implementation

Referred to as the 'construction' in our incremental plan, the implementation refers to development of the designs and is the execution of the design plan.

6.1 Technologies

To construct a system we need the correct building blocks, that is to say the technologies used to implement the tool. Before construction can begin the possible technologies need to be compared and the most appropriate chosen. This section covers the technologies used and the process on which they where chosen.

6.1.1 Language

The choice of programming language is an important step of development, choosing a fit-for-purpose language enables the implementation of all the designed functionality [37]. The choice of language is so important because if you got it wrong and needed to change the re-factoring costs would be enormous, having to convert everything written to the new language. To avoid this potential risk research took place, the results of which are shown in Table 6.1.

Table 6.1: Quick language Comparison (✓ - particularly good, * - potential issues)

Language	C	C++	Java	Python	C#	Visual Basic
Existing knowledge	✓*	✓*	✓	✓*	×	✓
cross-platform	✓	✓	✓	✓	✓*	×
compiler/interpreter available at no cost	✓	✓	✓	✓	✓	×
object oriented	×	✓	✓	✓	✓	✓

The project was implemented in Java, given it's the support of third part graphing libraries and the majority of current tools are developed in Java. Further benefits include our current knowledge in the language and it's cross platform support.

6.1.2 GUI

Java provides three frameworks for developing a GUI; Abstract Windowing Toolkit (AWT), Swing and JavaFx.

Java's first GUI interface AWT was based on using native system components, however this was then replaced by their Swing GUI interface. Programs written in AWT are most commonly very old or built for complete portability (supporting very old systems).

Swing was introduced by Java to replace the AWT framework providing all the features of AWT (buttons, scrollbars, labels, etc.). I also provides additional higher-level components such as the tabbed panes and tree views. Swing also comes with the 'Look and Feel' interface used to provide native styles to components. Swing technology is the most commonly adopted framework with large numbers of 3rd party software.

JavaFx provides applications with additional GUI features such as web views, basic animation and audio. Designed to provide additional support for GUI design through it's use of Cascading Style Sheets (CSS) as would be used on a website.

Given Swing provides good platform support, mature set of 3rd party libraries and given we are building a standard desktop application, Swing is the framework used for the UI interface.

6.1.3 Graphing

To develop the graphing technologies we are presented with two choices, either develop the system from scratch or reuse an existing software. Both methods have benefits, software reuse increases dependability; given the system is already tried and tested, reduces processes risk; the costs/limitations of the existing system is already known and provides accelerated development; with development and validation times reduced the system production is speed up [47]. However by producing the components ourselves we would be in full control of features only creating that which is required, reducing unnecessary complexity . As well we would not be limited by the existing system's functionality, reducing the need to compromise on functionality.

The decision to follow the principles of software reuse was made, the core reasoning being that implementation of graphing technology would take a great deal of time; most likely spanning over the deadline for the project.

There are a number of different open source tools available in Java, the majority have been developed over numerous years tried and tested by numerous applications. These tools where researched to find the best suited tools for the job. This was performed before the choice of programming language to ensure the language chosen would have the required graphing libraries.

Java Universal Network/Graph Framework Written in Java The Java Universal Network/Graph Framework (JUNG)[53] provides a Java library for the model, analysis and visualisation of graph's or network's data. Supporting directed graphs, visualisations and a number of automatic layouts the framework makes it perfect for building network graphs. Where it is lacking is within the ability to change the arrow edges, not something that was directly built into the framework it would take time to change, therefore making it not appropriate for a wiring diagram (Given the need for negated edges) [53].

NGraph Written in C++ NGraph [57] provides a library for analysis of small to medium networks, given it was designed for educational purposes and therefore has a short learning curve. However it is a limited library lacking a number of tools and has not be optimised [57].

LEDA Another C++ library LEDA [29] has more comprehensive tools compared to NGraph however lacks a number of features provided by its main competion Boost Graph (see bellow) [29].

Boost Graph Boost Graph [58] is a comprehensive graphing libary providing all of the required functionality, however it does lack an easy to follow user guide and users have often complained of complier and documentation issues. For this reason we will not be using Boost Graph [58].

xGraph (JgraphX) Similar to JUNG JGraphX [33] written in Java, it provides a Java Library for the visualisations of networks. Where it trumps JUNG is in its simple styling methods to change the styles of edges, most importantly it easily support the modification of arrow edges making it well suited to a wiring diagram view. Sadly its lack of a more advanced layout engine makes it not so well suited for Network state graphs (Given it was primly build for the user to define the layout) [33].

Honourable mentions There are a number of libraries that we researched that we have not mentioned mainly because they are outdated libraries such libraries included Prefuse's Java library [55] and the GINY Java library [56] (Not maintained since 2005). We mention this given the number of Java based libraries available was a large part of the selection of the language we used (Java).

On comparing the pros and cons of the various frameworks we have chosen to use two different frameworks. Although this adds complexity and maintenance, to get the best framework for required functionality it is a worthwhile compromise.

The wiring diagram view we will be using JGraphX given it can provide us with the negative style edges we need for the graphs (via choice of arrow heads). The state graph we will be using the framework JUNG for use of its more advanced layout engine, given the large number of states displayed. Also given the state graph does not need different edge styles this negative attribute of JUNG will not be an issue.

6.1.4 Software evolution

When implementing software one must understand that change is inevitable in order for software to remain useful [47]. There are three reasons for evolution, to repair bugs/faults, to adapt to a new operating environment and to add or modify functionality.

Taking this into account we dedicated time to structuring the system to allow for such changes to be produced with ease, with main consequence the increase of time in development. However as suggested by studies by those such as Erlikh [22] it is suggested around

90% of software costs are evolution costs, it was well worth reducing these costs where possible.

A lot of the steps required to deal with the system's evolution have already taken place in the application of design principles within the architectural design, most fundamentally the decision to separate GUI and networks by an interface layer. The key here is to continue ahead with the designs, building in modularisation wherever possible, to ensure new components can be added with ease and existing ones modified with minimum effect to other components. How this was done is displayed in Figure 6.1, which shows the system has been designed for the addition of modules for new network, network functionality and new network visualisations.

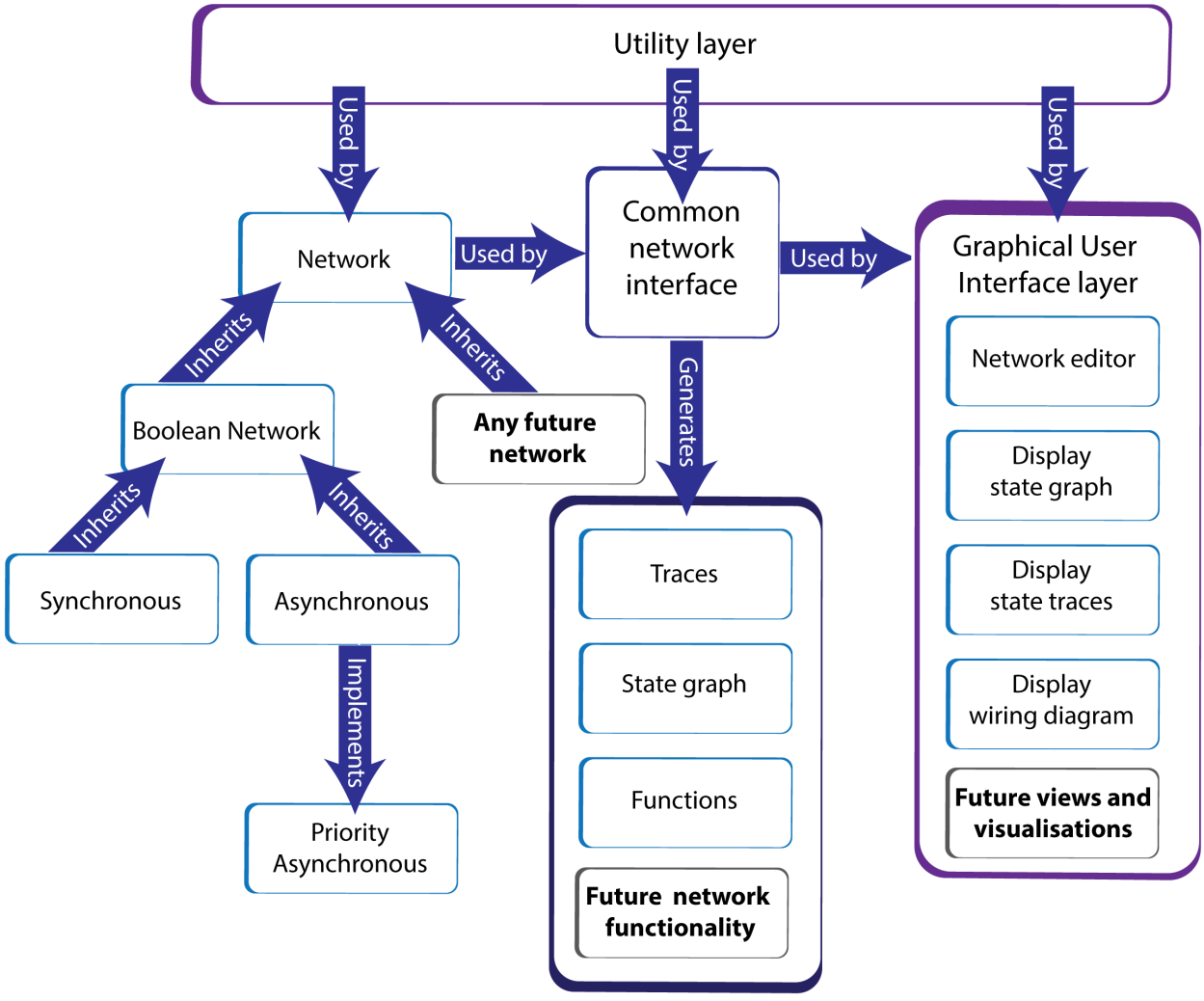


Figure 6.1: Diagram outlining the high level view of modularisation

6.2 Functionality waves

Figure 6.2 shows the key stages of the system's development, showing where each component was implemented. This separation helps to show the process performed to create the system as a whole, with functionality added in incremental stages. Starting with basic back-end in wave 1, moving to the basic network editor (for creation and modification of networks) in wave 2, then the more intermediary functionality of network visualisations in wave 3 and the final wave (wave 4) of development creating the more advanced features of priority classes and state clustering.

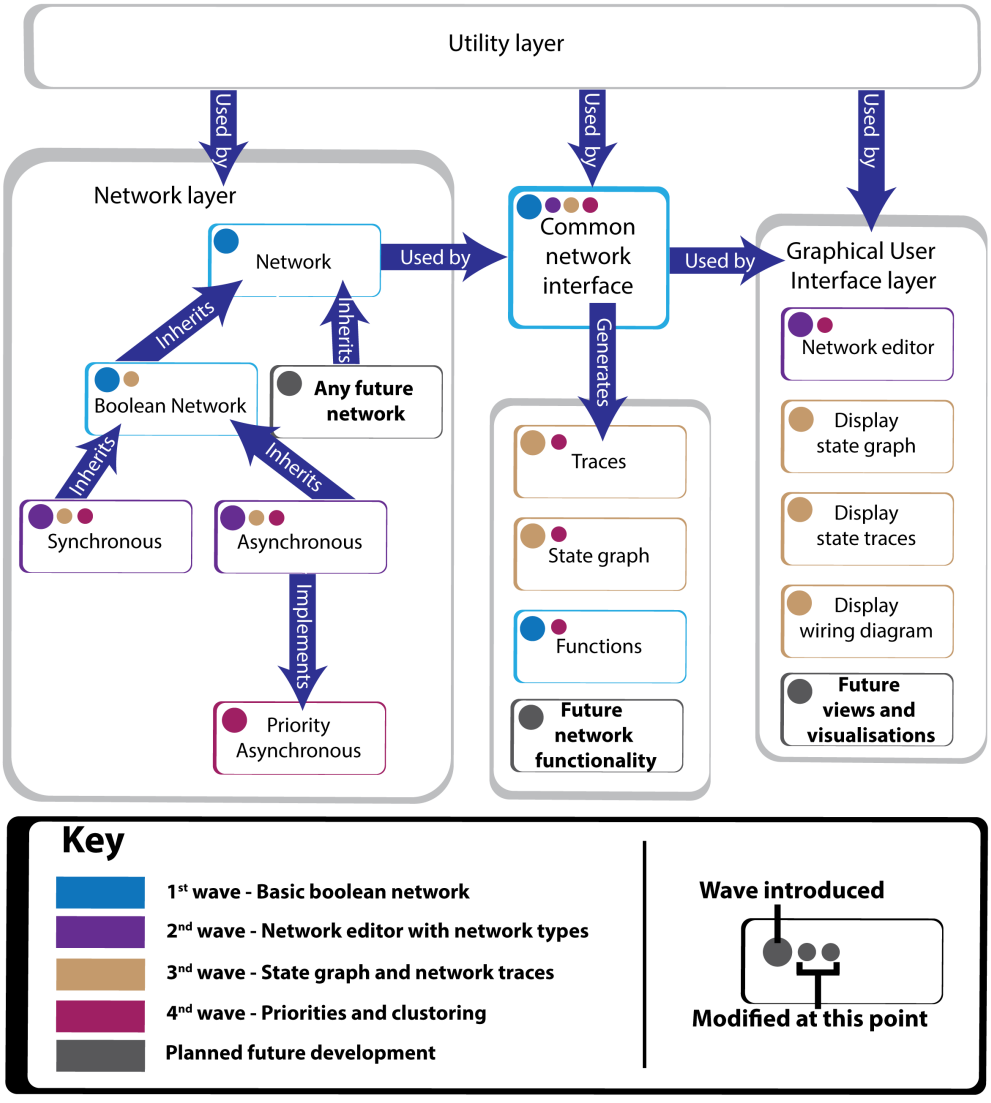


Figure 6.2: Diagram outlining the high level view of modularisation

6.3 Development of boolean network functionality

Named the '1st wave', implementation of the boolean network functionality alongside the common network interface set the foundation for the system.

6.3.1 Boolean network models

The first development that took place was the design of data models for boolean networks, here basic structures where created, using the initial research performed.

Take the example network provided in the research:

$$\begin{array}{ll} \textit{Entities:} & [\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}] \\ \textit{Functions :} & \begin{bmatrix} \text{A} & = & \text{A OR NOT C} \\ \text{B} & = & \text{A AND C} \\ \text{C} & = & \text{B} \end{bmatrix} \end{array}$$

Figure 6.3: Boolean network Example (same as 2.1)

The first thing we need is a representation of a boolean network, to do this a class `BooleanNetwork` which stores `BooleanFunction`s. A `BooleanFunction` being a representation of a networks entities and corresponding functions.

```
public abstract class BooleanNetwork{

        \\ list of functions representing entities and there functions
        private LinkedList<BooleanFunction> functions = new LinkedList<>();

        @XmlElement(name = "function")
        public LinkedList<BooleanFunction> getFunctions() {
                return functions;
        }

        public void setFunctions(LinkedList<BooleanFunction> functions) {
                this.functions = functions;
        }
}
```

Now a representation of a Boolean function is needed, for this the class `BooleanFunction` was generated. What we wanted was a way of representing a function so that a given entities function could be run with ease, efficiently. Therefore the concentration was on the evaluation of a given Boolean function.

Taking the example function: A = A OR NOT C , we investigated two different ways implementation techniques; String (Figure 6.4) or Boolean Tree (Figure 6.5) representations.

The string method has the advantage that no calculation is required on the creation of the function, the function is simply stored as it was in-putted. However each time the next state is required we apply the the steps shown in Figure 6.4 under 'Evaluation method'. Now given when calculating states the evaluation will be run many times (2^n *entities*), a different method is required.

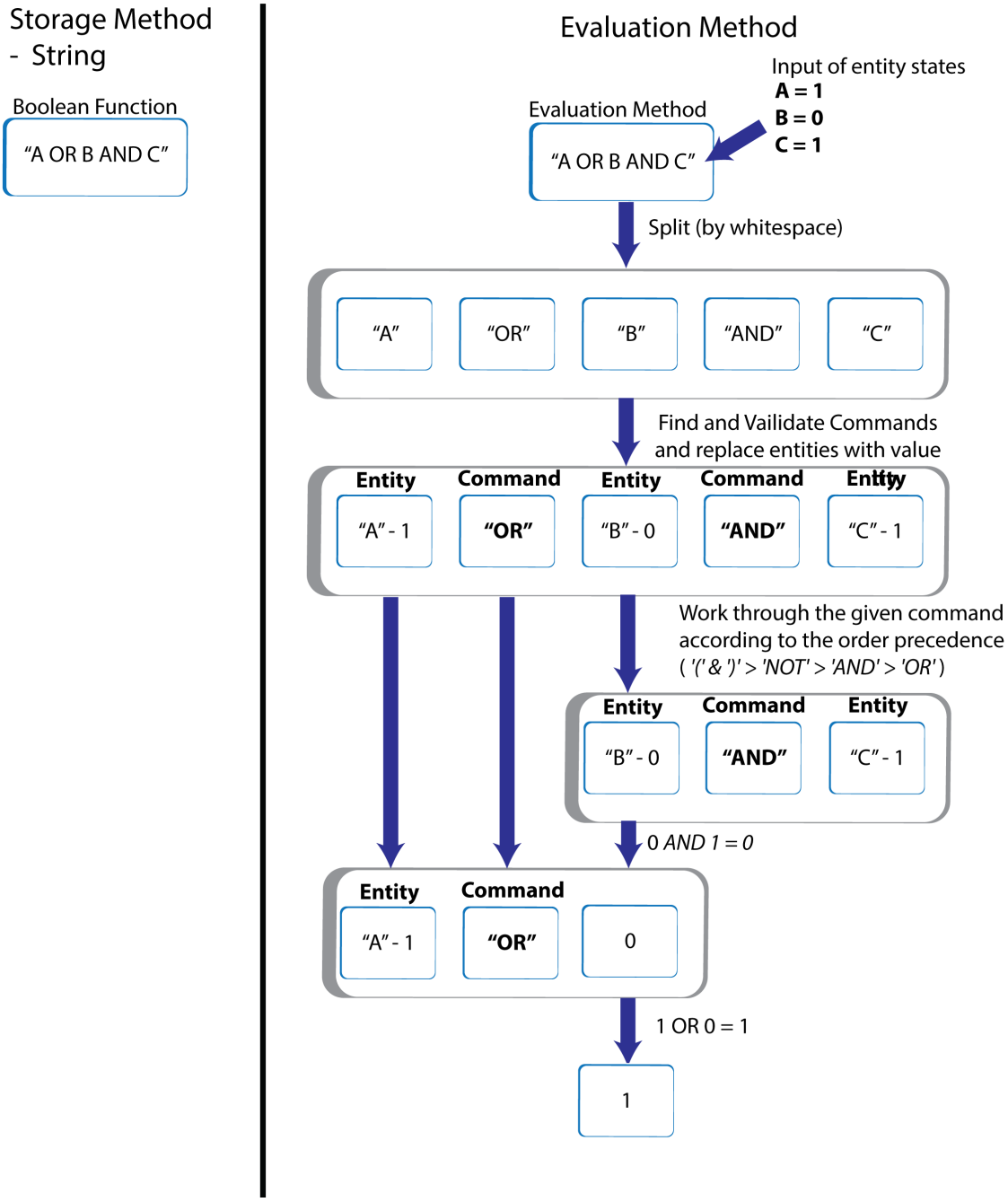


Figure 6.4: Diagram outlining storage and evaluation for string based storage

With the use of trees we can move the bulk of the calculation from evaluation to within the storage (see Figure 6.5). Due to the reduced work load on evaluation this method was implemented, with `BoolNode`.

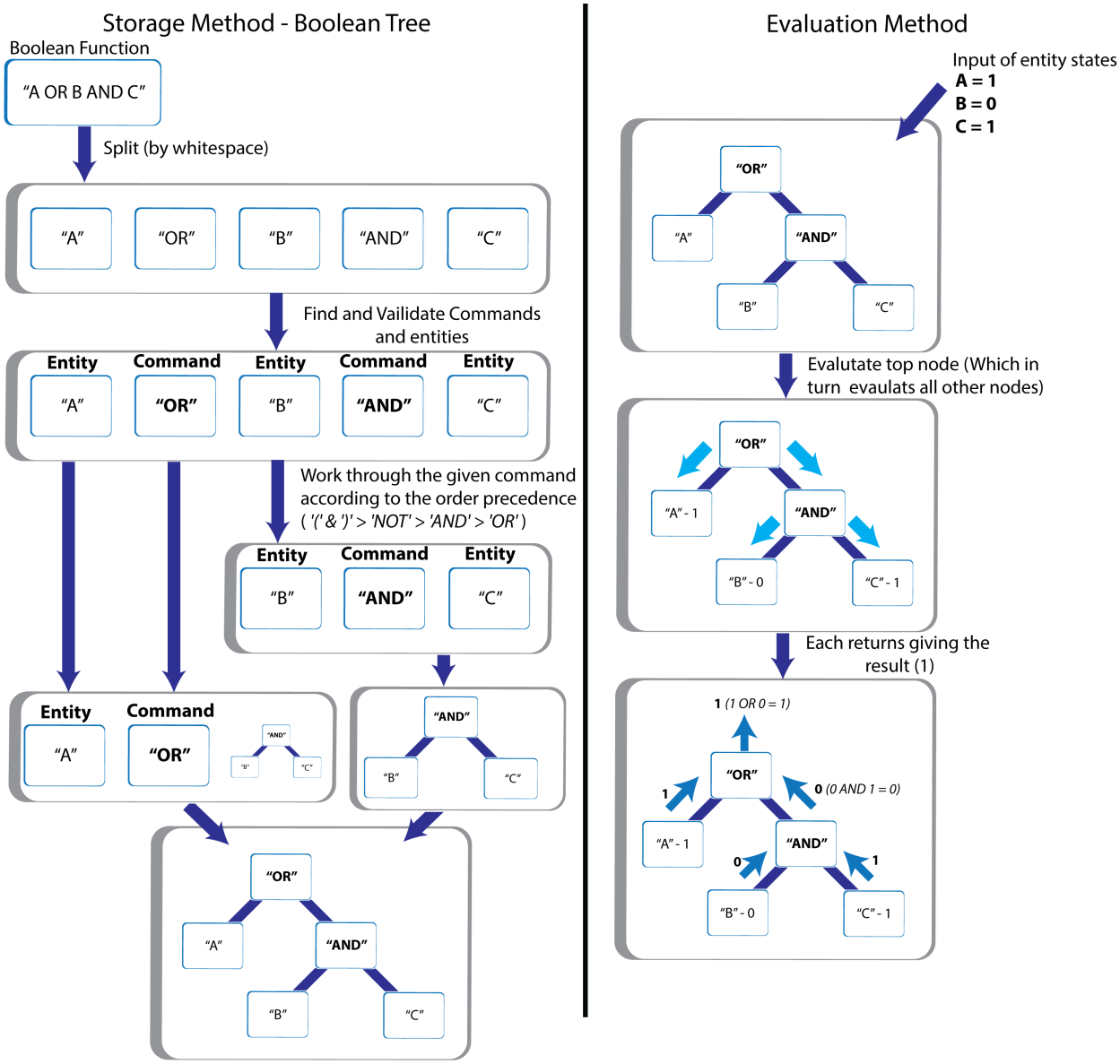


Figure 6.5: Diagram outlining storage and evaluation for Boolean function based storage

The `BoolNode` class is an abstract class on which all node types inherit, a Boolean Function will have one root `BoolNode`. The `BoolNode` class provides nodes with a string name and requires classes that inherit it to have an evaluate function. The `EntityBoolNode`, AndBoolNode, `OrBoolNode` and `NOTBoolNode` classes then all extend `BoolNode` providing Boolean logical functionality. This functionality is as follows:

- `EntityBoolNode` (Entity logic) - An entity node sets it's name to the entity name, it

returns the the current state of the entity (from it's name), provided with the network state.

- `AndBoolNode` (AND logic) - Consisting of a left and right node it evaluates it's left and right nodes returning the conjunction of both results.
- `OrBoolNode` (OR logic) - Consisting of a left and right node it evaluates it's left and right nodes returning the disjunction of both results.
- `NOTBoolNode` (NOT logic) - The not node has a single sub node, it evaluates to the negative result of that node's evaluation.

6.3.2 Implementing the common network interface

Figure 6.6 provides an incite into the implementation of the common network interface. The interface is created with an network identifier, invalid identifiers are rejected with only supported networks being accepted. The interface then provides an interface between the defined network type (through identifier) providing functionality for the GUI.

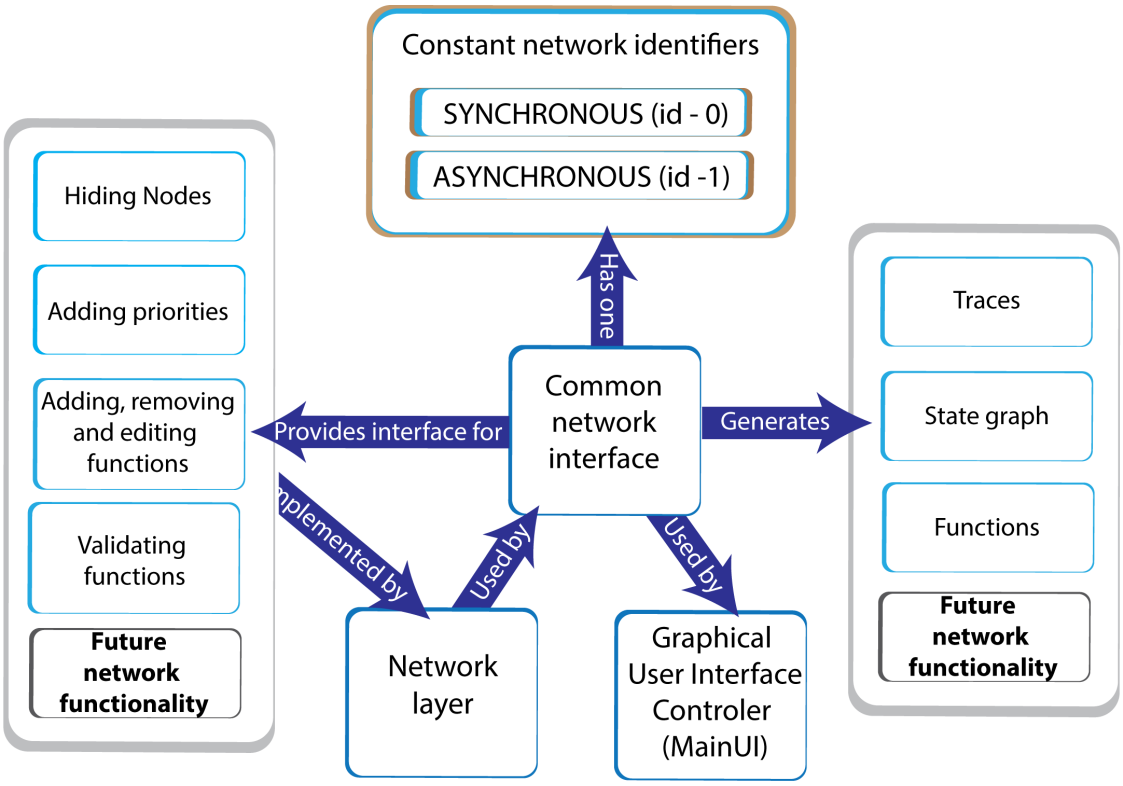


Figure 6.6: The common network interface

One of the aims here was to make the GUI require as little information as possible (remember the concepts of modularity). This lead to the interface being built so the GUI does not know what network it is dealing with, with the only time the GUI needing knowledge of Network types being in the creation of a new network. One example code snipped

shown below shows how the interface deals with removal of functions for the synchronous and asynchronous networks.

```
public void removeFunction(int position) throws Exception {
   switch (networkType) {
      case CommonNetworkInterface.SYNCHRONOUS:
      BooleanFunction oldSyncfunction =
          ((SynchronousBooleanNetwork) network).getFunctions()
              .get(position);
      SynchronousBooleanNetworkFactory.removeFunction(
          (SynchronousBooleanNetwork) network, oldSyncfunction, false
          );
      break;
      case CommonNetworkInterface.ASYNCHRONOUS:
      BooleanFunction oldAsyncfunction =
          ((AsynchronousBooleanNetwork) network).getFunctions()
              .get(position);
      AsynchronousBooleanNetworkFactory.removeFunction(
          (AsynchronousBooleanNetwork) network, oldAsyncfunction, false
          );
      break;
      case CommonNetworkInterface.MULTI_VALUED:
      throw new Exception("Unimplemented");
      default:
      throw new Exception("No such network exists");
   }
   stateGenerated = false;
   traceGenerated = false;
}
```

For this reason the resulting Trace and wiring diagram implementation was made a lot quicker to implement than anticipated. Given the complexity of the support libraries abstracting the networks allowed for a clearer vision of how to implement the required features using the JUNG or JgraphX libraries.

Extending Network models

The $2nd$ wave of development introduced the synchronous and asynchronous Boolean network models. This is the stage where states where introducing the separation of synchronous and asynchronous networks (Given there difference in states update schemes). Prompting the development of a representation of a network and entities state, achieved with the `BooleanNetworkState` and `BooleanNodeState` classes. A `BooleanNetworkState` representing a state of the network as a whole, for our example network in Figure 6.3 one state would be $[State_{\mathrm{A}}, State_{\mathrm{B}}, State_{\mathrm{C}}]$. With each entities state (e.g $State_{\mathrm{A}}$) being represented by `BooleanNodeState`.

With representation of a state, separate functions for synchronous and asynchronous was developed to find the next state (with methods outlined in research), stored in the

`SynchronousCommonUtil` and `SynchronousCommonUtil` classes (Utility as will be used by multiple classes).

Representations where then created of the networks, the `SynchronousBooleanNetwork` and `AsynchronousBooleanNetwork` classes which both extend `BooleanNetwork`. Each network having a state Network and state Trace, containing a representation of traces or of the state network.

This set the basis for the 3*rd* wave of development where the calculations for state and trace were introduced, through the development of the factory classes for the state network and trace representation of each network. The factories for `SynchronousBooleanNetwork` and `AsynchronousBooleanNetwork` generating states and traces using the corresponding updating scheme.

6.4 Implementing the GUI

6.4.1 Changes to technology from initial designs

The graphical user interface was developed within Intlij's form framework. However this was not always the case; originally the system was developed using the pure Swing layout. Although using Swing could have produced the same results we found the framework to provide the tools necessary to speed up the process dramatically. The two features we took advantage of was the preview and drag and drop functionality, allowing the quick design and view of a given layout without having to run the program as a whole.

An additional technology added to get the desired look, was the library 'seaglass'. Providing a number of clean GUI item styles, the package modifies the default looks provided by oracle to help increase the professional look of the system.

6.4.2 Icons

One key element to the GUI is the use of icons to assist the user with understanding of functionality. Again the principle of software reuse come up; should we create our own logos or use existing sources? In the end the choice was to use both; to assist with familiarity commonly used icons are important, however there are limitations to current icons particularly in regards to networking it is necessary to produce custom generated icons.

The use of Entypo [51] open source icons was used with the addition of bespoke logos generated with the Adobe suites (Illustrator® -Vector design and Fireworks® - Exporting to PNG) [1]. *Note: Fireworks® and Illustrator® are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries [1].*

button design Figure 6.7 shows the implemented buttons, built with two state; the normal state and the on hover state, which provides a user with feedback that the button is being hovered over.

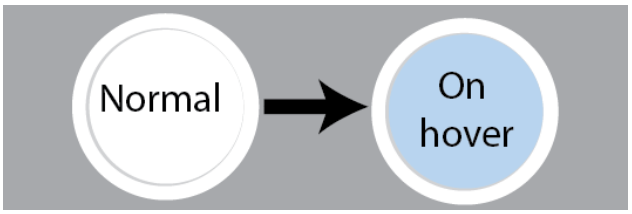


Figure 6.7: Button implementation

The process of then adding this to the code us simply implemented within the following code:

```
newButton.setIcon(new ImageIcon(getClass()
.getResource("/images/[logo-name].png")));

newButton.setRolloverIcon(new ImageIcon(getClass()
.getResource("/images/[logo-name]_RollOver.png")));
```

All Icons where developed as vectors and resized. the alternative would have been to develop pixel based icons (i.e JPEG), however resizing pixel based icons effects quality. Whereas Vectors can be resized with minimal quality issues and therefore was chosen to support future increasing in resolution.

6.4.3 Design of the main UI

The main UI was developed in the second wave of development to control the other GUI components, controlling the movement between functionalities. To understand the design of the `MainUI` one must first understand the principles and choices made in it's creation. The first such choice was the choice of control styles made between two principles Centralised control and Event-based control [47].

Centralised control uses a sub-component, given responsibility for the control of the other sub-components. This centralised control component starts/loads and stops/unloads other subsystems, possibly passing over control to that sub-system but will expect to be given back control [47].

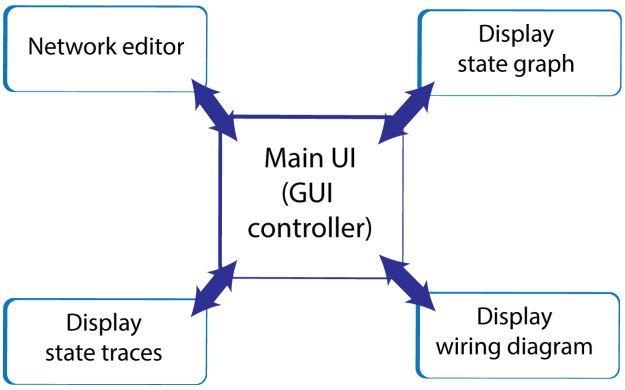


Figure 6.8: Central based control high-level design

Whereas in a event-based system, components run controlling themselves responding to externally generated events from other subcomponents or system environment [47].

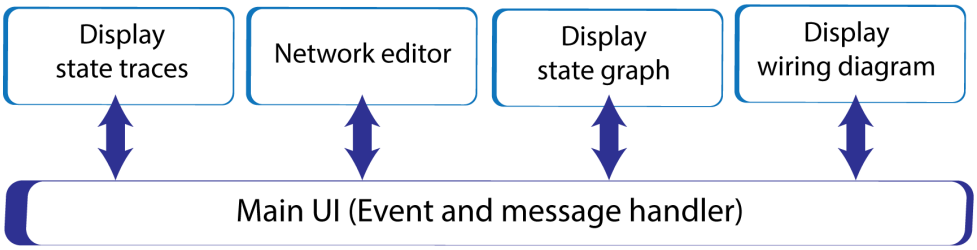


Figure 6.9: Event based control high-level design

The `MainUI` was designed with a centralised control seen in Figure 6.8 due to it's simpler design compared to an event-driven system, the lack of need for event-based systems complicity (network based interactions). Another factor was with the use of a sequential control scheme, it becomes easier to apply fault tolerance techniques.

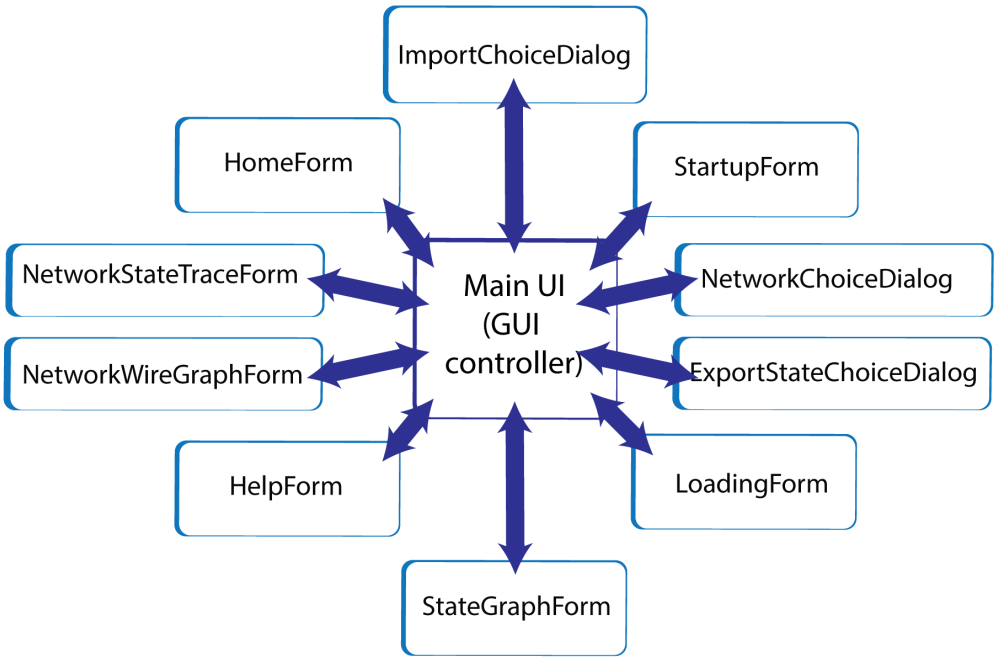


Figure 6.10: Central based control high-level design

6.4.4 The state based model

The `MainUI` class is designed to load, store and clean the state of the system. For this reason we chose to use a state based model, illustrated below in Figure 6.11, with a list of states provided in Apendix B by Figure B.1.

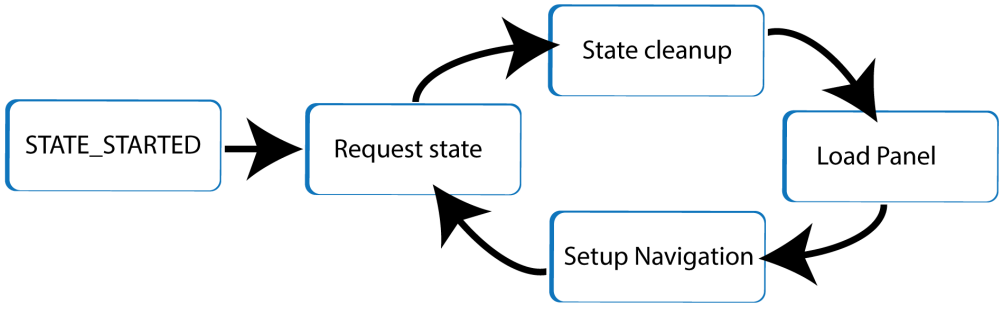


Figure 6.11: `MainUI` process of changing states

The `MainUI` accepts requests to change the state state and will run required clean-up for the previous state, load required panel for the state and set-up navigation (disable/enable menu items).

6.4.5 Editor

Taking place in the $2^{n}d$ wave the network editor proved to be one of the most challenging aspects of GUI development. Although fairly simple in nature, the need for feedback on the

network as the user changed entities and functions, required a number of listeners constantly checking the state of the functions as changes were made (see Figure 6.12 and (see Figure 6.13)). Additionally given the dynamic nature of the functions (can be added/removed), adding GUI components dynamically through the code proved difficult particularly while providing a consistent layout of the functions (see Figure 6.12).

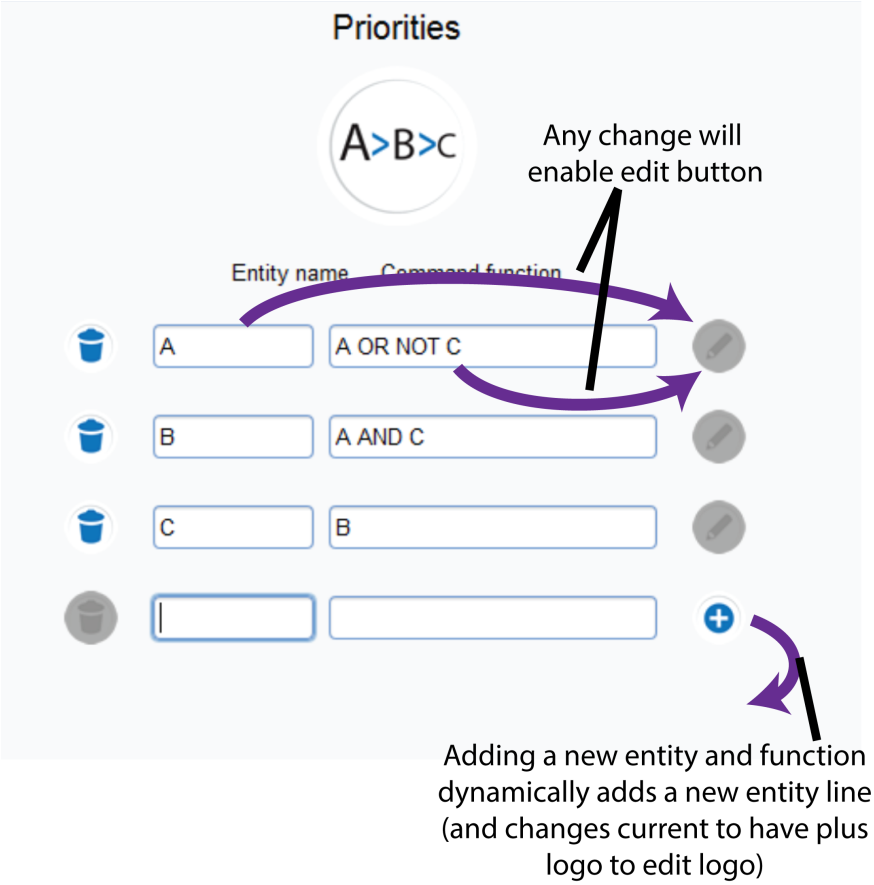


Figure 6.12: Annotated editor view

Another aspect that was added was the use of warning errors to tell the user when something was wrong with the function. The most common warning being for when a user has not yet defined a entity they use in a function, Figure 6.13 shows one such case.

One of the key aspect of this warning system is that the editor itself does not do the checking directly; it asks the `CommonNetworkInterface` if the given function is valid. This is important as in future development the editor will not need to be changed to accomidate new networks as this will be dealt with by the `CommonNetworkInterface`.

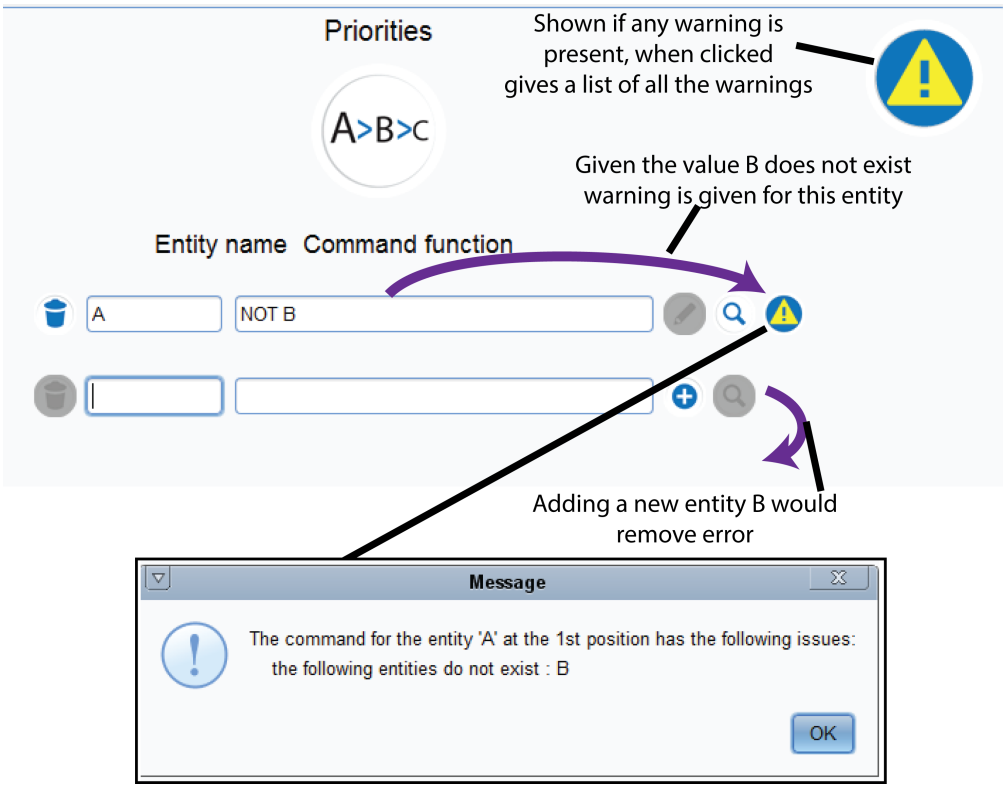


Figure 6.13: Annotated editor view showing warning case

6.4.6 Visualisations

Built in wave 3 creation of visualisations, involved the use of the JUNG and JGraphX libaries. As mentioned previously the process of abstracting networks from the GUI, lead to the implementation of the visualisations to be relatively straight forward.

To provide a template for interfacing with the JUNG and JGraph libraries the `JGraphXGraphPanel` and `JungGraphPanel` abstract classes where generated, both providing basic setup of the graphs. The interfaces both implement a `generateGraph` function which uses a `genNodes` and `genEdges` functions implemented by classes that extend them.

Then to generate a wiring diagram we use the `JGraphXNetworkGraph` which extends `JGraphXGraphPanel`, simply implementing `genNodes` and `genEdges` by requesting edges and nodes from the `CommonNetworkInterface` and adding them to the graph. With the exact same process done for generating the state graphs with the use of the `StateGraph` which extends `JungGraphPanel`.

6.4.7 Priorities and state clustering

Added in the final wave of implementation, the priorities and state clustering functionality was developed. Rather than develop priority based asynchronous networks as a completely separate component, it was integrated into the already existing asynchronous network (see Appendix B Section B.2). The priorities themselves are stored within the `CommonNetworkInterface`, which will check if a network is compatible before allowing them

to be added (so fo synchronous network it will not add them). The `CommonNetworkInterface` then generates states and traces with the priorities.

For the state clustering all functionality was added within the `CommonNetworkInterface`; `CommonNetworkInterface` generates states using the networks updating scheme and then hides and mergers nodes together, in the same processess outlined in research. The code snippet shows how the current list of edges is taken and reduced by hiding the selected entities/nodes.

```
private LinkedList<Pair<String>> hideNodes(
   LinkedList<Pair<String>> stateEdges) throws Exception {

   LinkedList<NetworkFunction> functions = getFunctions();

   // array for if a state is hiden or not
   Boolean[] isHidden = new Boolean[functions.size()];

   int newStateLength = 0;
   int i = 0;
   for (NetworkFunction function : functions){
      if (hiddenNodes.contains(function.getName())){
         isHidden[i] = true;
      }else{
         isHidden[i] = false;
         newStateLength++;
      }
      i++;
   }

   LinkedList<Pair<String>> hidenEdges = new LinkedList<>();
   for(Pair<String> Edge : stateEdges){
      char[] firstStateArray = Edge.getFirst()
         .replaceAll("\\s+","").toCharArray();
      char[] secondStateArray = Edge.getSecond()
        .replaceAll("\\s+","").toCharArray();

        // Quicker to use char array rather than appending
        char[] nextFirstStateArray = new char[newStateLength];
        char[] nextSecondStateArray = new char[newStateLength];

        i = 0;
        for (int j = 0; j < isHidden.length; j++) {
           if (!isHidden[j]){
              nextFirstStateArray[i] = (firstStateArray[j]);
              nextSecondStateArray[i] = (secondStateArray[j]);
              i++;
           }
        }
        hidenEdges.add(new Pair<>(new String(nextFirstStateArray),
            new String(nextSecondStateArray)));
   }

   // Cleanup linked list
   Set<Pair<String>> stateSet = new HashSet<>();
```

```
    stateSet.addAll(hidenEdges);
    hidenEdges.clear();
    hidenEdges.addAll(stateSet);

    return hidenEdges;
}
```

Chapter 7

Software testing

The purpose of our testing is two fold, first to demonstrate that software meets the requirements and second to discover faults within the software where the behaviour of the software is incorrect or undesirable [47]. These two purposes lead to two types of testing, validation testing where we test the system performs correctly reflecting the system's expected use and defect testing where tests are designed to expose defects with tests that often not reflecting how the system is used.

7.1 Static analysis

The first step of any testing process is with static testing. Here we inspected the code without running the program, skimming though the code to see if they can spot any problems. This in itself is not a valid testing strategy in itself however, static testing is important for finding mistake early on whilst the code is being developed often spotting bugs before the code is ever run. Also leading to higher quality as code is tested for understandability (if you cannot understand the code then it is re-write / additional comments added).

7.1.1 Test automation

The integrated set of tools used to support the test process is called the testing workbench [47]. We have used a set of tools that we use to automate testing to reduce the costs of testing. The automation testing workbench used within this project can be seen in Figure 7.1.

7.2 Unit testing

The support tool has been tested with the use of the JUnit [54] framework. JUnit is used to write repeatable tests, the key advantage being tests can be written and rerun as the project is being implemented to check additions do not effect previously written code [54]. A number of unit tests were written with JUnit running validation tests on the various components.

7.2.1 System testing

With the use of the AssertJ [15] Swing library to generate JUnit tests, all GUI tests were automated (AssertJ simulates user action on the gui) into a selection of functional system level tests (tests system as a whole rather an individual components), providing black box

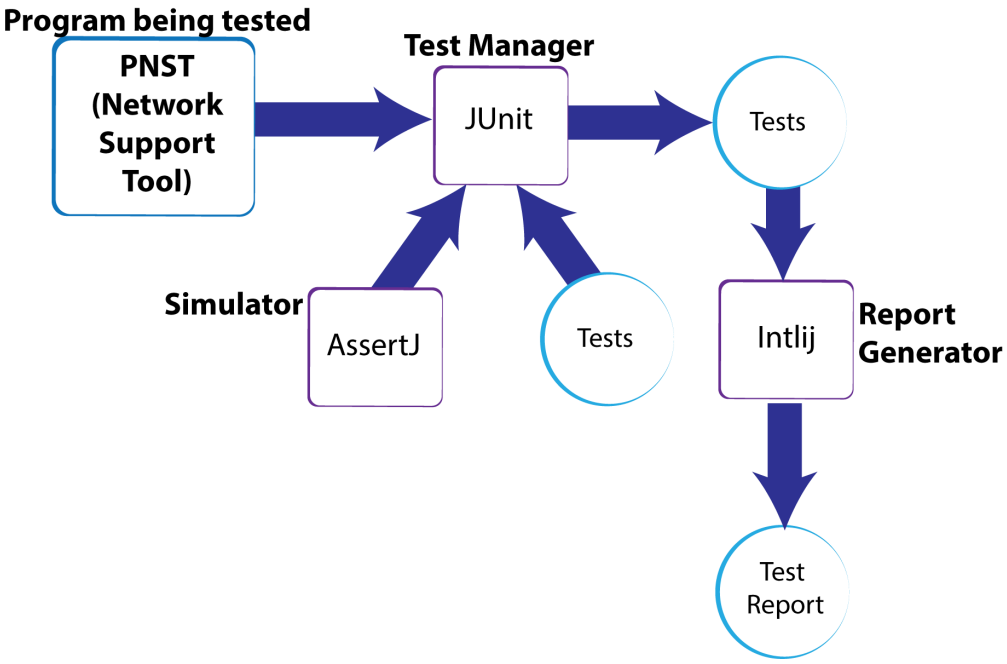


Figure 7.1: Testing workbench (automation architecture)

testing. These tests where observed manually by eye to ensure layouts were also correct for each view.

7.3 Test Results

Given the irritative way the tests have been implemented all of the test currently pass (see Appendix E SectionE.1). However it is worth noting that on each iteration a number of bugs were found in the code. A few of the bugs found are listed below:

- Additional nodes were added to the network graph which was only noticed on close expectation of the state graph visualisations. This was caused by nodes being added twice to the graph, which was caused by an additional space in the program so the same state ”1 0 1” was being added again as ”1 0 1 ”.

- During testing it was discovered that a user could move the edges. This modification of the graph should not be possible within the tool, the fix was a change to the 3rd party libraries setting. This does however point out the difficulties with software reuse as well as the importance to understand 3rd party libraries

- A major bug found early on in the system development was caught by unit testing was that synchronous traces would miss off the last state in the trace. Although a simple fix, it was a major bug given it gave the user inaccurate information. This bug in particular points out the importance for unit testing given this error was discovered directly after a re-factor of the code. If not for the automated tests, it is likely it would have been missed, only caught later in development.

- A bug caught by system testing was that users could see features that had not been implemented or were disabled in a particular view. Although not a major bug as it does not hinder the functionality of the system, it did effect the usability of the system as users may attempt to access features not available.

The tests were also run with coverage, showing us how much of the code was covered by the testing. With over 80% (see Appendix E Section E.2) coverage the majority of the code base was tested. Taking into account defensive code (code that cannot be run, designed to safeguard from errors from future development) and code pace-holders for future development, the coverage is at around 90%. Therefore our coverage is well within the commonly suggested code coverage levels of 80% [12, 31].

Chapter 8

Evaluation

8.1 Introduction

This chapter will cover the analysis and results of a number evaluation methods we have used. This chapter outlines whether the correct system was built. Covering the User feedback as well as the results from our expert evaluation.

8.2 User feedback - Questionnaire

Bellow is the evaluation of the questionnaire found in Appendix H, with it's answers found in Appendix I.

8.2.1 Reason for use

We chose the use of a questionnaire for user feedback given it provides user flexibility and is a much quicker process than that of an interview. particularly given these benefits help to get a larger capture group than interviews would provide. The downsides of questionnaires are that you often get users that 'abuse' the questionnaire answering silly responses and skew the data.

8.2.2 Demographics

Bellow in Figure 8.1 are the results for how competent users felt using software. We wanted to know this so that we could gage if inexperienced users could easily understand the system in addition to the more experienced users. As can be seen in Figure 8.1 we achieved this range we wanted from the questionnaire.

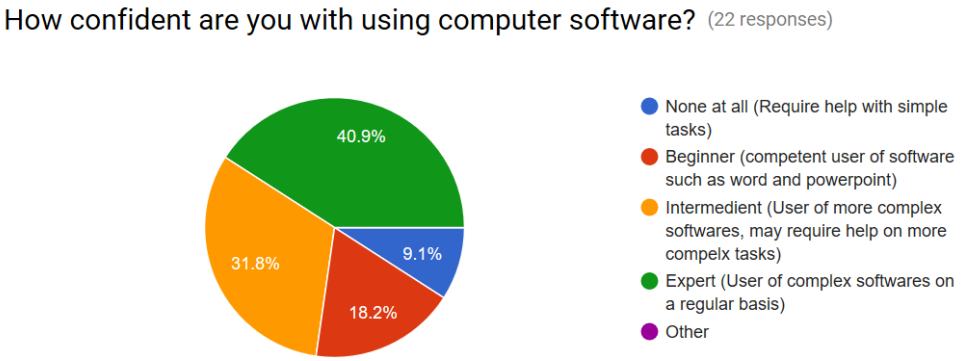


Figure 8.1: Users software experience

We would have liked to get opinions from those working with Boolean network tools. We did however get feedback from 3 users already familiar with Boolean networks in addition to two doctors familiar with using tools dealing with large sets of data.

8.2.3 Results and Analysis

Before being the analysis it is worth noting that within our results we had one outlier, which on closer inspection of name and email provided it was clear that this was a 'prank' response (We will not provide details of name/email for confidentiality reasons). This was unfortunate, but given our relatively large capture group (1/22), this did not effect the results and written feedback will be ignored (for example we believe 'Make it better' is not a valid response).

Demographics What we wanted from our questionnaire was opinions from a range of different backgrounds; although the majority of participants were computer scientists with an expert level of using software, we saw a large number of participants from other backgrounds. Giving us the capture group we wanted (range of experiences).

User Reaction We asked at the start and end of the questionnaire the users impression of the system and how they would rate the quality. On the first occasion the user was provided with an image of the home page of the application. Here we determined if a user's first impression was positive. On the other hand at the end of the questionnaire the user will have seen the demo and its functionality; here we wished to understand if the user's opinion improved after seeing what the tool could do.

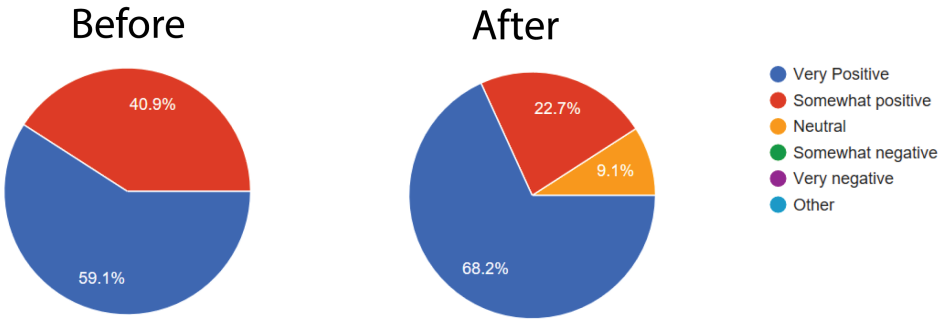


Figure 8.2: Users reaction of the system

In both case reactions were high with no negative responses. Interestingly we saw a slight decline to somewhat positive after users viewed the system, observing comments this was likely down to a number of features they would have like to have seen. However we did see the neutral responses move to very positive and somewhat positive. Again for quality responses were overwhelmingly positive, increasing further after the user's had view the demo video. These results give us a clear overall picture that users were impressed with the look and 'feel' of the system; seeing the prototype tool as a high quality application.

General feedback Users answered our general multiple choice questions with highly positive answers. Showing us that the users in general agreed with our design decisions (See Appendix for results) to quote one user 'I wouldn't change anything. It's clean, easy to navigate around and has clear descriptions of what each thing is'. What is more interesting is where users gave us constructive criticism, bellow we cover the key issues brought up by users and how we will/have deal with the issues raised:

Icons Asking '"Import file" intuitively looks like exit icon on other programs.' a user prompted us to think about redesigning our icon for export although similar icon style is commonly used for import.

Font A number of users asked about the fonts, asking 'I would use consistent fonts for title and other UI text' and 'Perhaps more constant fonts', on looking at they system a constant style would be preferable, because of this reason we changed the font of the title to match the rest of the fonts.

Menu bar Asking 'Do you really need help menu and help button?' raised the question if the menu bar is actually required. However given the menu bar is given for consistency we have chosen to keep it. Particularly given it does not take away functionality to a user but provides it for users whom prefer the menu bar style.

Text The questions 'too much text under each icon' and 'the size of the instructions is a bit small' show us we needed to cut back on the descriptions; making them concise for quick reading and to allow for the font size to be increased.

The questions 'Clearly label difference between import and open' & 'im not sure what the difference between opening and importing' show us that for beginner users the import function is perhaps less obvious than intended. To fix this the description could be changed to 'Import a file from another program' to make it clear of the action of the button.

Home Layout On user stated 'Settings I also think shouldn't be within the top action widget line, and import file fits much better considering they're all file orientated.', here they argue that the import button should be at the top. However we argue that the import feature is not a main feature of the tool and a user is more likely to want to change settings rather than import from another system.

One comment 'Would have all the icons the same size' shows us that the user did not comprehend that the smaller buttons were secondary functions. One possible solution as suggested by another user of the 'use of lines to separate the title, core and secondary functionalities on the page' would make the separation clearer to the user. s

General appearance A number of comments such as 'Less grey' , 'More colour' , 'Maybe add another colour' and Background colour makes some for the text hard to read.' shows us we need to consider the modification of background colours. Currently the choice of a grey background with black text allows the white buttons to stand out. However through this feedback it makes sense to make both text and buttons stand out.

On additional appearance suggestion was 'Less empty space on the sides'. However we feel that the space draws the users eye into the centre. Given the number of responses that said they liked the format we will not be changing it.

Editor layout Referring to the error messages one users suggests that it 'Needs to be bigger as a small icon in the corner will easily be missed'. On observing the interaction we agree with the user and plan to increase the button size and adding individual error buttons by each function where the error occurs.

General network views layout The majority of users liked the layout of the views however one user commented for 'Smaller buttons when viewing diagrams etc.'. However as this would not add to the usability of the program but instead take away we have chosen not to modify the button sizing.

Bugs spotted As is to be expected in prototypes one user spotted a bug in the program. The user spotted the 'typo "ersors" in the error message'. A quick and easy fixed this was repaired soon after the questionnaire.

8.2.4 Conclusions

To conclude overall the feedback was positive, those changes that will result from the feedback can be described as 'cosmetic fixes'. Given that there are no major problems with the tool we were pleased with this feedback from the tool. The overall user feedback was the software was of a high quality and all had positive views of the system.

8.3 Fulfilment of requirements

An overview list of all the requirements and how they were met can be found in Appendix J. Below we cover the key functionality implemented and how well each requirement was met.

Table 8.1: Requirements fulfilment table

Requirement identifier	Name	Discussion

F1	New network creation	The final tool gives the user the ability to create to create two networks the synchronous and the asynchronous boolean networks. An important part of the system is the considerations taken place for the future expansion of the tool through the design of the common network interface.
F5	Function manipulation	The final tool gives a user the ability to add, edit and remove additional entities and there corresponding functions. In addition to this the tool performs a number of tests on these functions given either errors or warnings to the use, such as the warning for when an non existing entity is refered to. This provides the user with feedback for why a given network is invalid. One improvement that could be made here would be a more specific error message for incorrect commands.
F6	Wiring diagrams	A user is provided with the ability to view a wiring diagram of from there given network, the system also allows for the manipulation of the layout of the wiring diagram. However the system does not save the layout of the wiring diagram and is generated each time it is loaded.
F7	State traces	The tool provides the ability to view traces of a valid given network, storing traces temperamentally and only reloaded on modification of the network. The traces are generated by the tool dependent on the updating scheme of the currently selected network. An possible improvement here would be to have a long term storage solution for these traces.
F8	State graph	The system will generate a state graph for a given network depending on the updating scheme of the given network. Again one improvement here would be to have a long term storage solution for the state transitions and layout.
F11	Hiding nodes	This requirement was met through the hide node function, whereby a user can hide entities on the state graph. When a node is hidden nodes are merged together reducing the complexity of the graph by reducing the nodes displayed to the user.
F12	Priority networks	This requirement was met through the network editor that provides the ability to add priorities to any asynchronous network.

8.4 Desirable qualities comparison table

The following tables (Table 8.2 & 8.3) gives an comparison of the desirable features between our tool and the competition. With Table 8.2 showing what we implemented and Table 8.3 showing what was not.

At first glance it appears that we did not implement a large number of the desirable features we perhaps should have. However given that a number of these features were complex bespoke features and our time scale this was to be expected. However all of the features that are listed where taken into account in the design of the system; planning for the future additional of such features.

Concentrating on what was achieved, it can be seen in Table 8.2 that we have implemented a tool that takes a number of the features distributed into the various tools and combined them in one tool. Most notability the use of aesthetics through our application of HCI, taking the usability from tool such as Boolesim and merging this with more advanced features of other tools such as Cytoscape and GINSim which have been added over many development years.

Support tool	Boolesim	GINSim	Cytoscape (Base)	Cytoscape with Plugins	PNST
Network Editor tool	✓	✓	✓	✓	✓
Generates Traces	×	✓	×	✓	✓
Generates State Graph	×	✓	×	✓	✓
synchronous	✓	✓	×	✓	✓
asynchronous	×	✓	×	✓	✓
priority classes	×	✓	×	✓	✓
Hiding Nodes	×	✓	×	✓	✓
interchangeable graph layout	✓(manual)	✓(manual)	✓(auto)	✓	✓
Export	✓	✓	✓	✓	✓
Aesthetics	✓	×	×	×	✓
Import	✓	✓	✓	✓	×

Table 8.2: Desirable features implemented comparison table

Support tool	Boolesim	GINSim	Cytoscape (Base)	Cytoscape with Plugins	PNST
Finds attractors	×	✓	×	✓	×
Multivalued	×	✓	×	✓	×
Time series	✓	✓	×	×	×
Visual simulation of state change	✓	✓	×	×	×
external public database support	×	×	✓	✓	×
custom graph styles	×	✓	✓	✓	×
network filter	×	×	✓	✓	×
Finding Clusters (sub networks and highly interconnected regions)	×	×	×	✓	×
Plugin/App support	×	×	✓	✓	×

Table 8.3: Desirable features not implemented comparison table

Chapter 9

Conclusions

9.1 The solution

This project developed a support tool provides capability for network creation and modification, supporting the two core Boolean network types (Synchronous and Asynchronous). With the addition of the creation visualisations of a networks dynamic through wiring diagrams, state traces and state graphs.

The project went further developing strategies to cope with the state based problem and reduce the complexity it brings. The first strategy we use to reduce a networks complexity is to reduce it's size by hiding specific entities. This method is similar to model reduction [7, 38, 60], however rather than reduce the size of the model itself we reduce the size of the state graph visualisation; we called this process 'state clustering'.

The second approach is the use of priority classes [7, 14, 23], where trajectories produced by an asynchronous updating scheme are reduced by assigning priorities. Here trajectories of lower priorities are ignored over higher ones. One particular example from [23] shows the successful application of priority classes to the mammalian cell cycle.

With the use of state clustering, reducing the number of states displayed visually to a user, coupled with the use of priorities reducing state connections we dramatically reduce the complexity of network visualisations. Therefore creating a solution providing visualisations tackling the state based explosion problem.

9.2 Satisfaction of aims and objectives

Our aim was 'To design and develop a support tool for creating and visualising the behaviours of Boolean networks'. This aim was then further split into a list of objectives, we will refer to these objectives by their number.

Objective 1 *To identify and evaluate a selection of at least 3 Boolean network examples.*

This research was undertaken, findings that can be seen in Section 2.1, most notably included the networks make-up as well as their dynamics. A number of networks where produced and identifies and can be seen in Appendix D.

Objective 2 *To evaluate existing Boolean network modelling support tools and produce a summary report.*

Evaluation was performed and the resulting report is show in Section 2.2. The review outlines three tools Boolesim, GINSim and Cytoscape, most importantly the report contains the desirable features that each tool has. The key development from this research being that current tools with lack the application usability principles (GINSim & Cytoscape) or are bespoke on off tools lacking in functionality (Boolesim).

Objective 3 *To identify techniques for visualising Boolean networks.*

Our research identifies wiring diagrams, state trace diagrams and stage graphs as key visualisations (see Section 2.1). Each visualisation was identified with techniques to generate them which where then applied within the implementation of the system.

Objective 4 *To identify a technique(s) for tackling the state space explosion problem.*

Section 2.1 covers two methods to tackle with the state based explosion problem: priority-based asynchronous network and state clustering via hiding of entities. Again the research was done well allowing the techniques discovered to be applied via the implementation of the system.

Objective 5 *To identify a list of requirements for a Boolean network support tool.*

As shown in Chapter 4, requirements where developed for the system. These were created with the use of the existing system research, as well as the research performed on Boolean networks and their visualisations.

Objective 6 *To develop a Boolean network support tool prototype that incorporates network creation and techniques identified from Objectives 3 and 4.*

Both low and high fidelity prototypes where produced, with a selection of screen-shots of the final system (high fidelity prototype) provided in Appendix F and an user-guide of the system in Appendix G. The tool was developed within schedule allowing for both techniques identified in Objective 4 to be implemented and all requirements were met.

Objective 7 *To evaluate the Boolean network support tool prototype and produce a summary report evaluating each feature within the tool.*

User evaluation was completed with a user questionnaire, testing and our reflective thoughts such as the design rational with the Design section. The usability and quality of the system was obtained from the questionnaire, where users rated the tool highly showing the system usability. However given the user lack of knowledge of boolean networks this does not show the reliability of the system. The testing shows the system to be reliable and meets the functionality required by the tool.

To conclude our overall aim was completed; a support tool was designed and developed which included the necessary features to visualise the behaviour of Boolean networks.

9.3 Personal Development

Throughout the process of developing this dissertation the author has seen a huge amount of personal develop. Such as:

- Research skills (and knowledge of BNs). Research skills is an area where I was lacking before starting, with only a basic knowledge into the area. Through need to read and understand academic papers my experience in research has dramatically increased.
- Vector design; On starting the project my ability to create vector images was questionable often relying on default button faces. Though this project I developed the skills to create professional looking vector images for use within the project. Personally I see this as an integral skill to gave in GUI design given it massively improves the appearance and therefore aesthetic quality of a UI.
- Graphical user interface design. Surprisingly (given Newcastle's Java based syllabus) this was the first time creating a comprehensive GUI using Java's swing interface. Form developing the tool I have become confident in developing UI's in the framework to a good standard.
- Re-factoring, Although a skill often used by myself this was something I learned a great deal about. For example with a code base of 3000+ lines of code early in the project I chose to refactor the whole system essentially starting again. Although this may sound counter productive hover the resulting system lead to future developed being much easier.
- Testing. Although as skill learned through my time at uni, implementing a test framework for a relatively large code base has taught me the skills required to develop a framework that can be used to check a product on each incremental stage of development.
- Full system design. Developing the tool has given be the insides into designing and build a full scale system; most importantly the design of a system architecture

9.4 Challenges

- **Research** One key challenge faced was in the initial research into Boolean networks, given all the information is contained within papers written on Boolean networks. Given this was the first time dealing solely with academic papers, it was a challenge to get to grips with the methods required to do academic research.

- **Willingness to start again (re-factoring)** - Although with years of experience large reactors to code essentially acting as rebuilding the system from scratch ,it is still takes a lot of confidence to do (particularly when working with a large code base).
- **Layouts of dynamic elements** - Relating to the implementation of the GUI in particular with that of the function editor, dynamically adding elements turned out to be quite a challenge Given Swing's layout engine would often resize elements on adding new elements the challenge was in in-keeping to a constant layout.
- **System size** - Given the number of features the system ended up supporting the system became much larger than anticipated. With a code base of 7000+ lines of source code (without comments or empty lines) in addition to a number of 3rd party libraries, it became a challenge to manage. However given the well designed architecture and modularisation of the system this provided less of a challenge, given designs such as the common network interface allowed for new features to be added with ease.
- **Priorities and state clustering** Both features for state space explosion proved to be difficult. The majority of this difficulty was faced at the research stage, given a small number of papers within the area as well as the understanding of how each method worked. However with the understanding of how each technique worked it was much easier to implement given our research already provided the method.

9.5 Future Work

The future of the tools can take many forms, because of the various areas of current and future research into the topic. Below is an outline of a few the possibilities:

- Additional HCI based support such as increased properties to allow for customer's to chose their own styles and settings. This would involve a redactor of the settings to match the button style implementation in the rest of the tool (rather than using the typical pop-up approach) for changing settings.
- Support of importing and exporting a multitude of additional tools such as Cytoscape and GinSim [59]. The challenge here would be the modification of their files to a simpler layout or the modification of our own network files to support additional functionality. For example GinSim outputs nodes styles whereas our tool currently only supports the one default style.
- Support of multi-valued networks and Context based networks; as discussed in research a clear vision for the future of the tool is the support of MVNs. In additional to supporting Context based networks.
- Clearer bespoke help for users. for each view of the system it would provide help directly for that page. This would be further advanced with the usage of video guides alongside text descriptions.

Some more challenging but worthwhile features the tool the tool could implement would be:

- Modular Boolean networks; conjoining of Boolean networks to represent an overall interaction [41]. Further dealing with the state based explosion problem.
- Supporting new research areas; as new topics arise we can support research by adding the necessary tools to perform the research. Given the modular approach to the tool this would be relatively easy to implement.
- One difficult but worthwhile feature would be the addition of database store support. Many networks are stored in large databases which could be imported into the tool. An example where this is used is in to tool Cytoscape which uses these data stores to load very large networks.
- Advanced analysis of the networks, much like the Cytoscape tool implements statistical analysis of networks. This could include numbers relating to Isolated nodes, Network dynamiter, Shortest paths etc.

Bibliography

[1] Adobe. Icons and web logo guidelines — adobe. `http://www.adobe.com/uk/legal/permissions/icons-web-logos.html`.

[2] T. Akutsu, S. Kuhara, O. Maruyama, and S. Miyano. Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. In *SODA*, volume 98, pages 695–702. Citeseer, 1998.

[3] T. Anderson. *Fault tolerance, principles and practice*. Prentice/Hall International, Englewood Cliffs, N.J, 1981.

[4] F. Ay, F. Xu, and T. Kahveci. Scalable steady state analysis of boolean biological regulatory networks. *PloS one*, 4(12):e7992, 2009.

[5] R. Banks. Qualitatively modelling genetic regulatory networks: Petri net techniques and tools. Ph. D. Dissertation, School of Computing Science, University of Newcastle upon Tyne, 2009., 2009.

[6] R. Banks and L. J. Steggles. An abstraction theory for qualitative models of biological systems. *Theoretical Computer Science*, 431:207–218, 2012.

[7] D. Bérenguier, C. Chaouiya, P. T. Monteiro, A. Naldi, E. Remy, D. Thieffry, and L. Tichit. Dynamical modeling and analysis of large cellular regulatory networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 23(2):025114, 2013.

[8] M. Bock, T. Scharp, C. Talnikar, and E. Klipp. Boolesim: an interactive boolean network simulator. *Bioinformatics*, 30(1):131, 2013.

[9] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[10] I. Bray. *An introduction to requirements engineering*. Addison-Wesley, Harlow, 2002.

[11] D. Budgen. *Software design*. Addison-Wesley, Harlow, England New York, 2003.

[12] bullseye. Minimum acceptable code coverage. `http://www.bullseye.com/minimum.html`, 2013.

[13] M. A. L. Caetano and T. Yoneyama. Boolean network representation of contagion dynamics during a financial crisis. *Physica A: Statistical Mechanics and its Applications*, 417:1–6, 2015.

[14] C. Chaouiya, A. Naldi, and D. Thieffry. Logical modelling of gene regulatory networks with ginsim. *Bacterial Molecular Networks: Methods and Protocols*, pages 463–479, 2012.

[15] J. Costigliola. Assertj / fluent assertions for java. `https://joel-costigliola.github.io/assertj/`, 2017.

[16] A. Dennis. *Systems analysis and design*. Wiley, Chichester, 2010.

[17] E. W. Dijkstra. *On the Role of Scientific Thought*, pages 60–66. Springer New York, New York, NY, 1982.

[18] E. W. "Dijkstra, O.-J. Dahl, and C. A. R. Hoare. *Structured programming*. Academic Press Ltd., 1972.

[19] A. Dix. *Human-computer interaction*. Pearson/Prentice-Hall, Harlow, England New York, 2004.

[20] B. Drossel, T. Mihaljev, and F. Greil. Number and length of attractors in a critical kauffman model with connectivity one. *Physical Review Letters*, 94(8):088701, 2005.

[21] H. Ebadi, M. Saeedian, M. Ausloos, and G. Jafari. Effect of memory in non-markovian boolean networks illustrated with a case study: A cell cycling process. *EPL (Europhysics Letters)*, 116(3):30004, 2016.

[22] L. Erlikh. Leveraging legacy system dollars for e-business. *IT professional*, 2(3):17–23, 2000.

[23] A. Fauré, A. Naldi, C. Chaouiya, and D. Thieffry. Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 22(14):e124–e131, 2006.

[24] V. Filkov. Identifying gene regulatory networks from gene expression data. In *Handbook of Computational Molecular Biology*, pages 27–1. Chapman and Hall/CRC, 2005.

[25] M. Flöttmann, T. Scharp, and E. Klipp. A stochastic model of epigenetic dynamics in somatic cell reprogramming. *Frontiers in physiology*, 3:216, 2012.

[26] C. Floyd. A systematic look at prototyping. In *Approaches to prototyping*, pages 1–18. Springer, 1984.

[27] D. Flynn. *Information systems requirements : determination and analysis*. McGraw Hill, London u.a, 1998.

[28] C. Gershenson. Introduction to random boolean networks. *arXiv preprint nlin/0408006*, 2004.

[29] A. S. S. GmbH and h. A. Kirchmeyer. Algorithmic solutions software gmbh. `http://www.algorithmic-solutions.com/leda/ledak/index.htm`, 2015.

[30] S. Gupta, S. S. Bisht, R. Kukreti, S. Jain, and S. K. Brahmachari. Boolean network analysis of a neurotransmitter signaling pathway. *Journal of theoretical biology*, 244(3):463–469, 2007.

[31] IBM. Ibm knowledge center. `https://www.ibm.com/support/knowledgecenter/SS4QVT_8.5.1/com.ibm.debug.pdt.codecoverage.doc/topics/tccsetac.html`.

[32] K. Inoue. Logic programming for boolean networks. In *IJCAI*, 2011.

[33] jgraph. jgraph/jgraphx. `https://github.com/jgraph/jgraphx`.

[34] S. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437 – 467, 1969.

[35] S. Kauffman, C. Peterson, B. Samuelsson, and C. Troein. Random boolean network models and the yeast transcriptional network. *Proceedings of the National Academy of Sciences*, 100(25):14796–14799, 2003.

[36] H. Lähdesmäki, I. Shmulevich, and O. Yli-Harja. On learning gene regulatory networks under the boolean network model. *Machine learning*, 52(1):147–167, 2003.

[37] Microsoft. Choosing a programming language. `https://msdn.microsoft.com/en-us/library/cc168615.aspx`.

[38] A. Naldi, E. Remy, D. Thieffry, and C. Chaouiya. Dynamically consistent reduction of logical regulatory graphs. *Theoretical Computer Science*, 412(21):2207–2218, 2011.

[39] W. Newman. *Interactive system design*. Addison-Wesley, Wokingham, Eng. Reading, Mass, 1995.

[40] E. Nier, J. Yang, T. Yorulmazer, and A. Alentorn. Network models and financial stability. *Journal of Economic Dynamics and Control*, 31(6):2033–2060, 2007.

[41] R. Poblanno-Balp and C. Gershenson. Modular random boolean networks. *Artificial Life*, 17:331–351, 2010.

[42] J. Preece. *Interaction design : beyond human-computer interaction*. John Wiley & Sons Ltd, Chichester, West Sussex, 2015.

[43] R. Pressman. *Software engineering : a practitioner's approach*. McGraw-Hill Higher Education, New York, 2010.

[44] W. W. Royce et al. Managing the development of large software systems. In *proceedings of IEEE WESCON*, volume 26, pages 1–9. Los Angeles, 1970.

[45] T. Schlitt and A. Brazma. Current approaches to gene regulatory network modelling. *BMC Bioinformatics*, 8(6):S9, 2007.

[46] I. Shmulevich. *Probabilistic boolean networks : the modeling and control of gene regulatory networks*. Society for Industrial and Applied Mathematics, Philadelphia, 2010.

[47] I. Sommerville. *Software engineering*. Addison-Wesley, Harlow, England New York, 2007.

[48] L. J. Steggles. Abstracting asynchronous multi-valued networks. *Sci. Ann. Comp. Sci.*, 21(2):249–282, 2011.

[49] Z. Szallasi and S. Liang. Modeling the normal and neoplastic cell cycle with realistic boolean genetic networks: Their application for understanding carcinogenesis and assessing therapeutic strategies. In *Pac. Symp. Biocomput*, volume 3, pages 66–76, 1998.

[50] R. Thomas. Boolean formalization of genetic control circuits. *Journal of theoretical biology*, 42(3):563–585, 1973.

[51] [Unknown]. Entypo. `http://www.entypo.com/`.

[52] [Unknown]. fidelity meaning in the cambridge english dictionary. `http://dictionary.cambridge.org/dictionary/english/fidelity`.

[53] [Unknown]. Jung - java universal network/graph framework. `http://jung.sourceforge.net/`.

[54] [Unknown]. Junit - about. `http://junit.org/`.

[55] [Unknown]. prefuse — interactive information visualization toolkit. `http://prefuse.org/`.

[56] [Unknown]. Giny. `http://csbi.sourceforge.net/`, 2005.

[57] [Unknown]. Ngraph: a simple netowrk graph library in c++. `http://math.nist.gov/` $\overleftarrow{R}$ `Pozo/ngraph/ngraph_index.html`, 2012.

[58] [Unknown]. The boost graph library - 1.64.0. `http://www.boost.org/doc/libs/1_64_0/libs/graph/doc/`, 2017.

[59] [Unknown]. Ginsim — qualitative analysis of regulatory networks: a computational tool based on a discrete formal framework. `http://ginsim.org/`, 2017.

[60] A. Veliz-Cuba. Reduction of boolean network models. *Journal of theoretical biology*, 289:167–172, 2011.

[61] X. Wen, S. Fuhrman, G. S. Michaels, D. B. Carr, S. Smith, J. L. Barker, and R. Somogyi. Large-scale temporal gene expression mapping of central nervous system development. *Proceedings of the National Academy of Sciences*, 95(1):334–339, 1998.

[62] Y. Zhu, F. Yang, and W. Ye. Financial contagion behavior analysis based on complex network approach. *Annals of Operations Research*, pages 1–19, 2016.

Appendix A

Tables and Figures

A.1 List of Tables

A.2 List of Figures

Code Snippets

Given the size of the code base (in excess of 7000 lines of source code), not all the source code for the system is provided in this Appendix. This appendix does however contain code snippets refereed to within the dissertation.

B.1 Possible states snippet

```
 /* The possible states available*/
// State where program has been first started
private static final int STATE_STARTED          = 0;
// Initial loading state
private static final int STATE_STARTUP          = 1;
// Home state
private static final int STATE_HOME             = 2;
// State for creating new network
private static final int STATE_NETWORK_CREATION = 3;
// Displaying network graph
private static final int STATE_WIRE_GRAPH       = 4;
// Displaying trace graph
private static final int STATE_TRACE            = 5;
// Displaying state graph
private static final int STATE_STATE_GRAPH      = 6;
// Displaying about information
private static final int STATE_ABOUT            = 7;
// Displaying tutorial pages
private static final int STATE_TUTORIALS        = 8;
// Displaying manual page
private static final int STATE_MANUAL           = 9;
// Loading state
private static final int STATE_LOAD             = 10;
// Used for refresh
private static final int STATE_REFRESH          = 11;
```

Figure B.1: Code showing the list of possible states (and there corresponding IDs)

B.2 AsynchronousCommonUtil

Below is the code for the `AsynchronousCommonUtil`. This class has been specifically provided to show how priorities were intergrated into the asynchronous class. With the `nextStatesWithPriority` function being called rather than the `nextStates` on calculating state traces for priority based networks. Also shows the use of the logger for logging to file.

```
package network_models.boolean_network.asynchronous.utils;

import network_models.boolean_network.common.models.BooleanFunction;
import network_models.boolean_network.common.models.BooleanNetworkState;
import network_models.boolean_network.common.models.BooleanNodeState;
import utils.BoolNetLogger;

import java.util.*;
import java.util.logging.Logger;

/**
 * Created by Will on 14/02/2017.
 *
 * Common asynchronous functions are defined here (Repeated Code)
 *
 */
public class AsynchronousCommonUtil {

        // logger for logging to log file
    private final static Logger LOGGER = BoolNetLogger.getLogger(
       AsynchronousCommonUtil.class);

    /**
     *
     * Find the next states for the asynchronous network
     *
     * @param startSt State to start from
     * @param functions List of boolean functions for the network
     * @return returns a list of states from the start state
     */
    public static LinkedList<BooleanNetworkState> nextStates(
       BooleanNetworkState startSt, LinkedList<BooleanFunction> functions)
       {

        LOGGER.finer("Start state" + startSt.toString());

        // List of all the possible next states from the given state
        LinkedList<BooleanNetworkState> generatedStates = new LinkedList
           <>();
        LinkedList<String> currentStates = new LinkedList<>();

        // We want to iterator through all functions to map the node name
           to a boolean value
        Iterator<BooleanFunction> itrFunction = functions.iterator();

        // Go through each function to update the state
        for(int i = 0; itrFunction.hasNext(); i++){

            BooleanNodeState[] stateBoolValues = startSt.getEntityState().
               clone();
            // Calculate the state for each node (Given any of the nodes
               could change)
            BooleanFunction function = itrFunction.next();
```

```
            // Run the boolean tree evaluation on the function to test
               this state
            LOGGER.finer("State calculated FROM " + startSt.toString());
            stateBoolValues[i] = new BooleanNodeState(function.getName(),
               function.getRoot().evaluate(startSt));
            LOGGER.finer("State calculated TO " + new BooleanNetworkState(
               stateBoolValues));

            Boolean previous = stateBoolValues[i].getState();
            Boolean currentState  = startSt.getEntityState()[i].getState()
               ;

            if (previous == currentState) {
                continue;
            }

            // Create new state from array
            BooleanNetworkState nextState = new BooleanNetworkState(
               stateBoolValues);
            // We only want to add each possible state once
            if (!currentStates.contains(nextState.toString())){
                // add to generated states
                generatedStates.add(nextState);
                currentStates.add(nextState.toString());
            }
            LOGGER.finer("End state " + i + ": " + nextState.toString());
        }
        LOGGER.finer("End states : " + Arrays.toString(generatedStates.
           toArray()));

        // Return the new state
        return generatedStates;
    }

    /**
     * Find the next states for when priorities are added
     *
     * @param startSt State to start from
     * @param functions List of boolean functions for the network
     * @param priorities Entity mapped to a List of Suppressed Entities
     * @return returns a list of states from the start state
     */
    public static LinkedList<BooleanNetworkState> nextStatesWithPriority(
       BooleanNetworkState startSt, LinkedList<BooleanFunction> functions,
        HashMap<String,LinkedList<String>> priorities) {
        LOGGER.finer("Start state" + startSt.toString());

        // We want to iterator through all functions to map the node name
           to a boolean value
        Iterator<BooleanFunction> itrFunction = functions.iterator();

        HashMap<String,BooleanNetworkState> hashMap = new HashMap<>();
        // Go through each function to update the state
        for(int i = 0; itrFunction.hasNext(); i++){
```

```
            BooleanNodeState[] stateBoolValues = startSt.getEntityState().
                clone();
            // Calculate the state for each node (Given any of the nodes
                could change)
            BooleanFunction function = itrFunction.next();
            // Run the boolean tree evaluation on the function to test
                this state
            LOGGER.finer("State calculated FROM " + startSt.toString());
            stateBoolValues[i] = new BooleanNodeState(function.getName(),
                function.getRoot().evaluate(startSt));
            LOGGER.finer("State calculated TO " + new BooleanNetworkState(
                stateBoolValues));


            Boolean previous = stateBoolValues[i].getState();
            Boolean currentState  = startSt.getEntityState()[i].getState()
                ;

            // If the state has not changed do nothing
            if (previous == currentState) {
                continue;
            }

            // Create new state from array
            BooleanNetworkState nextState = new BooleanNetworkState(
                stateBoolValues);

            hashMap.put(stateBoolValues[i].getName(),nextState);

            LOGGER.finer("End state " + i + ": " + nextState.toString());
        }

        // Check priorities
        for (String state: priorities.keySet()){
            // If the state exists then suppress others
            if(hashMap.containsKey(state)) {
                for (String suppressedState : priorities.get(state)) {
                    hashMap.remove(suppressedState);
                }
            }
        }

        LinkedList<BooleanNetworkState> generatedStates = new LinkedList
            <>();
        for (String state : hashMap.keySet()){
            generatedStates.add(hashMap.get(state));
        }

        LOGGER.finer("End states :" + Arrays.toString(generatedStates.
            toArray()));

        // Return the new state
        return generatedStates;
```

```
    }
}
```

Original project plan

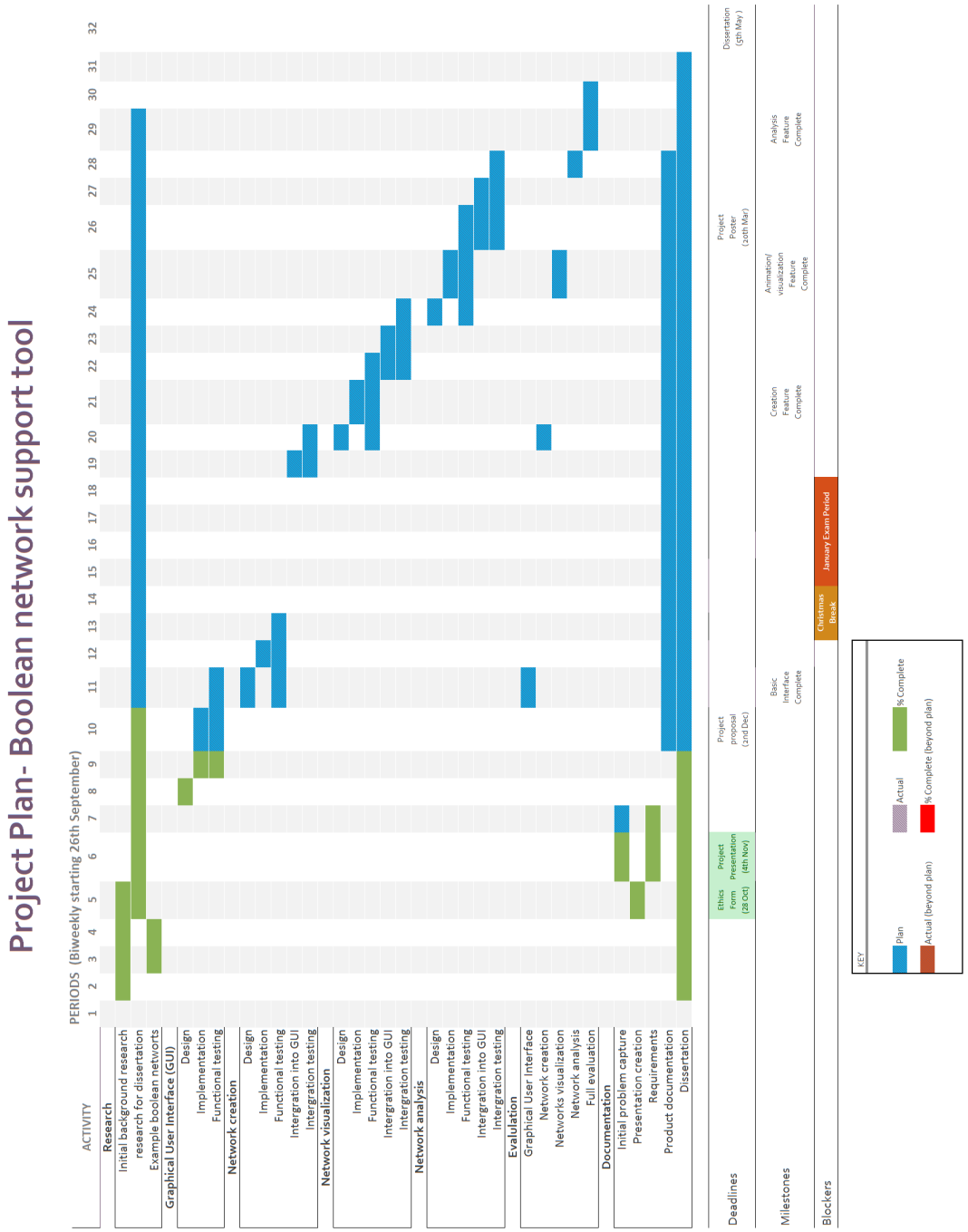


Figure C.1: Project Plan

Appendix D

Example Networks

This appendix provides a number of networks both real life examples and simplified examples.

D.1 Basic demo Network

$$\textit{Entities:} \quad [\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]$$

$$\textit{Functions :} \quad \begin{bmatrix} \text{A} & = & \text{A OR NOT C} \\ \text{B} & = & \text{A AND C} \\ \text{C} & = & \text{B} \end{bmatrix}$$

Figure D.1: Boolean network Example

D.2 Pluripotency demo Network

Entities:

pluri_exo , pluri_exo_meth , pluri_exo_het_chrom , pluri_net , pluri_meth , pluri_het_chrom , B_net , B_meth , B_het_chrom , A_net , A_meth , A_het_chrom , dnmt , de_meth

Functions :

pluri_exo	=	NOT pluri_exo_meth OR NOT pluri_exo_het_chrom
pluri_exo_meth	=	pluri_exo_meth
pluri_exo_het_chrom	=	pluri_exo_het_chrom
pluri_net	=	(pluri_net OR pluri_exo) AND NOT(B_net OR A_net) AND NOT pluri_meth AND NOT pluri_het_chrom
pluri_meth	=	pluri_meth
pluri_het_chrom	=	pluri_het_chrom
B_net	=	B_net AND NOT (pluri_net OR A_net) AND NOT B_meth AND NOT B_het_chrom
B_meth	=	B_meth
B_het_chrom	=	B_het_chrom
A_net	=	A_net AND NOT (pluri_net OR B_net) AND NOT A_meth AND NOT A_het_chrom
A_meth	=	A_meth
A_het_chrom	=	A_het_chrom
dnmt	=	pluri_net OR pluri_exo OR dnmt
de_meth	=	pluri_net OR pluri_exo OR de_meth

Figure D.2: Pluripotency Network (a submodel from Flttmann et al. (2012) [25])

D.3 Mammalian cell cycle synchronous

Entities: [CycD, Rb, E2F, CycE, CycA, p27, Cdc20, cdh1, UbcH10, CycB]

Functions :

CycD	=	CycD
Rb	=	NOT CycD AND NOT CycE AND NOT CycA AND NOT CycB OR NOT CycD AND NOT CycE AND CycA AND p27 AND NOT CycB OR NOT CycD AND CycE AND p27 AND NOT CycB
E2F	=	NOT Rb AND NOT CycA AND NOT CycB OR NOT Rb AND CycA AND p27 AND NOT CycB
CycE	=	NOT Rb AND E2F
CycA	=	NOT Rb AND NOT E2F AND CycA AND NOT Cdc20 AND NOT cdh1 OR NOT Rb AND NOT E2F AND CycA AND NOT Cdc20 AND cdh1 AND NOT UbcH10 OR NOT Rb AND E2F AND NOT Cdc20 AND NOT cdh1 OR NOT Rb AND E2F AND NOT Cdc20 AND cdh1 AND NOT UbcH10
p27	=	NOT CycD AND NOT CycE AND NOT CycA AND NOT CycB OR NOT CycD AND NOT CycE AND CycA AND p27 AND NOT CycB OR NOT CycD AND CycE AND NOT CycA AND p27 AND NOT CycB
Cdc20	=	CycB
cdh1	=	NOT CycA AND NOT Cdc20 AND NOT CycB OR NOT CycA AND Cdc20 OR CycA AND NOT p27 AND Cdc20 OR CycA AND p27 AND NOT Cdc20 AND NOT CycB OR CycA AND p27 AND Cdc20 a
UbcH10	=	NOT CycA AND NOT Cdc20 AND NOT cdh1 OR NOT CycA AND NOT Cdc20 AND cdh1 AND UbcH10 AND CycB OR NOT CycA AND Cdc20 AND NOT cdh1 OR NOT CycA AND Cdc20 AND cdh1 AND UbcH10 OR CycA AND NOT cdh1 OR CycA AND cdh1 AND UbcH10
CycB	=	NOT Cdc20 AND NOT cdh1

Figure D.3: Mammalian Network [23]

Appendix E

Testing results

This appendix contains the most recent result of the Unit and System tests run as well as the coverage results for these tests.

E.1 Test Results

This section contains the results from both system and unit tests. In the most resent run all of the tests have passed (153/153) taking a total of 18 minutes and 45 seconds.

The classification for Unit/System tests are defined as follows:

- System tests
  - StateGraph
  - StateTraces
  - WiringDiagram
  - HomePageTests
  - NetworkEditor
  - StartupTests
- Unit tests
  - AsynchronousBooleanNetworkFactoryTest
  - AsynchronousStateTraceFactoryTest
  - CommonNetworkInterfaceTest
  - SynchronousBooleanNetworkFactoryTest
  - SynchronousStateNetworkFactoryTest
  - SynchronousStateTraceFactoryTest
  - BooleanFunctions

junit in BooleanNetworks: 153 total, 153 passed 18 m 45 s

BooleanFunctions		136 ms
BooleanFunctions.Creation_Null	passed	76 ms
BooleanFunctions.Creation_Basic	passed	5 ms
BooleanFunctions.Test3States	passed	54 ms
BooleanFunctions.Creation_ANDCommand	passed	1 ms
AsynchronousBooleanNetworkFactoryTest		29 ms
AsynchronousBooleanNetworkFactoryTest.creatingFactoryAndUtils	passed	23 ms
AsynchronousBooleanNetworkFactoryTest.removeFunction	passed	3 ms
AsynchronousBooleanNetworkFactoryTest.generateStateNetwork	passed	1 ms
AsynchronousBooleanNetworkFactoryTest.addFunction	passed	1 ms
AsynchronousBooleanNetworkFactoryTest.generateEmptyStateNetwork	passed	0 ms
AsynchronousBooleanNetworkFactoryTest.replaceFunction	passed	1 ms
AsynchronousStateTraceFactoryTest		2 ms
AsynchronousStateTraceFactoryTest.generateStateNetwork	passed	1 ms
AsynchronousStateTraceFactoryTest.generatePriorityStateNetwork	passed	1 ms
CommonNetworkInterfaceTest		54 ms
CommonNetworkInterfaceTest.canSupportPriorities_synchronous	passed	13 ms
CommonNetworkInterfaceTest.checkFunction_entityInUse_asynchronous	passed	1 ms
CommonNetworkInterfaceTest.checkFunctionBeforeChange_emptyEntity_asynchronous	passed	2 ms
CommonNetworkInterfaceTest.setPriorities_synchronous	passed	0 ms
CommonNetworkInterfaceTest.exportTraceToGraphViz_synchronous	passed	0 ms
CommonNetworkInterfaceTest.addFunction_valid_synchronous	passed	0 ms
CommonNetworkInterfaceTest.checkFunctionBeforeChange_valid_asynchronous	passed	2 ms
CommonNetworkInterfaceTest.hasPriorities_synchronous	passed	0 ms

CommonNetworkInterfaceTest.checkFunction_emptyCommand_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.addFunction_valid_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.checkFunctionBeforeChange_entityInUse_asynchronous	passed	3 ms
CommonNetworkInterfaceTest.getTraces_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.getNetworkType_Synchronous	passed	0 ms
CommonNetworkInterfaceTest.getStates_synchronous	passed	3 ms
CommonNetworkInterfaceTest.replaceFunction_invalid_synchronous	passed	1 ms
CommonNetworkInterfaceTest.setHiddenNodes_asynchronous	passed	1 ms
CommonNetworkInterfaceTest.replaceFunction_valid_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.exportTraceToGraphViz_asynchronous	passed	1 ms
CommonNetworkInterfaceTest.removeFunction_invalid_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.checkFunction_emptyEntity_synchronous	passed	0 ms
CommonNetworkInterfaceTest.getFunctions_Asynchronous	passed	1 ms
CommonNetworkInterfaceTest.hasPriorities_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.getTraces_synchronous	passed	1 ms
CommonNetworkInterfaceTest.getPriorities_asynchronous	passed	1 ms
CommonNetworkInterfaceTest.checkCommandEntities_synchronous	passed	0 ms
CommonNetworkInterfaceTest.replaceFunction_noneExisting_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.hasPriorities_noPriorities_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.getPossibleStates_synchronous	passed	0 ms
CommonNetworkInterfaceTest.setPriorities_asynchronous	passed	1 ms
CommonNetworkInterfaceTest.getPossibleStates_asynchronous	passed	1 ms
CommonNetworkInterfaceTest.checkFunction_entityInUse_synchronous	passed	0 ms
CommonNetworkInterfaceTest.checkFunction_emptyCommand_synchronous	passed	0 ms
CommonNetworkInterfaceTest.getHiddenNodes_asynchronous	passed	1 ms
CommonNetworkInterfaceTest.replaceFunction_valid_synchronous	passed	0 ms
CommonNetworkInterfaceTest.getNetworkName_Asynchronous	passed	0 ms

CommonNetworkInterfaceTest.addFunction_invalid_synchronous	passed	0 ms
CommonNetworkInterfaceTest.getFunctions_SynchronousWithAsynchronous	passed	0 ms
CommonNetworkInterfaceTest.checkFunctionBeforeChange_emptyEntity_synchronous	passed	2 ms
CommonNetworkInterfaceTest.removeFunction_valid_synchronous	passed	0 ms
CommonNetworkInterfaceTest.getNetworkType_Asynchronous	passed	1 ms
CommonNetworkInterfaceTest.checkFunctionBeforeChange_emptyCommand_synchronous	passed	3 ms
CommonNetworkInterfaceTest.getHiddenNodes_synchronous	passed	0 ms
CommonNetworkInterfaceTest.checkFunction_valid_synchronous	passed	0 ms
CommonNetworkInterfaceTest.removeFunction_valid_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.checkFunctionBeforeChange_valid_synchronous	passed	2 ms
CommonNetworkInterfaceTest.checkFunction_valid_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.checkCommandEntities_asynchronous	passed	1 ms
CommonNetworkInterfaceTest.getFunctions_Synchronous	passed	4 ms
CommonNetworkInterfaceTest.setHiddenNodes_synchronous	passed	2 ms
CommonNetworkInterfaceTest.checkFunction_emptyEntity_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.removeFunction_invalid_synchronous	passed	0 ms
CommonNetworkInterfaceTest.getPriorities_synchronous	passed	1 ms
CommonNetworkInterfaceTest.replaceFunction_noneExisting_synchronous	passed	0 ms
CommonNetworkInterfaceTest.addFunction_invalid_asynchronous	passed	1 ms
CommonNetworkInterfaceTest.getNetworkName_Synchronous	passed	0 ms
CommonNetworkInterfaceTest.checkFunctionBeforeChange_entityInUse_synchronous	passed	3 ms
CommonNetworkInterfaceTest.checkFunctionBeforeChange_emptyCommand_asynchronous	passed	1 ms
CommonNetworkInterfaceTest.getStates_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.canSupportPriorities_asynchronous	passed	0 ms
CommonNetworkInterfaceTest.replaceFunction_invalid_asynchronous	passed	0 ms

SynchronousBooleanNetworkFactoryTest		6 ms
SynchronousBooleanNetworkFactoryTest.creatingFactoryAndUtils	passed	0 ms
SynchronousBooleanNetworkFactoryTest.removeFunction	passed	3 ms
SynchronousBooleanNetworkFactoryTest.generateStateNetwork	passed	1 ms
SynchronousBooleanNetworkFactoryTest.addFunction	passed	1 ms
SynchronousBooleanNetworkFactoryTest.generateEmptyStateNetwork	passed	0 ms
SynchronousBooleanNetworkFactoryTest.replaceFunction	passed	1 ms
SynchronousStateNetworkFactoryTest		2 ms
SynchronousStateNetworkFactoryTest.generateStates_nullRoot	passed	1 ms
SynchronousStateNetworkFactoryTest.generateStateNetwork	passed	1 ms
SynchronousStateTraceFactoryTest		0 ms
SynchronousStateTraceFactoryTest.generateStateNetwork	passed	0 ms
HomePageTests		49.68 s
HomePageTests.QuitBtn	passed	6.25 s
HomePageTests.HelpBtn	passed	3.29 s
HomePageTests.CorrectStartup	passed	2.67 s
HomePageTests.createNewSynchronousNetwork	passed	3.46 s
HomePageTests.OpenFileBtn	passed	3.73 s
HomePageTests.SettingsChange	passed	6.96 s
HomePageTests.CancelNewNetwork	passed	3.10 s
HomePageTests.ImportNetwork	passed	5.88 s
HomePageTests.createNewAsynchronousNetwork	passed	3.13 s
HomePageTests.SettingsBtn	passed	2.83 s
HomePageTests.AboutBtn	passed	2.76 s
HomePageTests.NewFileBtn	passed	2.82 s
HomePageTests.ImportBtn	passed	2.80 s
NetworkEditor		16 m 0 s
NetworkEditor.asynchronous_Save_EmptyFile	passed	7.03 s
NetworkEditor.asynchronous_PrioritiesBtn	passed	4.24 s
NetworkEditor.synchronous_entityDoesNotExist_Added	passed	45.84 s
NetworkEditor.synchronous_invalidBrackets	passed	7.79 s
NetworkEditor.synchronous_Save_ValidNetworkFile	passed	8.18 s

NetworkEditor.asynchronous_AddPriorities	passed	7.17 s
NetworkEditor.asynchronous_editRow	passed	4.35 s
NetworkEditor.asynchronous_entityDoesNotExist_Edited	passed	47.45 s
NetworkEditor.asynchronous_invalidOR	passed	25.91 s
NetworkEditor.asynchronous_Save_ValidNetworkFile	passed	8.18 s
NetworkEditor.synchronous_invalidOR	passed	26.04 s
NetworkEditor.asynchronous_ExportBtn	passed	3.14 s
NetworkEditor.synchronous_checkNoCommandWarning	passed	3.91 s
NetworkEditor.synchronous_ExportBtn	passed	3.19 s
NetworkEditor.synchronous_Save_EmptyFile	passed	7.03 s
NetworkEditor.asynchronous_basicAddFunction	passed	13.85 s
NetworkEditor.asynchronous_invalidBrackets	passed	7.93 s
NetworkEditor.synchronous_entityDoesNotExist_Edited	passed	49.53 s
NetworkEditor.synchronous_Save_NetworkWithWarningFile	passed	8.39 s
NetworkEditor.asynchronous_Save_NetworkWithWarningFile	passed	7.78 s
NetworkEditor.synchronous_invalidAND	passed	34.15 s
NetworkEditor.synchronous_invalidNOT	passed	12.62 s
NetworkEditor.asynchronous_checkNoEntityWarning	passed	4.09 s
NetworkEditor.asynchronous_SaveFileBtn	passed	3.77 s
NetworkEditor.synchronous_editRow	passed	4.54 s
NetworkEditor.asynchronous_entityDoesNotExist	passed	49.49 s
NetworkEditor.asynchronous_checkNoCommandWarning	passed	4.16 s
NetworkEditor.asynchronous_invalidAND	passed	36.59 s
NetworkEditor.asynchronous_invalidNOT	passed	13.83 s
NetworkEditor.synchronous_Priorities	passed	3.87 s
NetworkEditor.asynchronous_entityDoesNotExist_Added	passed	52.47 s
NetworkEditor.synchronous_entityDoesNotExist	passed	52.83 s
NetworkEditor.asynchronous_deleteRow	passed	4.29 s
NetworkEditor.synchronous_deleteRow	passed	4.28 s
NetworkEditor.synchronous_basicAddFunction	passed	13.97 s
NetworkEditor.synchronous_checkNoEntityWarning	passed	4.19 s
NetworkEditor.asynchronous_viewRow	passed	26.46 s
NetworkEditor.addOrEditFunction_entities	passed	4.34 s
NetworkEditor.synchronous_testCommands	passed	2 m 31 s
NetworkEditor.synchronous_SaveFileBtn	passed	3.97 s

NetworkEditor.synchronous_viewRow	passed	26.67 s
NetworkEditor.asynchronous_testCommands	passed	2 m 31 s
StartupTests		1.65 s
StartupTests.CorrectStartup	passed	1.65 s
StateGraph		1 m 23 s
StateGraph.asynchronous_hideDiagramDialog	passed	10.40 s
StateGraph.synchronous_hideDiagramEntities	passed	12.71 s
StateGraph.synchronous_export_GraphViz	passed	3.92 s
StateGraph.asynchronous_changeDiagramLayout	passed	6.93 s
StateGraph.synchronous_SaveDiagram	passed	6.62 s
StateGraph.asynchronous_export_GraphViz	passed	7.72 s
StateGraph.asynchronous_hideDiagramEntities	passed	12.46 s
StateGraph.synchronous_changeDiagramLayout	passed	6.94 s
StateGraph.synchronous_hideDiagramDialog	passed	8.66 s
StateGraph.asynchronous_SaveDiagram	passed	6.57 s
StateTraces		7.50 s
StateTraces.asynchronous_loadDiagram	passed	3.74 s
StateTraces.synchronous_loadDiagram	passed	3.77 s
WiringDiagram		22.50 s
WiringDiagram.asynchronous_loadDiagram	passed	5.43 s
WiringDiagram.synchronous_SaveDiagram	passed	6.70 s
WiringDiagram.synchronous_loadDiagram	passed	3.73 s
WiringDiagram.asynchronous_SaveDiagram	passed	6.63 s

E.2 Coverage Results

This section contains the Coverage result from both system and unit tests. With over 80% coverage the majority of code being tested, taking into account defensive code (code that cannot be run, designed to safeguard from errors from future development) and code pace-holders for future development, the coverage is at around 90%.

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	92.3% (72/ 78)	81% (341/ 421)	85.2% (3648/ 4284)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
network_models.boolean_network.asynchronous.factories	100% (2/ 2)	100% (13/ 13)	100% (50/ 50)
network_models.boolean_network.common.models.functionTree	100% (5/ 5)	100% (14/ 14)	100% (25/ 25)
network_models.boolean_network.synchronous.utils	100% (1/ 1)	100% (4/ 4)	100% (16/ 16)
network_models.boolean_network.asynchronous.models	100% (2/ 2)	100% (6/ 6)	100% (8/ 8)
network_models.boolean_network.asynchronous.utils	100% (1/ 1)	100% (4/ 4)	100% (47/ 47)
gui.forms.subForms	100% (3/ 3)	100% (17/ 17)	100% (151/ 151)
network_models.boolean_network.synchronous.models	100% (3/ 3)	100% (11/ 11)	100% (17/ 17)
network_models.boolean_network.common.factories	100% (2/ 2)	91.7% (11/ 12)	93.9% (201/ 214)
network_models	100% (3/ 3)	100% (30/ 30)	93.7% (327/ 349)
gui.forms	100% (11/ 11)	79.5% (62/ 78)	92.9% (671/ 722)
network_models.boolean_network.common.utils	100% (2/ 2)	62.5% (5/ 8)	91.9% (34/ 37)
gui.forms.Dialogs	100% (12/ 12)	61.7% (37/ 60)	86.7% (660/ 761)
network_models.boolean_network.synchronous.factories	100% (3/ 3)	100% (15/ 15)	85.6% (101/ 118)
gui.networks.graphing	100% (2/ 2)	100% (6/ 6)	84.2% (64/ 76)
gui.networks.common	100% (2/ 2)	90% (9/ 10)	82.9% (97/ 117)
network_models.boolean_network.common.models	100% (4/ 4)	77.8% (14/ 18)	80.4% (37/ 46)
gui.main_ui	66.7% (2/ 3)	74.1% (20/ 27)	75.9% (485/ 639)
gui.networks	60% (6/ 10)	67.3% (33/ 49)	74.3% (440/ 592)
utils	100% (4/ 4)	78.1% (25/ 32)	73.6% (184/ 250)
gui.main_ui.dialogs	100% (1/ 1)	80% (4/ 5)	68.9% (31/ 45)

network_models.exceptions	50% (1/ 2)	50% (1/ 2)	50% (2/ 4)

Final system screen-shots

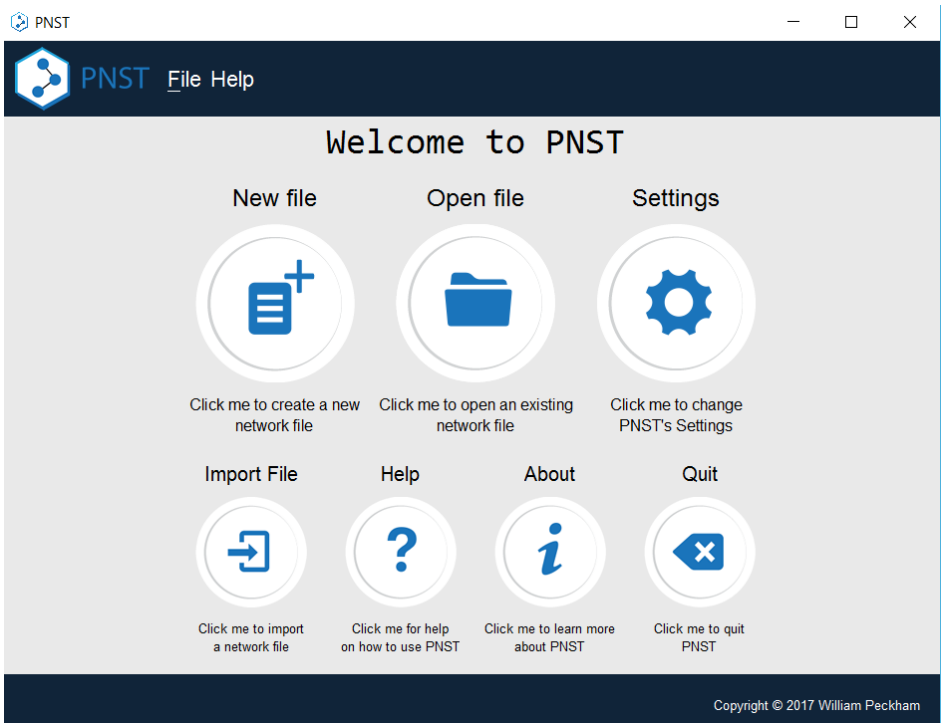


Figure F.1: Home Screen

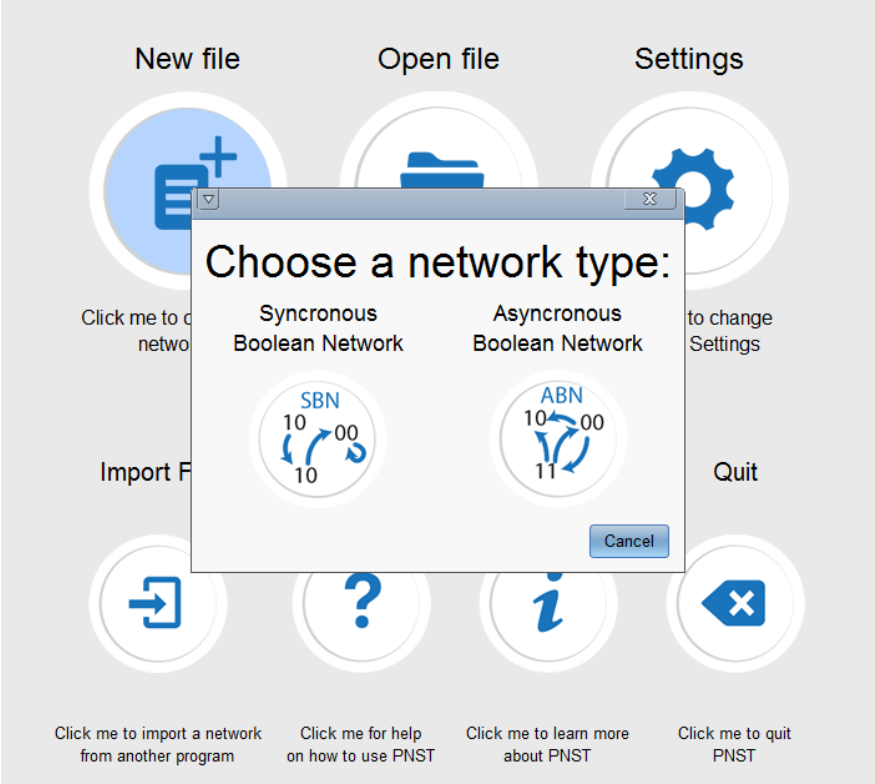


Figure F.2: New File Popup Choice

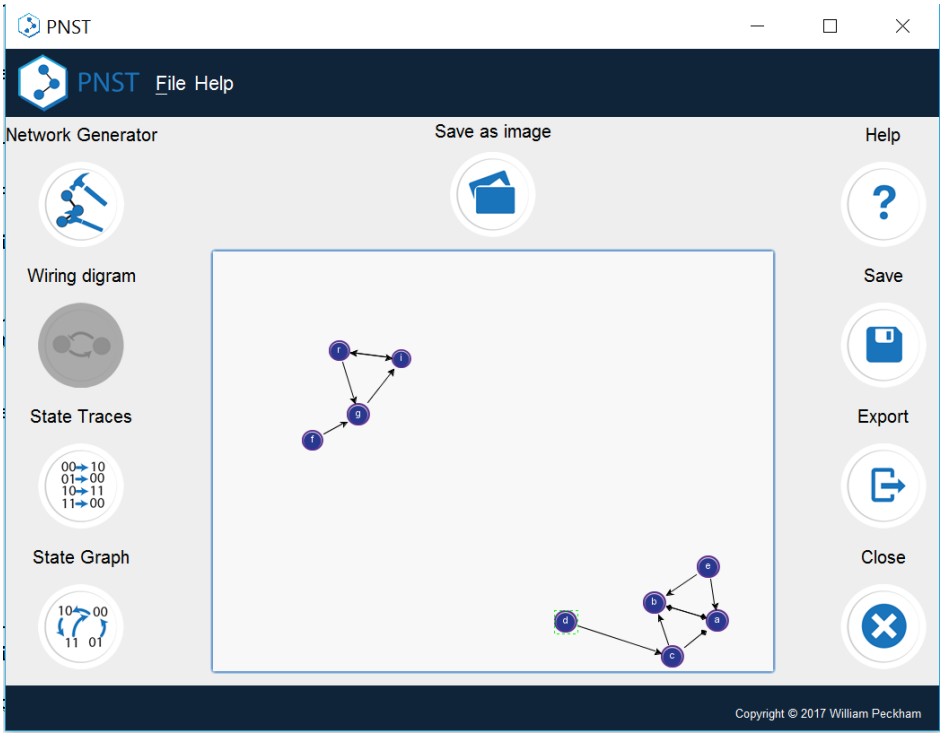


Figure F.3: Wiring Diagram Screen

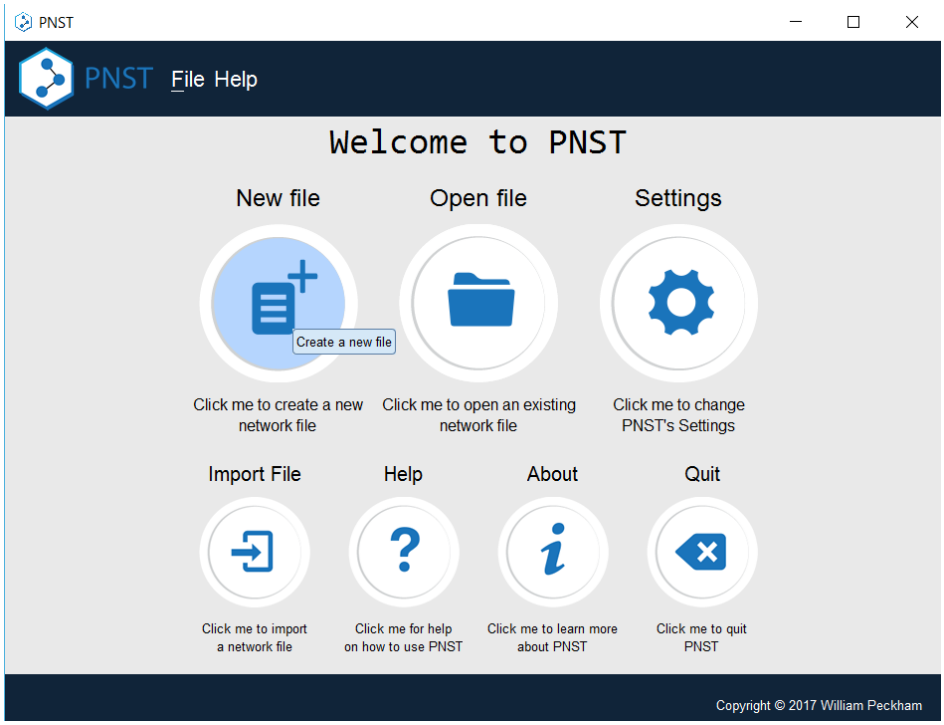


Figure F.4: Home Screen Highlighted Button

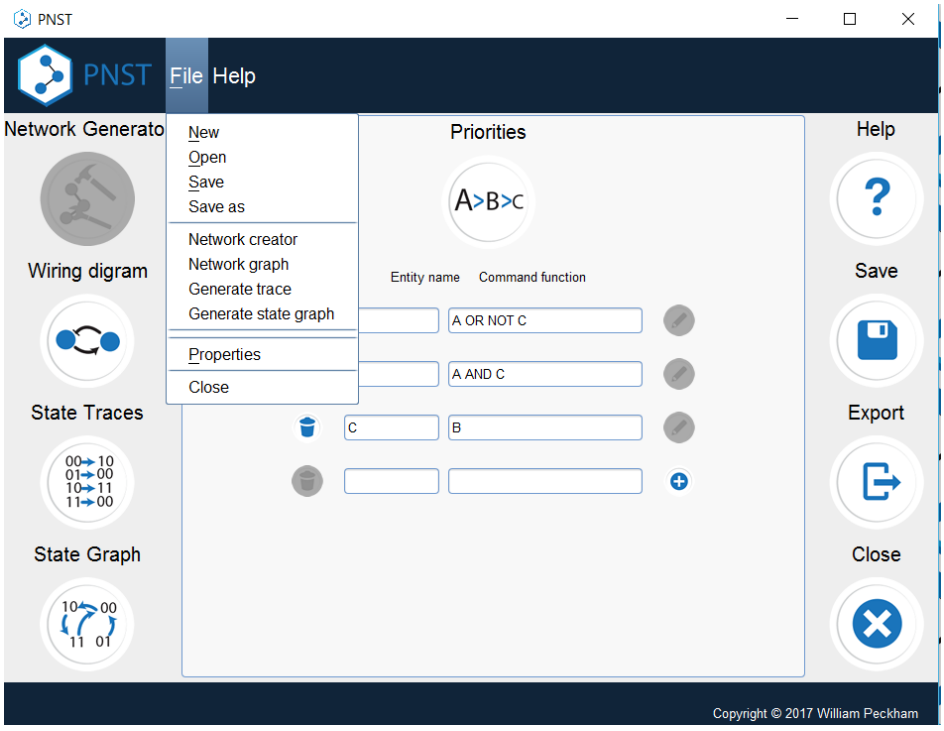


Figure F.5: Menu View

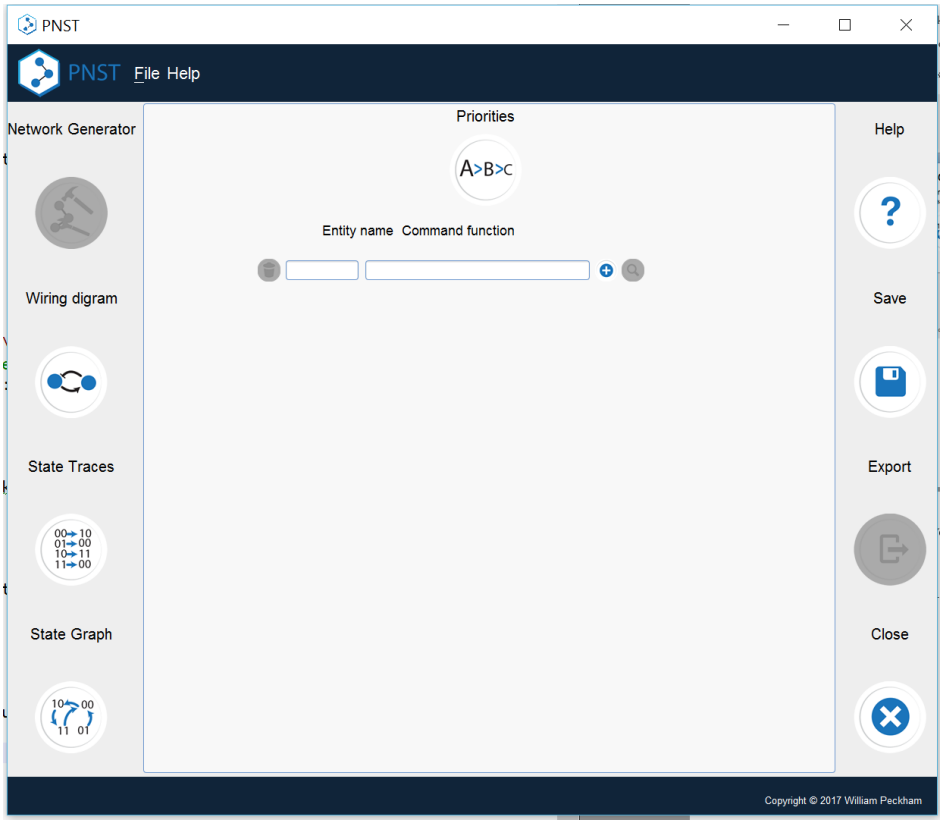


Figure F.6: Network Editor

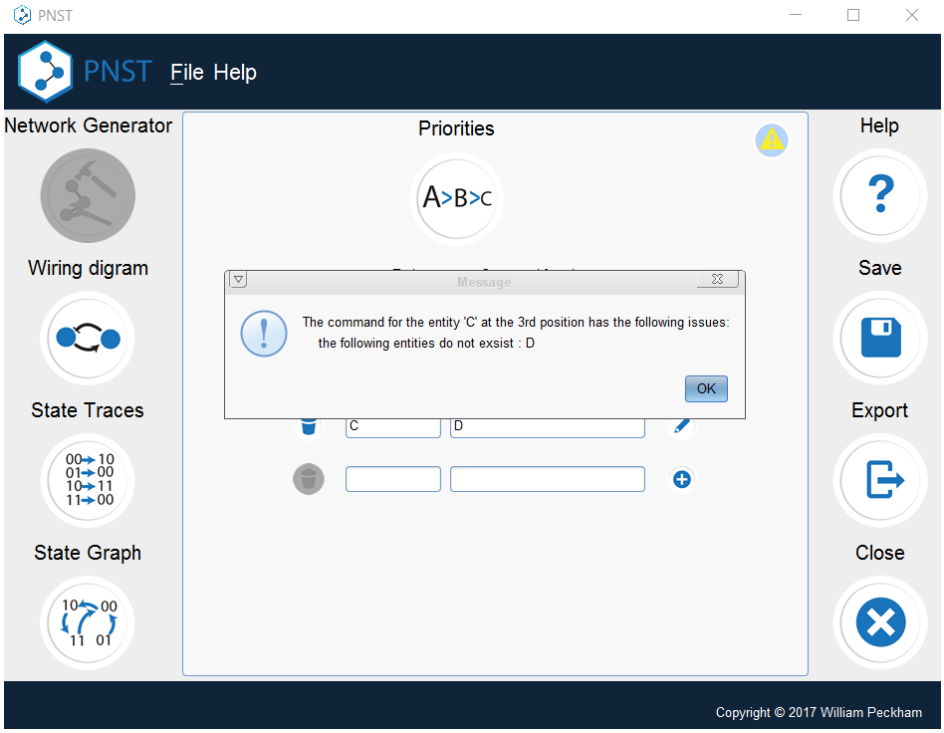


Figure F.7: Network Editor With Invalid Values With Type

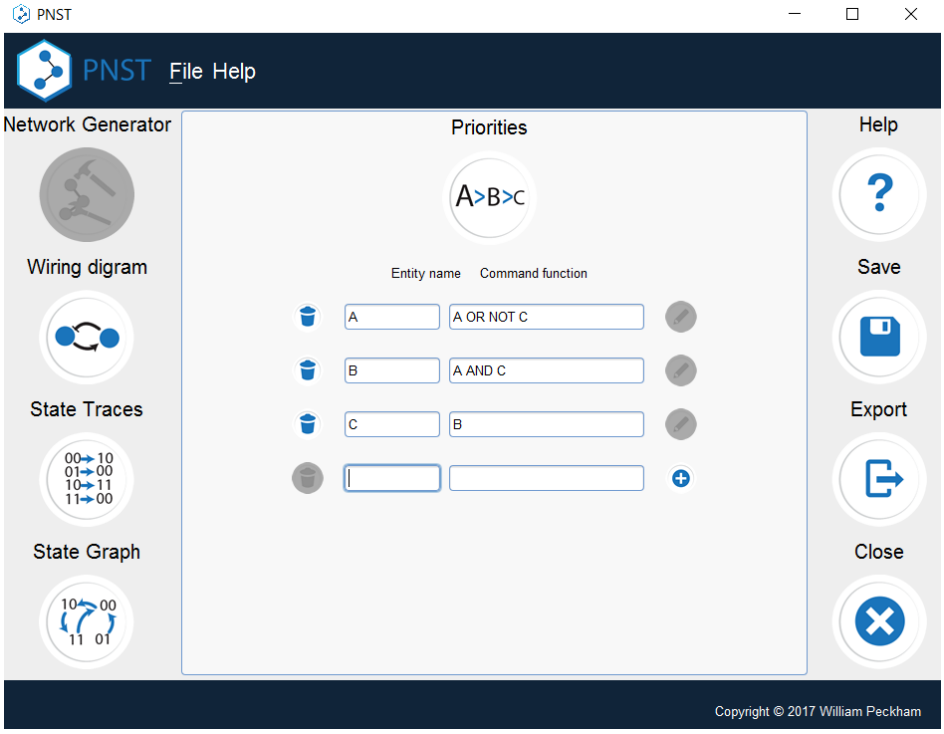


Figure F.8: Network Editor With Values

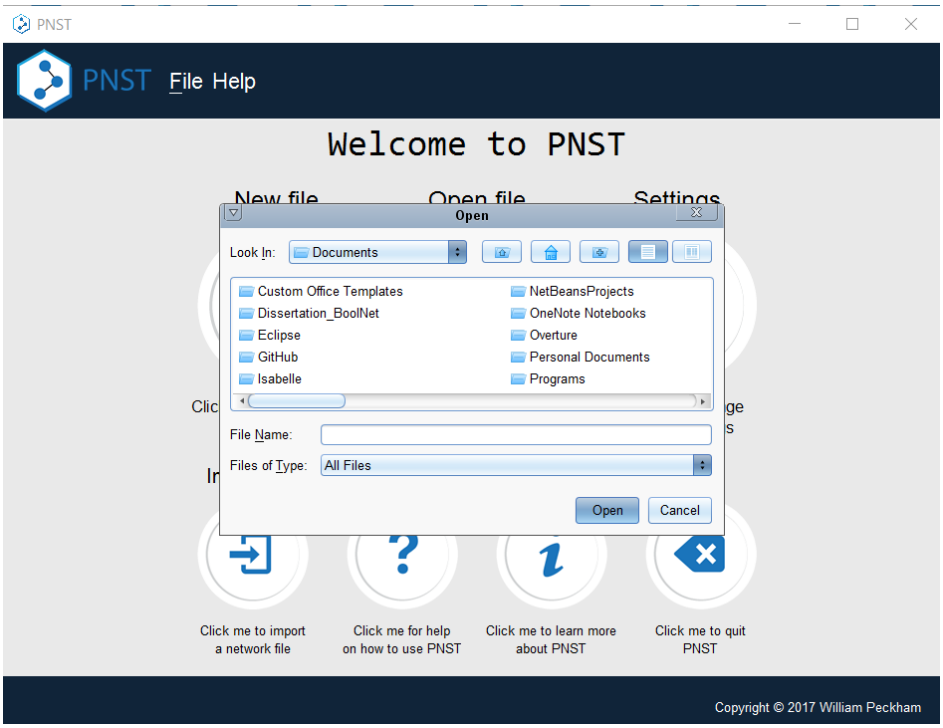


Figure F.9: Open Screen

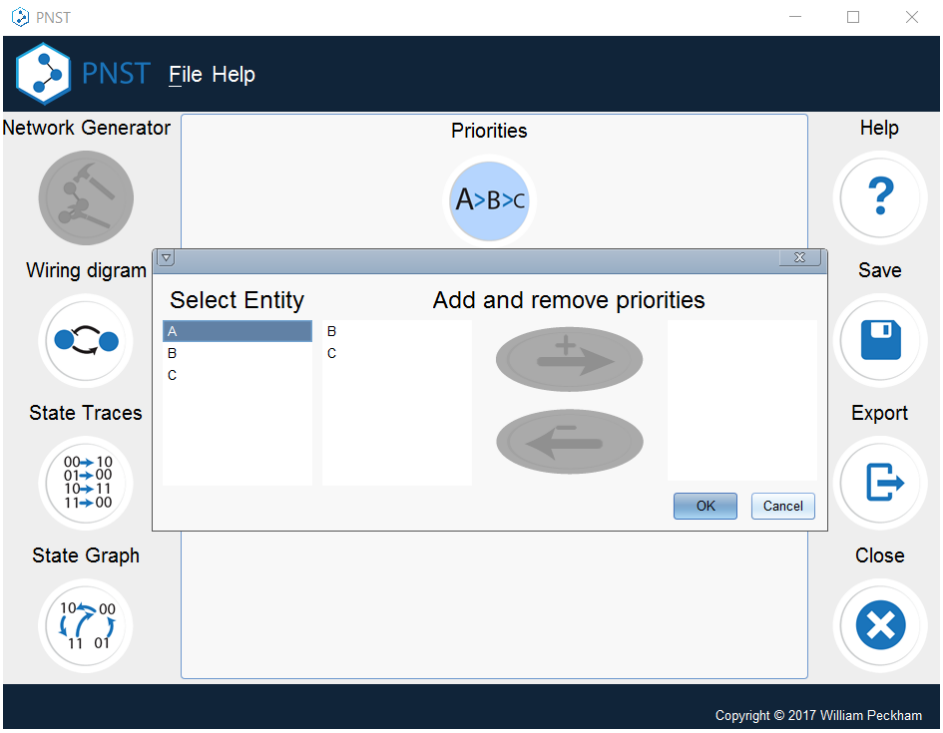


Figure F.10: Priorities

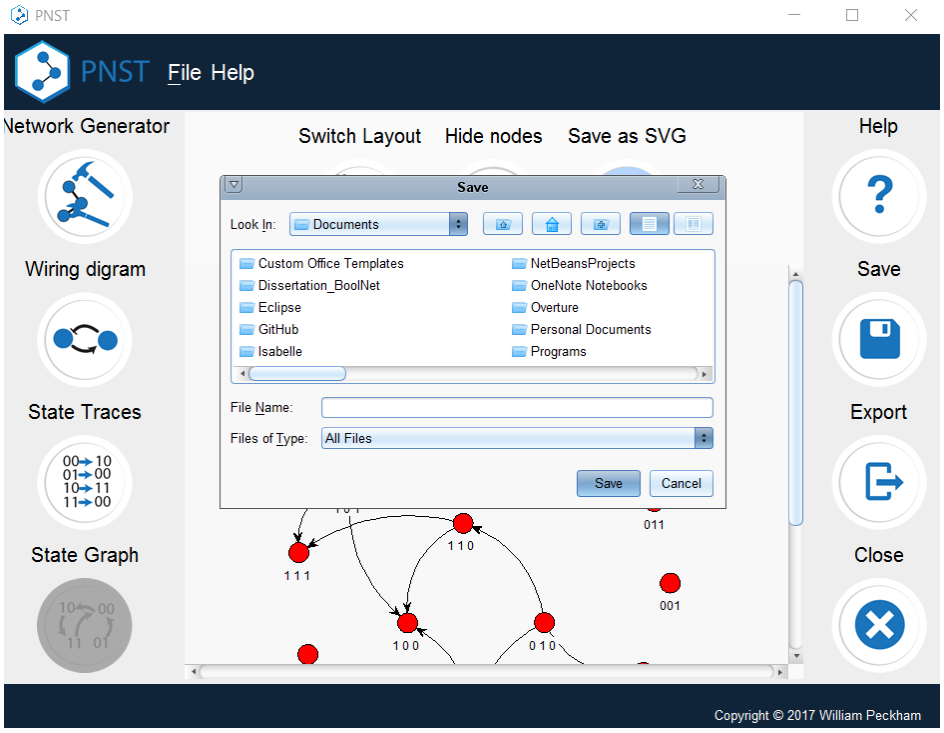


Figure F.11: Save To Image Dialog

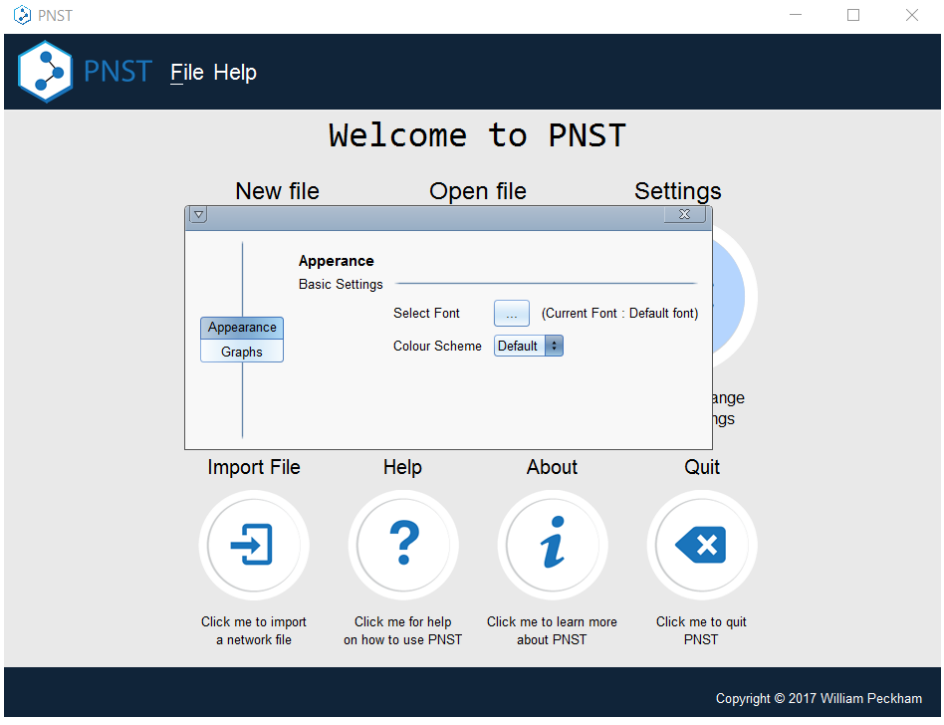


Figure F.12: Settings Screen

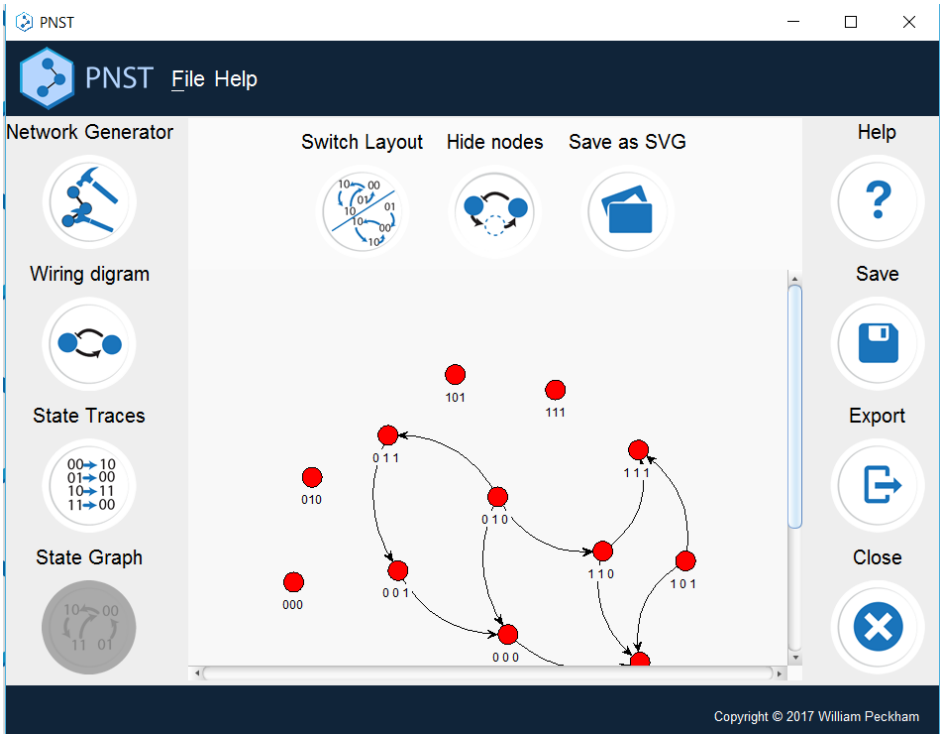


Figure F.13: State Graph View

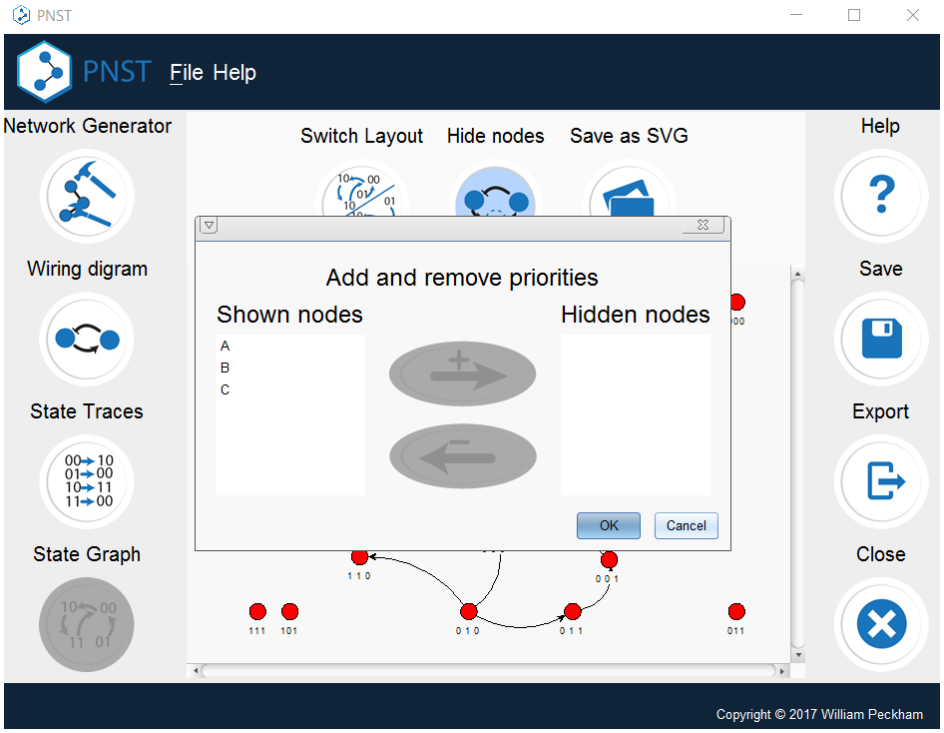


Figure F.14: State Graph View Hide Nodes

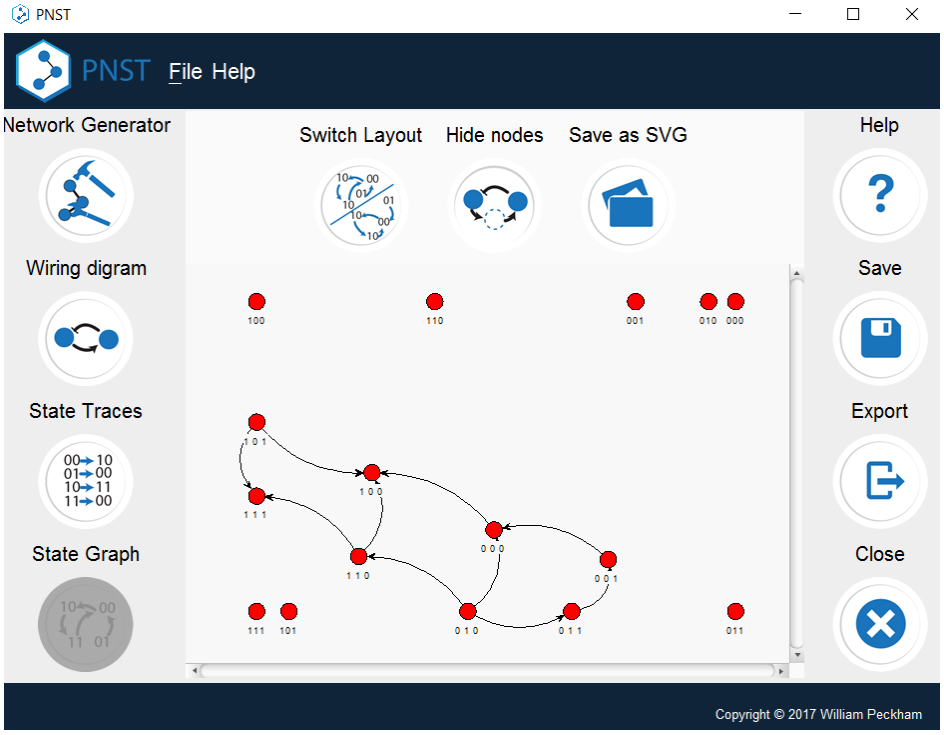


Figure F.15: State Graph View Modifed View

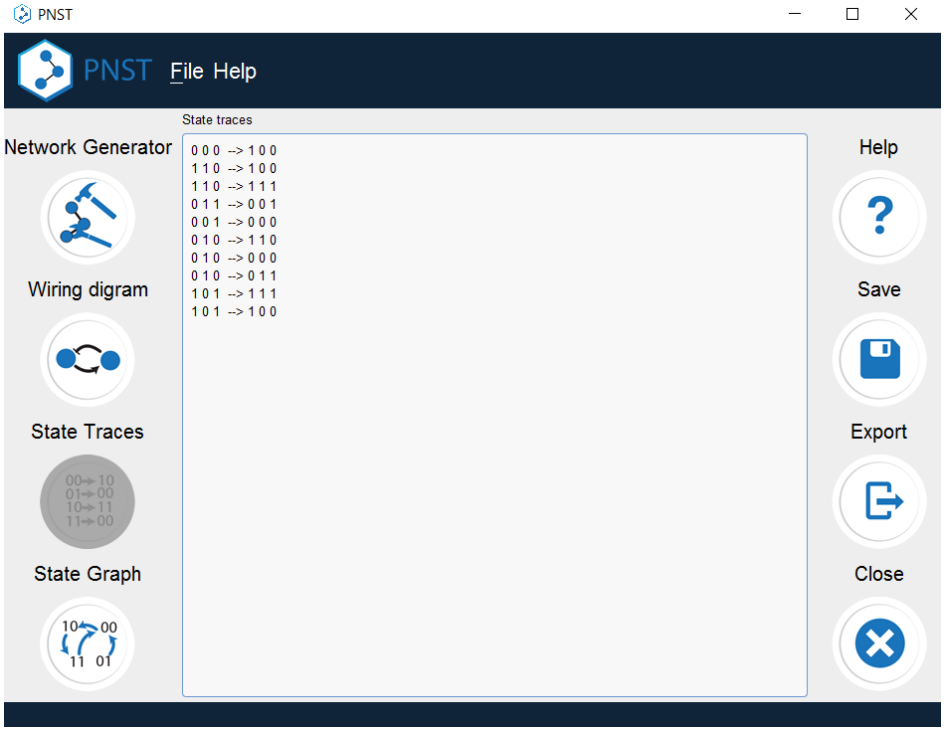
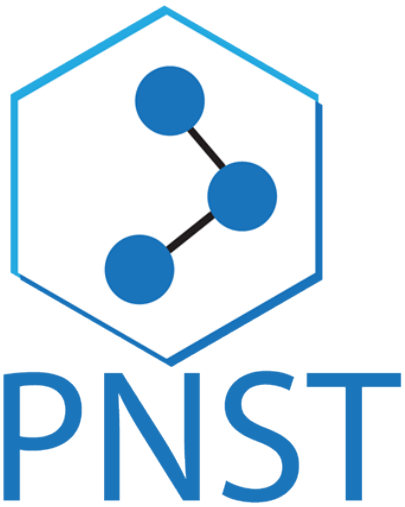


Figure F.16: State Traces Screen

Appendix G

User guide

This Appendix section contains the user guide for the system.



PNST -Network Support Tool User Guide

Produced by William Peckham

Contents

Chapter 1

Overview of the system

PNST is a network support tool, supporting Synchronous and Asynchronous Boolean networks. The tool tackles the state based explosion problem with the user of priority-based asynchronous networks and hiding nodes in the sate graph (see Section 5).

1.1 Navigation

Figure 1.1 shows the home page for the tool. To navigate to different areas of the tool either use the buttons presented or the menu bar located at the top of the tool.

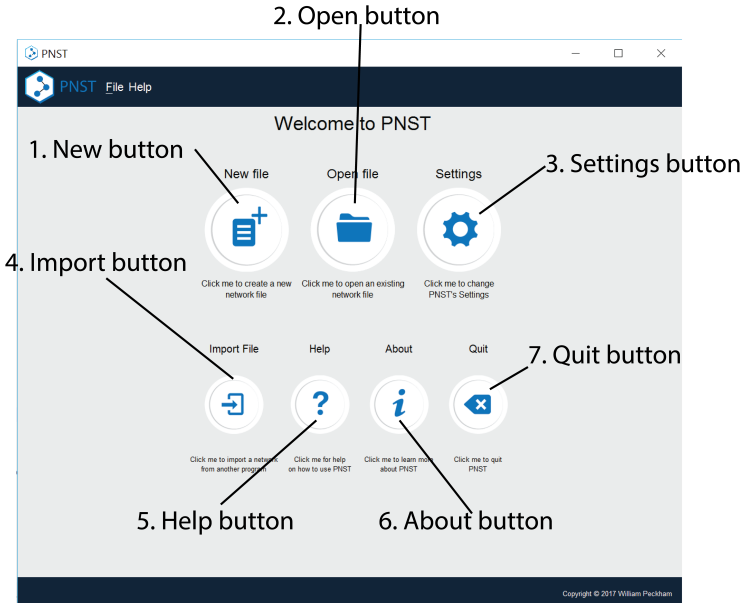


Figure 1.1: Annotated view of the home page

Chapter 2

Creating networks

2.1 Creating a new network

To create a new network, navigate to the home page (See Figure 1.1), then click the 'New file' button (marked as 1. in Figure 1.1). This will present the dialogue shown in Figure 2.1, whereby you need to choose the network type you wish to use. The current supported network types are:

- Synchronous Boolean network
- Asynchronous Boolean network
- Priority-based asynchronous Boolean network through asynchronous Boolean network

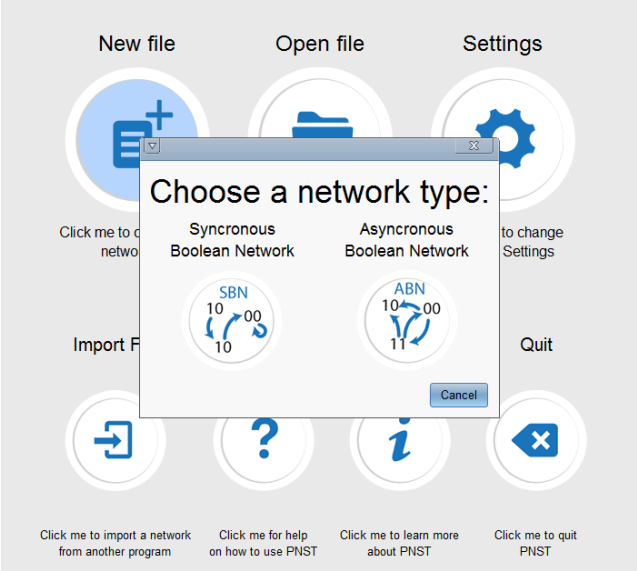


Figure 2.1: Network choice dialogue

Once you have chosen the network type you will be presented with the network editor (See Figure 2.2). From here you can add new entities and commands to the network.

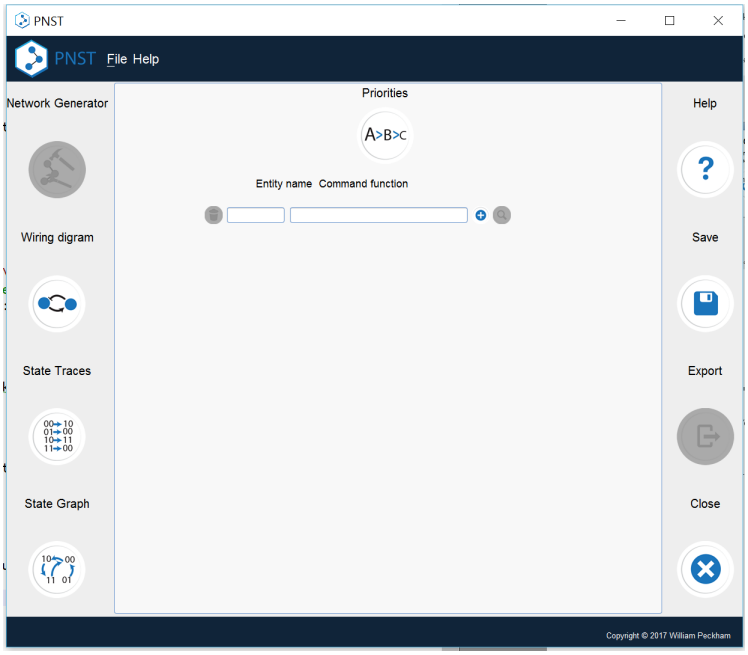


Figure 2.2: The network editor

2.2 Open a new file

To open an existing file click on the the open button on the home screen (Item 2 in Figure 1.1). Then select the file you want to open with the dialogue presented (Figure 2.3) and click open to open the file.

2.3 Importing JSON Based Graph-Exchange Formate (jSBGN) file

To open an existing file from another program in the jSBGN format, click on the the open button on the home screen (Item 4 in Figure 1.1). Next select the jSBGN option on the dialogue presented (Figure 2.4). Then select the file you want to open with the dialogue presented and click open to open the file.

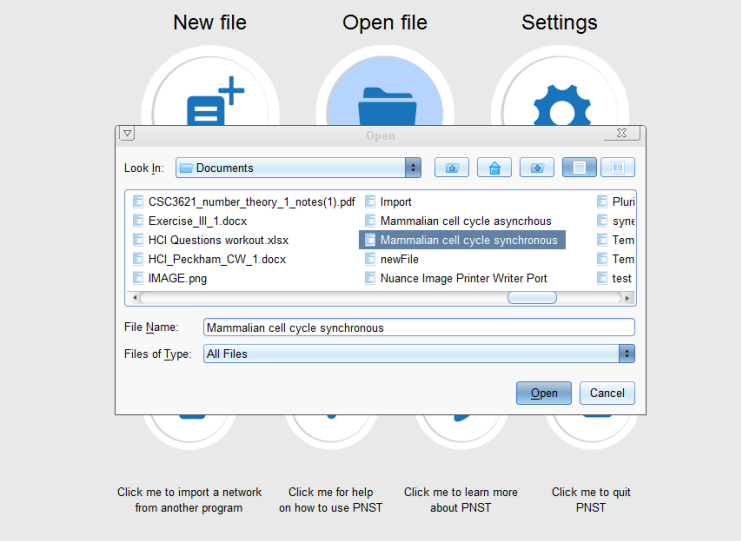


Figure 2.3: Open dialogue

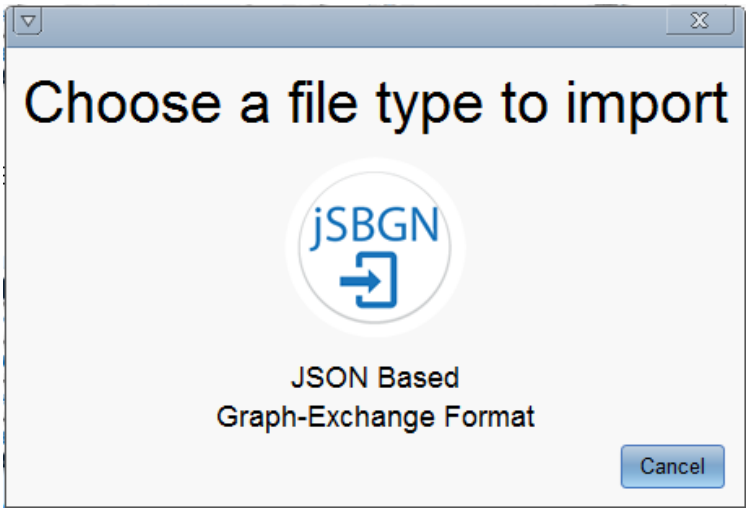


Figure 2.4: Importing dialogue

Chapter 3

The network editor

For help on adding priorities to asynchronous networks, see Section 5.1

3.1 General Layout

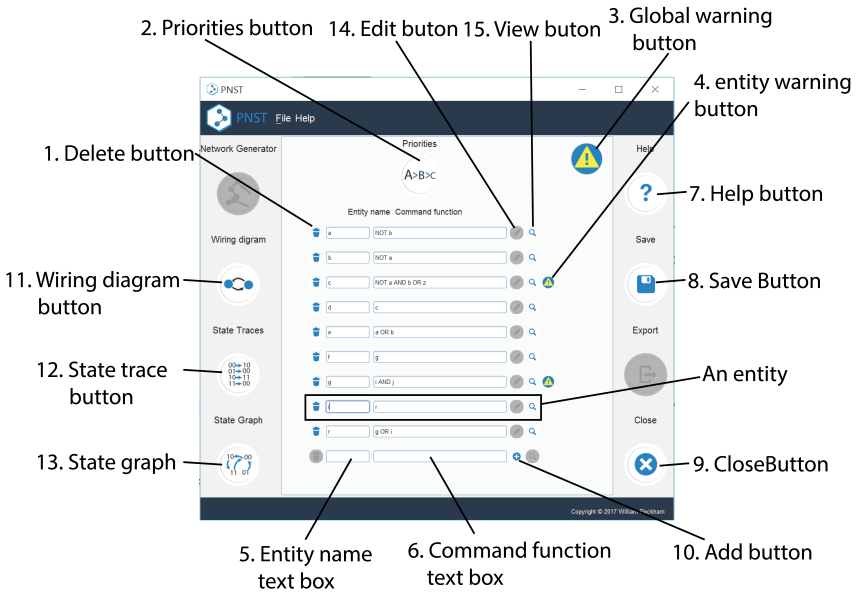


Figure 3.1: Annotated view of the editor

3.2 Supported notation

3.2.1 Boolean networks

For the synchronous and asynchronous networks the following notations are supported (can be used interchangeably):

- Boolean 'Not':

  'NOT', 'not' , '!' or ''

- Boolean 'AND':

  'AND', 'and' , '&&', '&' or '.'

- Boolean 'OR':

  'OR', 'or' , '||' or '+'

- parentheses:

  '(' and ')' - (required in pairs)

3.3 Adding a new entity

To add a new entity simply enter an entity name within the 'Entity name' text box (item 5 in Figure 3.1) and its command within the 'Command function' text box (item 6 in Figure 3.1) and either click the add button (item 10 in Figure 3.1) or hit the 'enter' key. The add button will transform into an edit button and a new entity row created.

3.4 Removing an entity

To remove an entity, simply click the remove button (item 1 in Figure 3.1) of the entity you wish to delete. After clicking the button the entity will be removed from the network.

3.5 Editing an entity

To edit an entity's name or command function, simply enter the new values for the name (item 5 in Figure 3.1) and/or command function (item 6 in Figure 3.1), then click the edit button (item 14 in Figure 3.1) to the right of the entity you wish to edit.

3.6 Viewing an entity

To view an entity in order to see how the command will be interpreted by the tool, click on the view button (item 15 in Figure 3.1). You will be presented with a dialogue showing the entity's name and command function (see Figure 3.2). The command formate is as follows:

- Boolean commands - Light Blue (not underlined)
- entities - highlighted in green and underlined

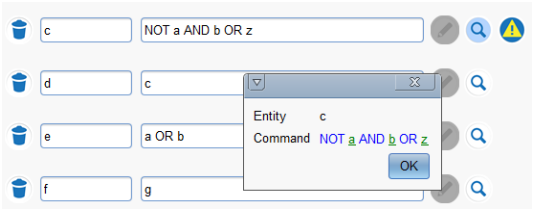


Figure 3.2: Entity view dialogue

3.7 Warnings

When presented with the warning icon, the tool is informing you that there is a problem with the network. There are two warning buttons the 'global warning' button and the 'entity warning' button

3.7.1 Global warning button

By clicking on this icon you will get a message containing all of the warnings for the network.

3.7.2 Entity warning button

By clicking on this icon you will get a message containing the warning message for the entity that it relates to.

3.8 Saving the network

To save a network click the 'Save' button (item 8 in Figure 3.1) or using the menu bar navigate to 'File' - > 'Save'. To save with a different file name use the 'Save as' button in the menu bar ('File' - > 'Save as')

3.9 Viewing Visualisations

Below is a list of visualisations and how to view them:

- Wiring diagram (see Section 4.1)

  Click the 'Wiring diagram' button (item 11 in Figure 3.1) or navigate to 'File' - > 'Wiring diagram'

- State traces (see Section 4.2)

  Click the 'State traces' button (item 12 in Figure 3.1)or navigate to 'File' - > 'Generate traces'

- State graph (see Section 4.3)

  Click the 'State graph' button (item 13 in Figure 3.1) or navigate to 'File' - > 'Generate state graph'

3.10 Troubleshooting Errors

On attempting to edit or add a function, if the command entered is invalid you will be presented with the error message *"The inputted command is invalid, please enter a valid command and try again"*.

3.10.1 Troubleshooting Boolean network errors

Given both synchronous and asynchronous share the same functions the possible errors are the same. Below is a list of possible errors that could occur:

- Incomplete command

  An incomplete command such as "A OR ", given that it is an invalid will cause an error. To fix ensure all commands are complete and valid

- Missing spaces

  The tool works on a command separated basis, therefore all commands require spaces. For example the command "AORB" is invalid. The correct command would be "A OR B".

- Use of commands

  The tools supports a multitude of different notations for boolean networks, to check the tool has recognised the notation you are using click on the view button (see Section 3.6). Additionally refer to Section 3.2 for the supported notation

- empty command or entity

  For an entity to be valid it requires a entity name or command

Chapter 4

Visualisations

4.1 Wiring diagram

4.1.1 General layout

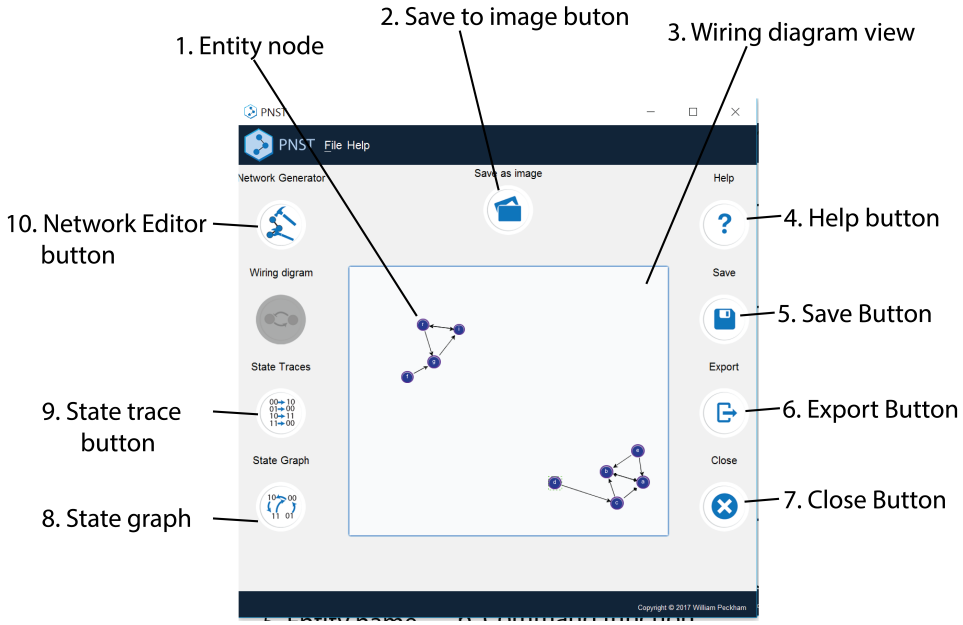


Figure 4.1: Annotated wiring diagram view

4.1.2 Viewing wiring diagrams

Wiring diagrams are displayed in the 'Wiring diagram' view (item 3 in Figure 4.1).

4.1.3 Changing the layout

All of the entity nodes (item 1 in Figure 4.1) in the wiring diagram (item 3 in Figure 4.1) can be manually moved. To move a entity click and drag the entity to the desired location.

4.1.4 Save as image

To save the wiring diagram to a PNG file click on the 'Save to image' button (item 2 in Figure 4.1). You will be presented with a save dialogue (see Figure 4.2) pick the location and name the file then click the save button.

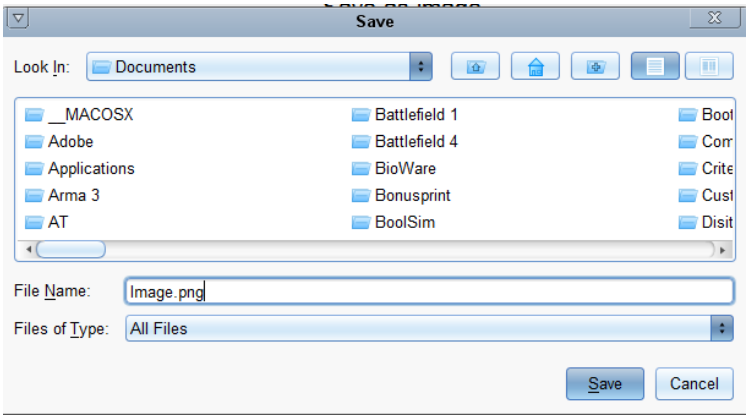


Figure 4.2: Save dialogue for saving to image

4.1.5 Saving the network

Saving the network will not save the layout of the wiring diagram, however it will save any changes made in the 'Network Generator' view.

To save a network click the 'Save' button (item 8 in Figure 3.1) or using the menu bar navigate to 'File' - > 'Save'. To save with a different file name use the 'Save as' button in the menu bar ('File' - > 'Save as')

4.2 State traces

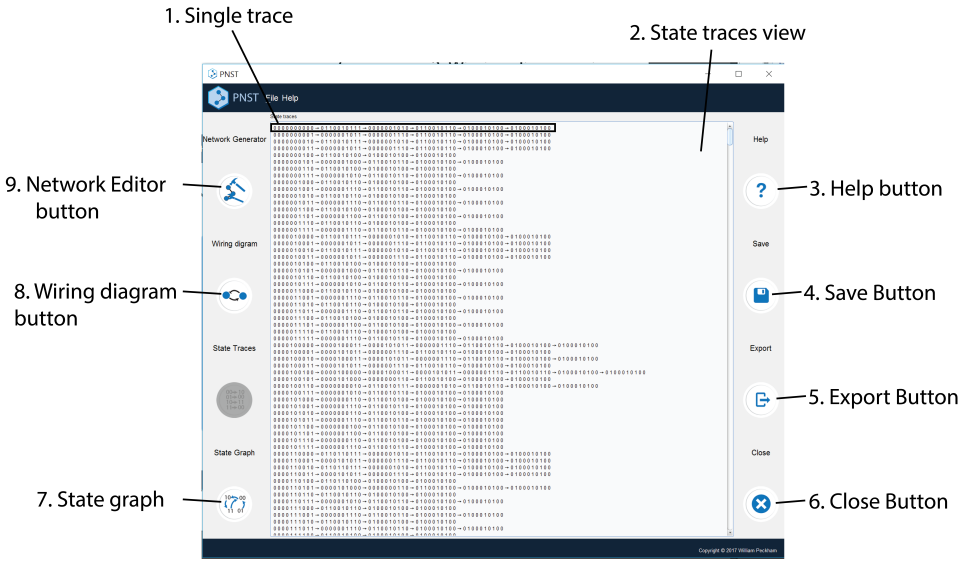


Figure 4.3: Annotated state trace view

4.2.1 Viewing traces

Traces are displayed in the 'State traces' view (item 2 in Figure 4.3), they can be directly copied from the view by highlighting and coping the traces you wish to copy.

4.2.2 Saving the network

Saving the network will not save the layout of the wiring diagram, however it will save any changes made in the 'Network Generator' view.

To save a network click the 'Save' button (item 8 in Figure 4.3) or using the menu bar navigate to 'File' - > 'Save'. To save with a different file name use the 'Save as' button in the menu bar ('File' - > 'Save as')

4.3 State graph

For hiding nodes, see Section 5.2.

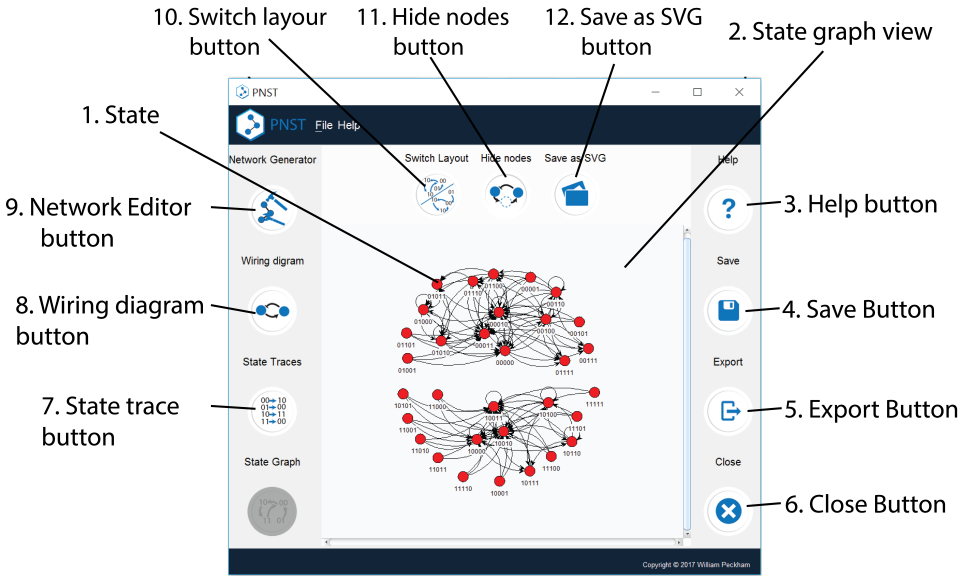


Figure 4.4: Annotated state graph view

4.3.1 Viewing state graph

The State graph is displayed in the 'State graph' view (item 2 in Figure 4.4).

4.3.2 Changing the layout

The layout style can be changed by clicking on the 'switch layout' button (item 10 in Figure 4.1), this will automatic re-organise the layout the nodes.

You can zoom in/out on the graph with the use of the mouse stroller.

4.3.3 Save as image

To save the wiring diagram to a SVG file click on the 'Save to image' button (item 2 in Figure 4.1). You will be presented with a save dialogue (see Figure 4.5) pick the location and name the file then click the save button.

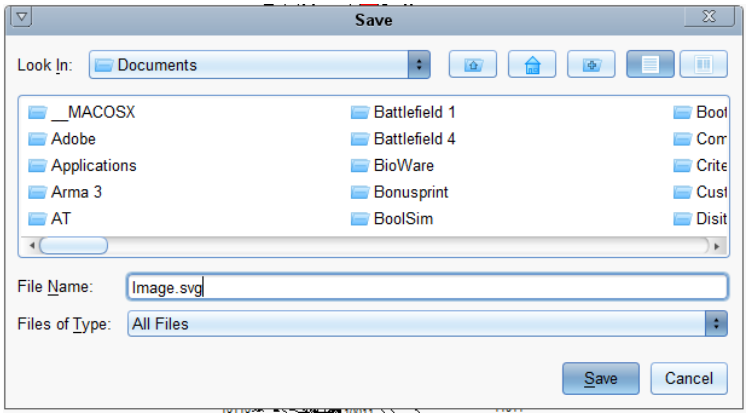


Figure 4.5: Save dialogue for saving to image

4.3.4 Export network to GraphViz

To export the state graph to a GraphViz format click on the export button (item 5 in Figure 4.4), you will be presented with Figure 4.6, to export to 'GraphViz' click the 'GraphViz' button and enter the desired name and click the save button on the save file dialogue.

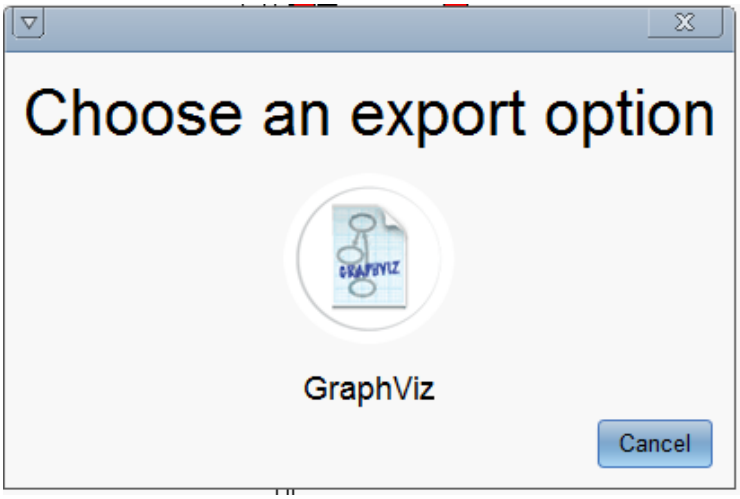


Figure 4.6: Export dialogue for saving to GraphViz

4.3.5 Saving the network

Saving the network will not save the layout of the wiring diagram, however it will save any changes made in the 'Network Generator' view.

To save a network click the ‘Save’ button (item 8 in Figure 4.4) or using the menu bar navigate to ‘File’ - > ‘Save’. To save with a different file name use the ‘Save as’ button in the menu bar (‘File’ - > ‘Save as’)

Chapter 5

Advanced features

5.1 Priorities

To add priorities to a asynchronous network (to create a priority-based asynchronous) first click on the priorities button on the 'Network generator' view. You will be presented with the dialogue shown in Figure 5.1.

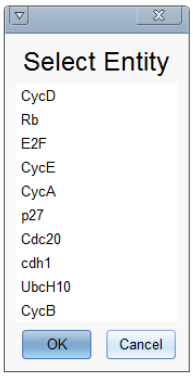


Figure 5.1: priorities dialogue with no entities selected

To add an priority click on the entity you wish to add priorities, which will display the Add and remove priorities section (see Figure 5.2).

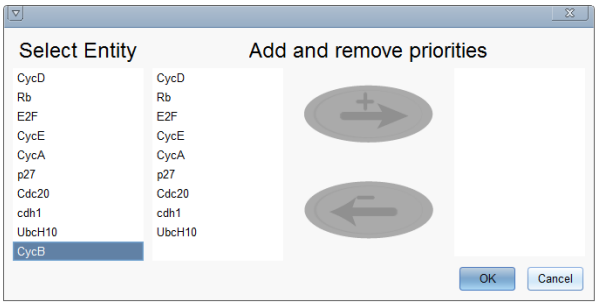


Figure 5.2: priorities dialogue with entities selected

Next select entities you want to add as lower priorities and click the add arrow to add the priority. This priority will be displayed in the current rules section.

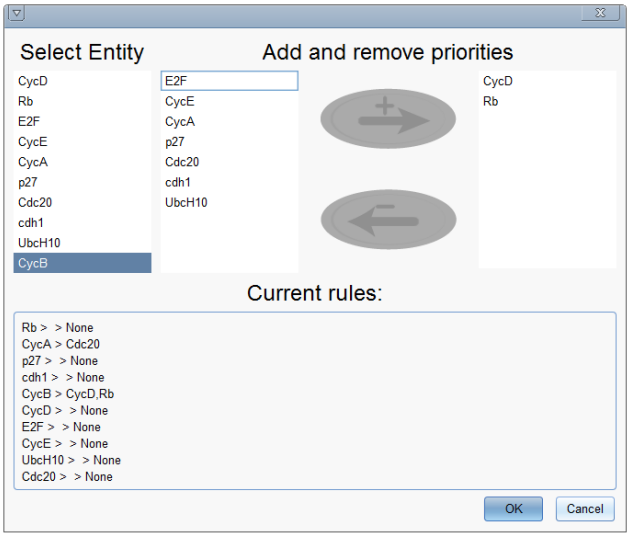


Figure 5.3: priorities dialogue with priorities added

To remove a priority select it and click on the remove arrow.

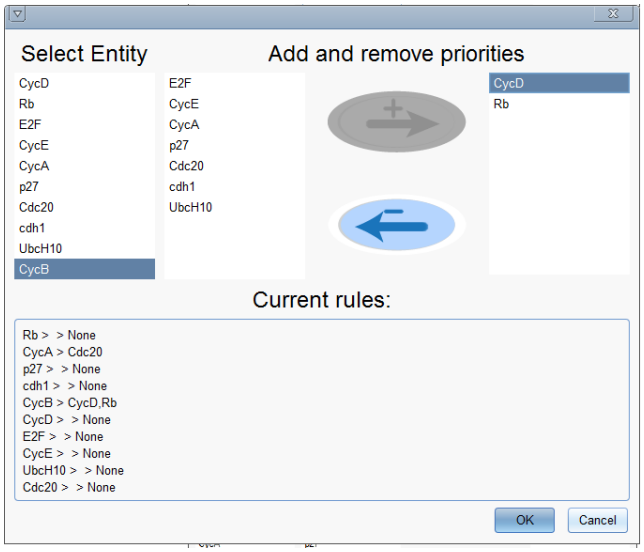


Figure 5.4: Removing priorities

Clicking ‘ok’ will then save the priorities.

5.2 Hiding nodes

To hide nodes on the state graph click on the Hide nodes button on the 'State graph' view (see Figure 5.5)

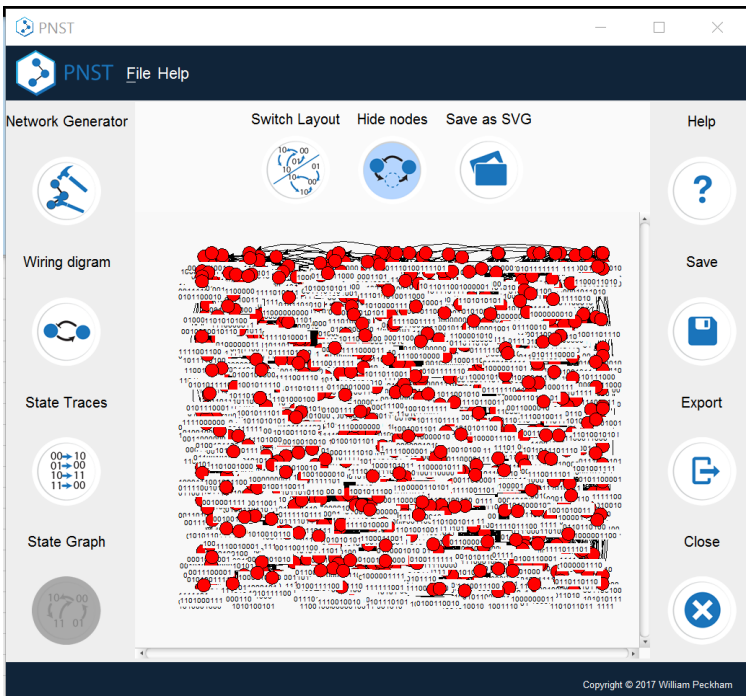


Figure 5.5: State graph view with hide node button highlighted

You will be presented with the dialogue shown in Figure 5.6:

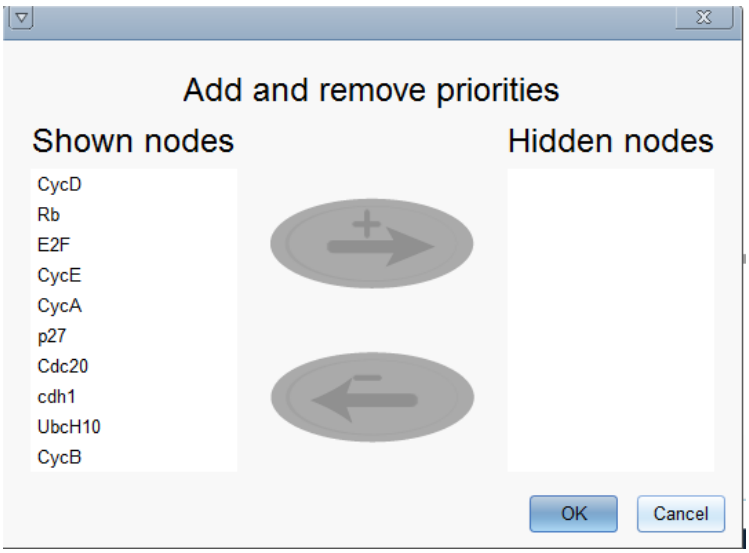


Figure 5.6: hide nodes dialogue with no entities hidden

To hide a node select the node you wish to hide and click on the add arrow, to move it into the hidden nodes.

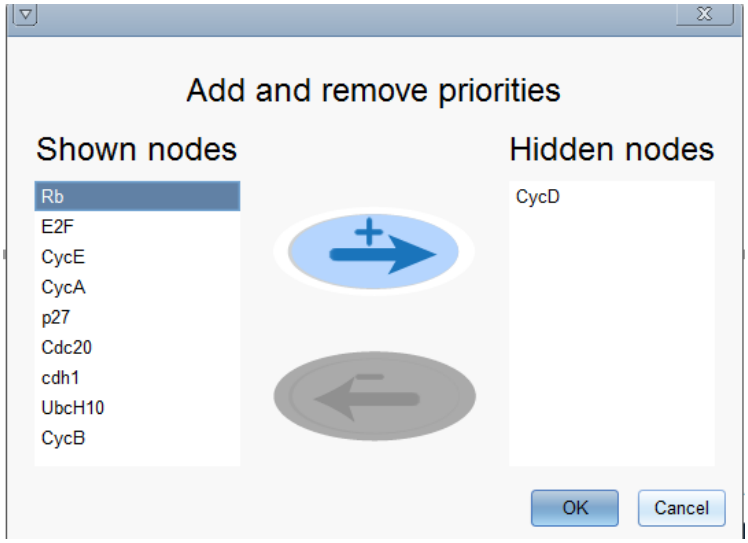


Figure 5.7: hide nodes dialogue - hiding a node

To un-hide a node select the node you wish to show and click on the remove arrow, to move it into the shown nodes.

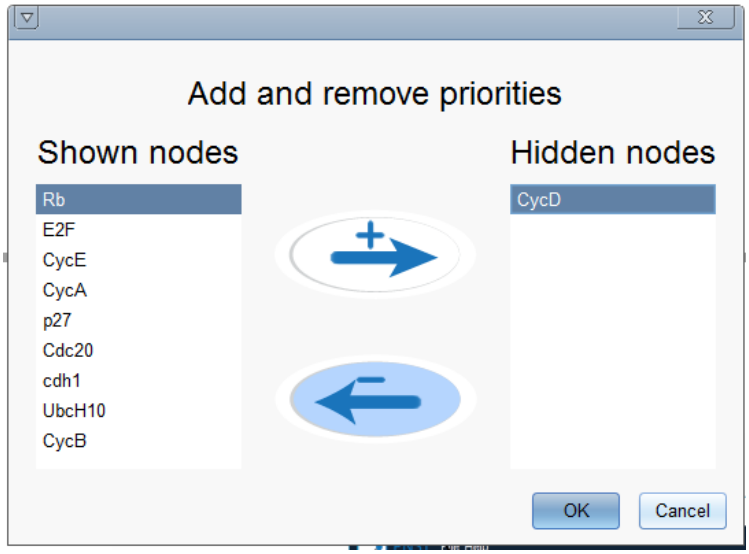


Figure 5.8: hide nodes dialogue - showing a node

Appendix H

Questionnaire

This Appendix section contains the user questionnaire performed that is referred to in the Evaluation section. The video used in the demo is available at : `http://youtube.com/watch?v=KAuKTsLLwIg` (will be left available until 01/01/2018)

PNST - Product Questionnaire

* Required

1. **How confident are you with using computer software?** *
Mark only one oval.

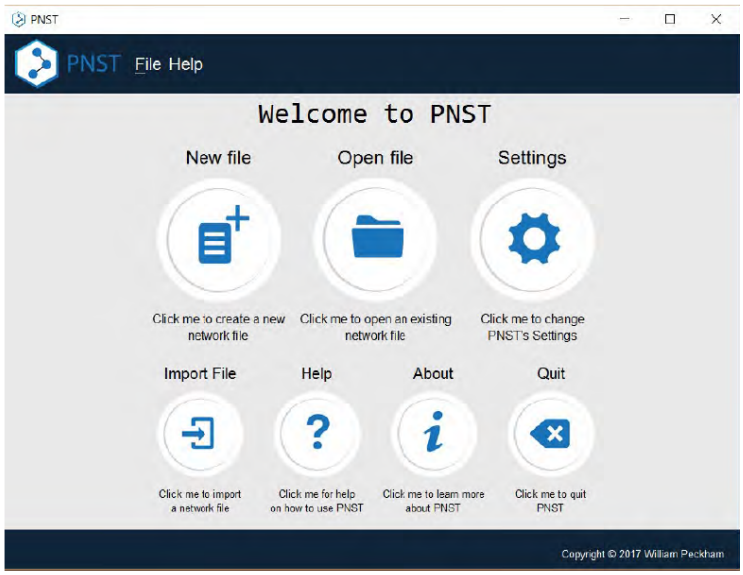
- ◯ None at all (Require help with simple tasks)
- ◯ Beginner (competent user of software such as word and powerpoint)
- ◯ Intermedient (User of more complex softwares, may require help on more compelx tasks)
- ◯ Expert (User of complex softwares on a regular basis)
- ◯ Other: _____

2. **Check those that are appropriate, Other section is optional (Note none are required to answer this questionnaire)**
Check all that apply.

- ☐ I study or have studied computer science
- ☐ I am a student
- ☐ I have prior knowloage about Boolean networks
- ☐ I have a good undstanding of GUI Design (HCI principles etc.)
- ☐ Other: _____

Initial reaction

PNST's home page



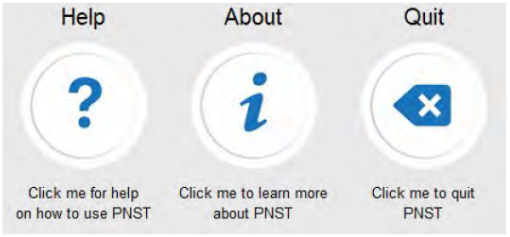
3. **What is your first reaction to the tool?** *
Mark only one oval.

- ◯ Very Positive
- ◯ Somewhat positive
- ◯ Neutral
- ◯ Somewhat negative
- ◯ Very negative
- ◯ Other: _____

4. **On a scale from 1 (very low) to 5 (very high), from your first look at the tool how would you rate the quality of the product?** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

5. **Here are some example buttons from the homepage. On a scale from 1 (not at all relevant) to 5 (extremely relevant) how well do you think their icons relate to the action we want to perform.** *



Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

On a scale of 1(terrible) to 5 (excellent) please rate the following features of our home page design

6. **General Looks/Style** *

Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

7. **Button Looks/Styles** *

Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

8. **Button Layout - I.E with core functionality on the top** *

Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

9. **Icons choice** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

10. **Colour Scheme** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

11. **Reading text on the screen** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

12. **Instructions of commands (under buttons)** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

13. **Our logo** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

14. **If anything, what would you change about our homepage?** *

15. **If anything, where there any features that your particularly liked?** *

Example Usage

The below Questions relate to a quick demo video (no sound required)

Quick Demo of features

http://youtube.com/watch?v=KAuKTsLLwIg

On a scale of 1(terrible) to 5 (excellent) please rate the following features of our design

16. **General looks of the system** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

17. **Comments**

18. **Creation of a new File** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

19. **Comments**

20. **Choosing a network type** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

21. **Comments**

22. **Adding new entitles (and functions)** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

23. **Comments**

24. **Feedback on errors in the commands** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

25. **Comments**

26. **Switching between views** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

27. **Comments**

28. **Clearness of what is the current view** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

29. **Comments**

30. **Modifying layouts (Wiring Diagram)** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

31. **Comments**

32. **Modifying layouts (State Graph)** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

33. **Comments**

34. **Adding priorities** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

35. **Comments**

36. **Hiding Nodes** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

37. **Comments**

38. **Saving and Opening networks** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

39. **Comments**

Answer the following with a short paragraph/sentence

40. **If I gave you the information about a network (a list of functions) Do you think you could create it in the tool? If not what would stop you?** *

41. **If anything, what would you change about the tool?** *

42. **If anything, what features did you like the most** *

43. **What now is your reaction to the tool?** *

Mark only one oval.

- ◯ Very Positive
- ◯ Somewhat positive
- ◯ Neutral
- ◯ Somewhat negative
- ◯ Very negative
- ◯ Other: _____

44. **On a scale from 1 (very low) to 5 (very high), how would you now rate the quality of the product?** *

Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

Finishing up

45. **Name (Optional)**

46. **Email (Optional)**

47. **Check if appropriate. I would be interested in you emailing about :**

Check all that apply.

- [ ] Taking part in additional questionaires about this tool
- [ ] Taking part in a hands on testing of the tool

44. **On a scale from 1 (very low) to 5 (very high), how would you now rate the quality of the product?** *
Mark only one oval.

1	2	3	4	5
⬭	⬭	⬭	⬭	⬭

Finishing up

45. **Name (Optional)**

46. **Email (Optional)**

47. **Check if appropriate. I would be interested in you emailing about :**
Check all that apply.

- [ ] Taking part in additional questionaires about this tool
- [ ] Taking part in a hands on testing of the tool

44. **On a scale from 1 (very low) to 5 (very high), how would you now rate the quality of the product?** *

Mark only one oval.

1	2	3	4	5
⬭	⬭	⬭	⬭	⬭

Finishing up

45. **Name (Optional)**

46. **Email (Optional)**

47. **Check if appropriate. I would be interested in you emailing about :**

Check all that apply.

- [ ] Taking part in additional questionaires about this tool
- [ ] Taking part in a hands on testing of the tool

Questionnaire Results

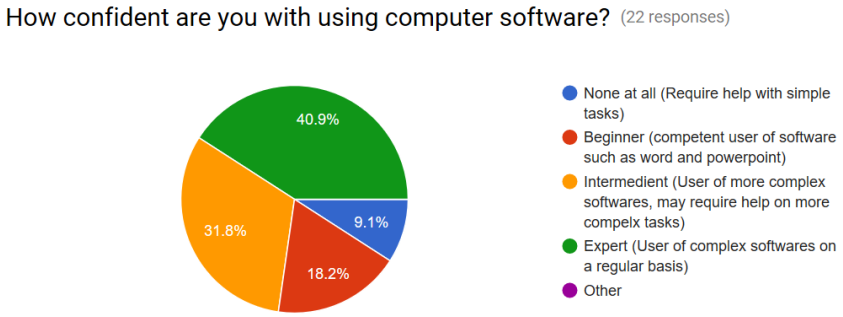


Figure I.1: Confidence of the user with use of software

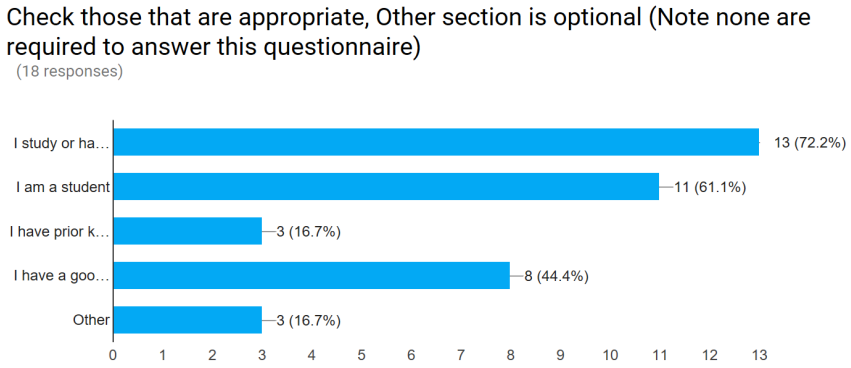


Figure I.2: User background

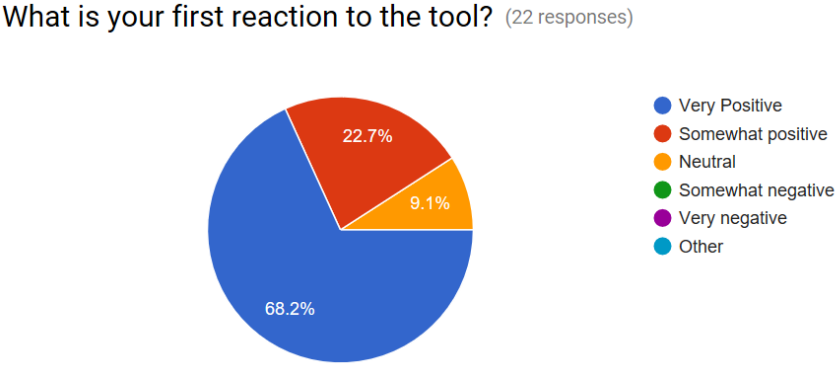


Figure I.3: Users initial reaction on viewing system for the first time

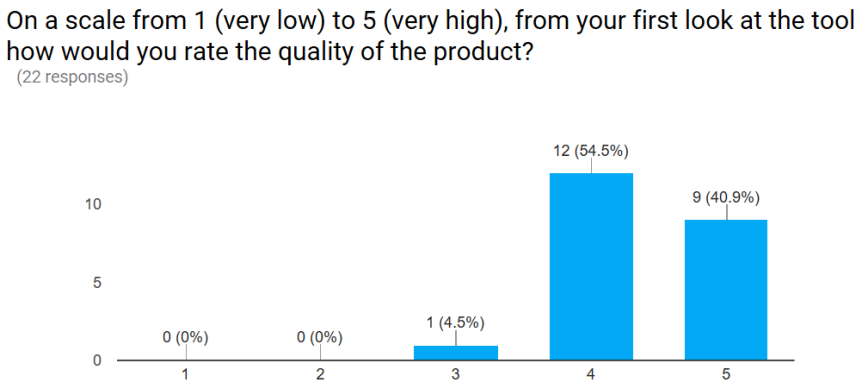


Figure I.4: User's first impression of the system quality

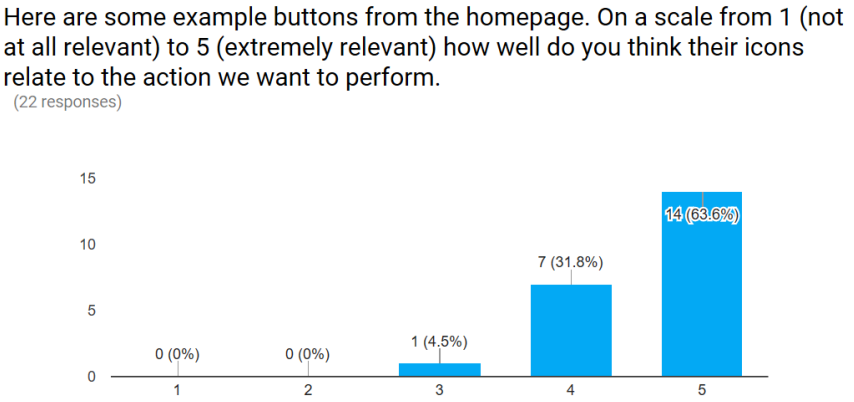


Figure I.5: User evaluation of logo usage

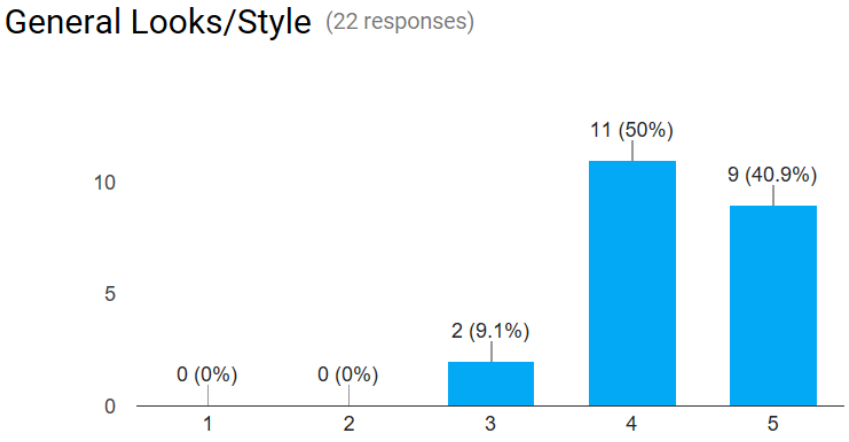


Figure I.6: User feedback on general looks

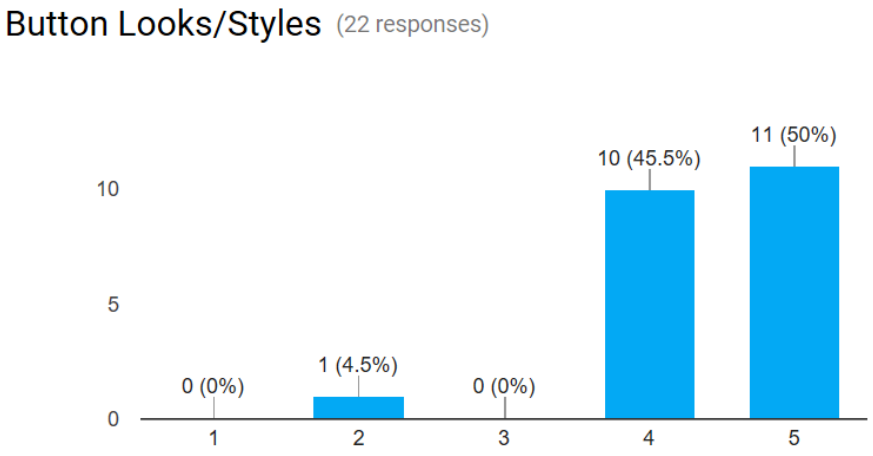


Figure I.7: User feedback on button looks and styles

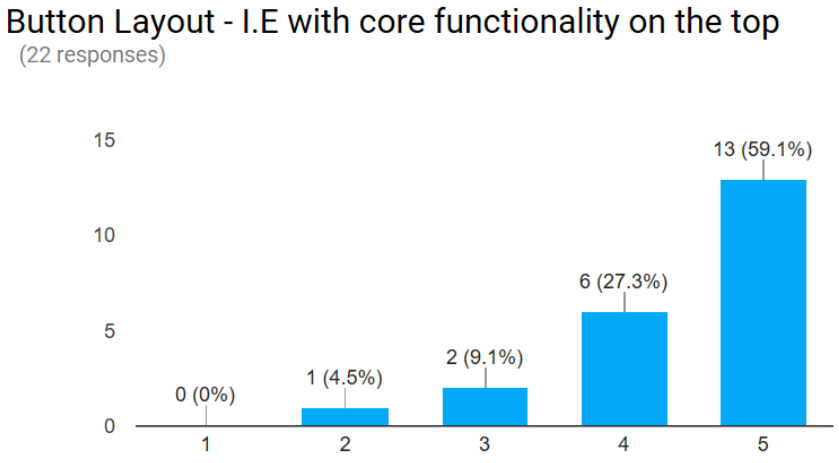


Figure I.8: User feedback on button layout

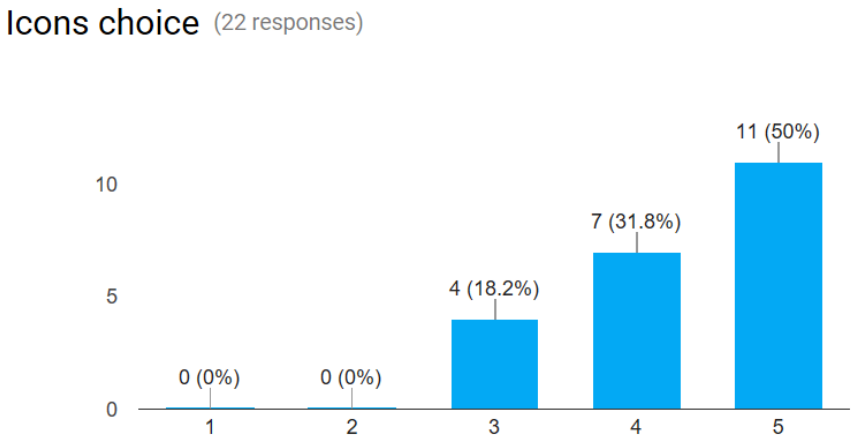


Figure I.9: User feedback on icon choices

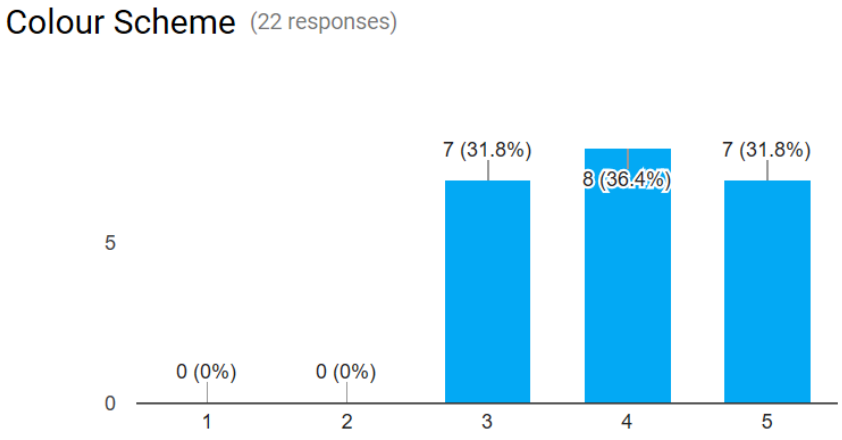


Figure I.10: User feedback on colour scheme

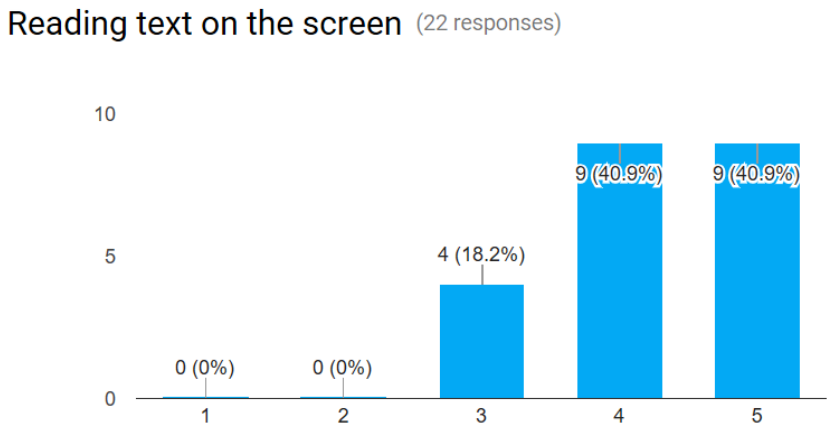


Figure I.11: User feedback on the readability of the text

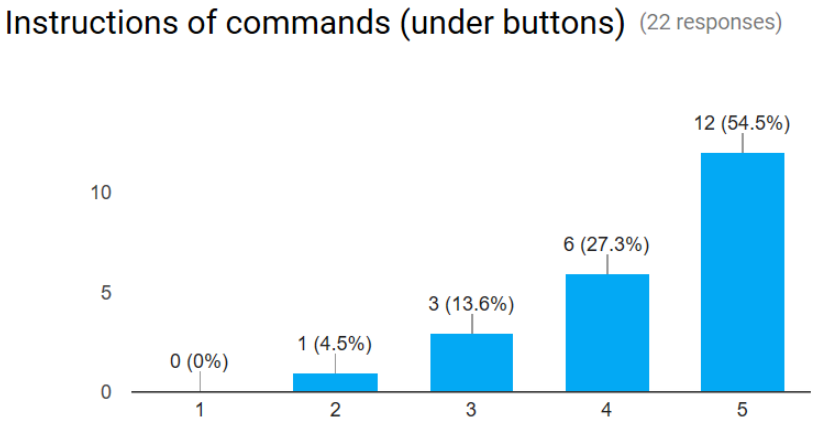


Figure I.12: User feedback on descriptions under the logos

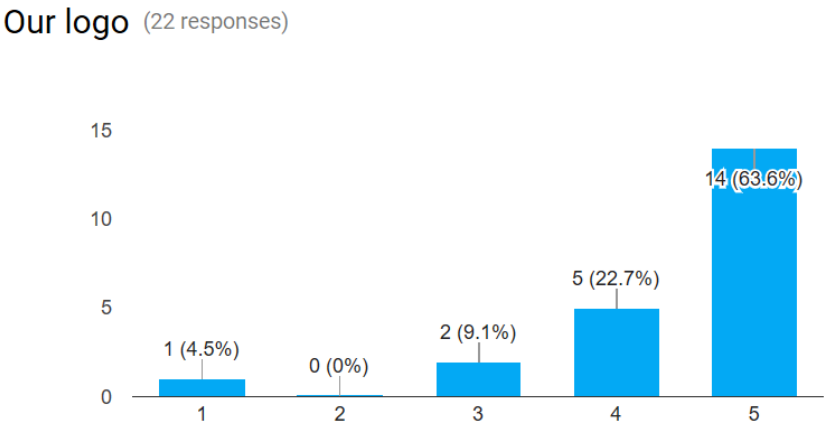
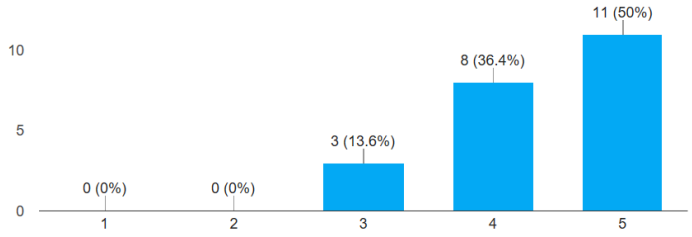


Figure I.13: User feedback on the logo used

General looks of the system (22 responses)

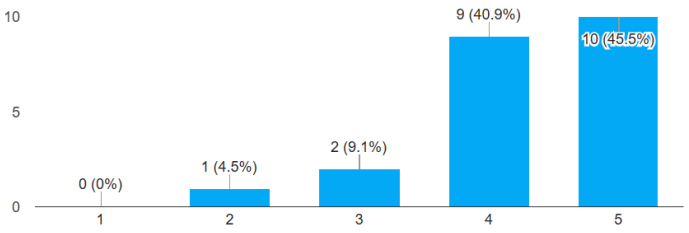


Comments (3 responses)

Great
Looks easy to navigate and use
It seems simple and self explanatory.

Figure I.14: User feedback on general looks (after demo video)

Creation of a new File (22 responses)



Comments (6 responses)

I don't understand because I am 16 BUT it looks very good and if I was a computer person I would think it was good
Video demonstrates a wide range of features that the system can complete.
All abut over my head
Above my level ,but, folowed it ok
Does the job
I think this would take me a little while to get used to as I've never used a program like this before.

Figure I.15: User feedback on creation of a new file (after demo video)

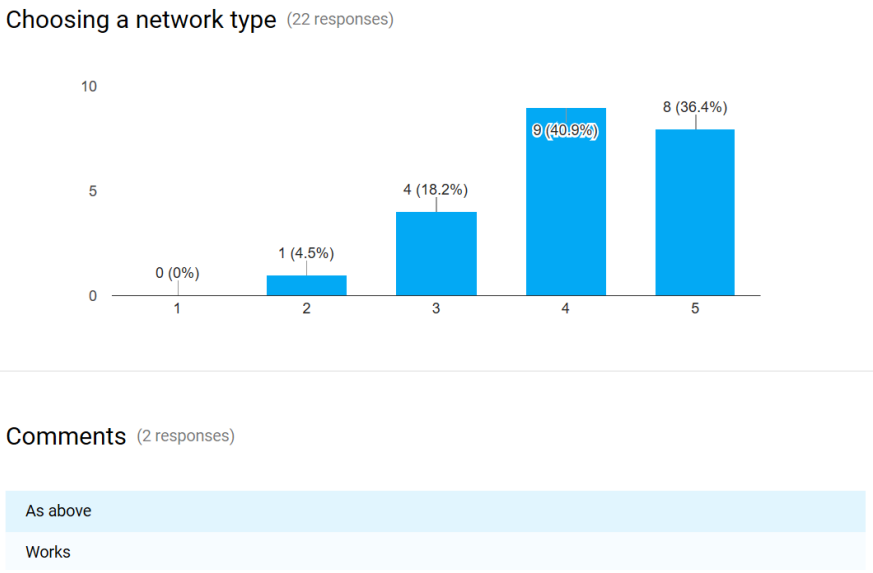


Figure I.16: User feedback on choosing a network type (after demo video)

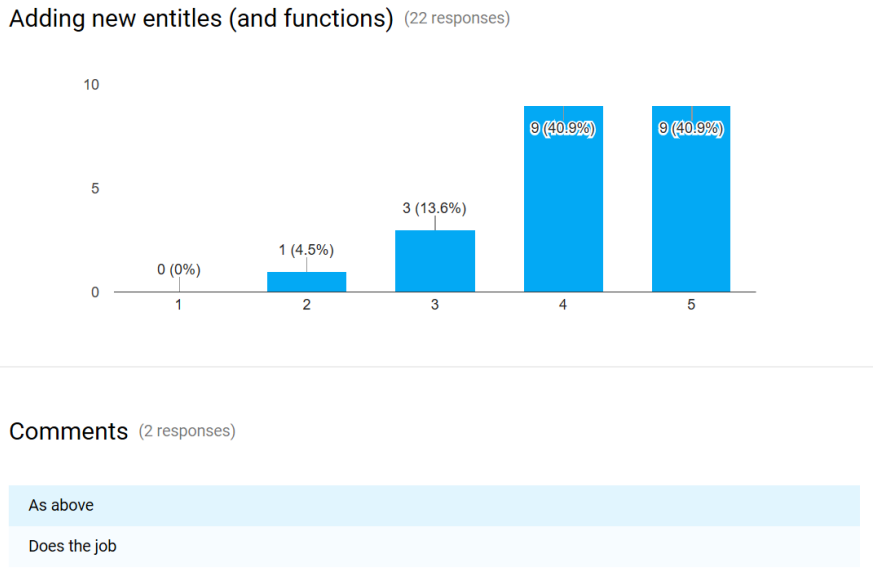


Figure I.17: User feedback on adding new entities (after demo video)

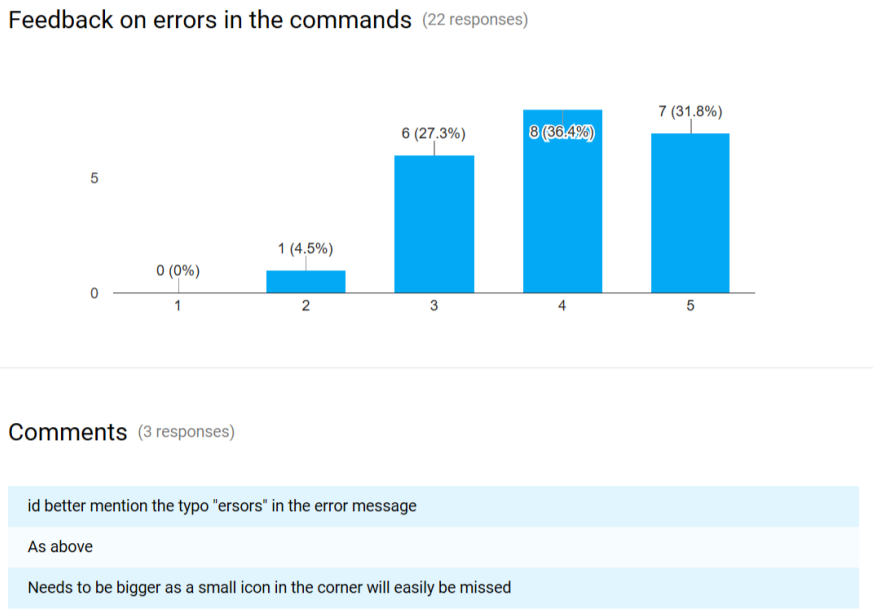


Figure I.18: User feedback on Error messages produced (after demo video)

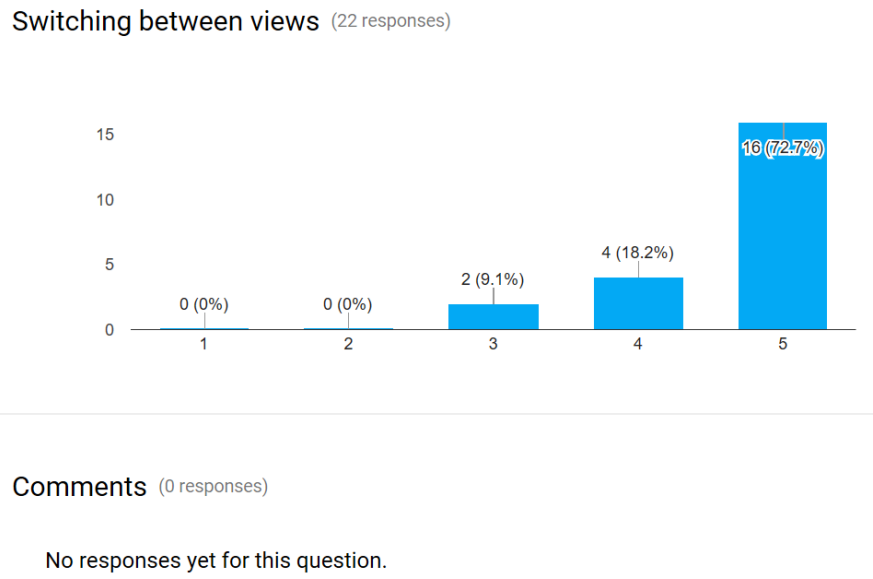


Figure I.19: User feedback on switching between views (after demo video)

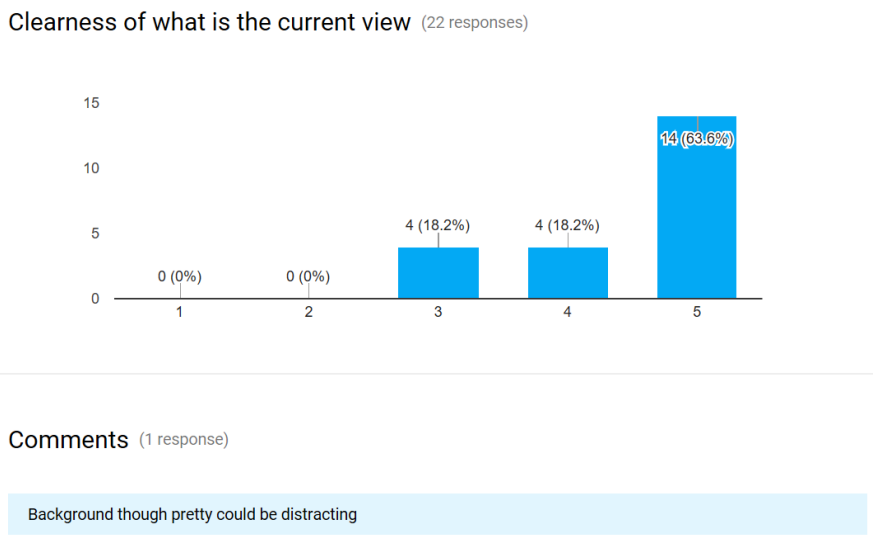


Figure I.20: User feedback on clearness of the current view (after demo video)

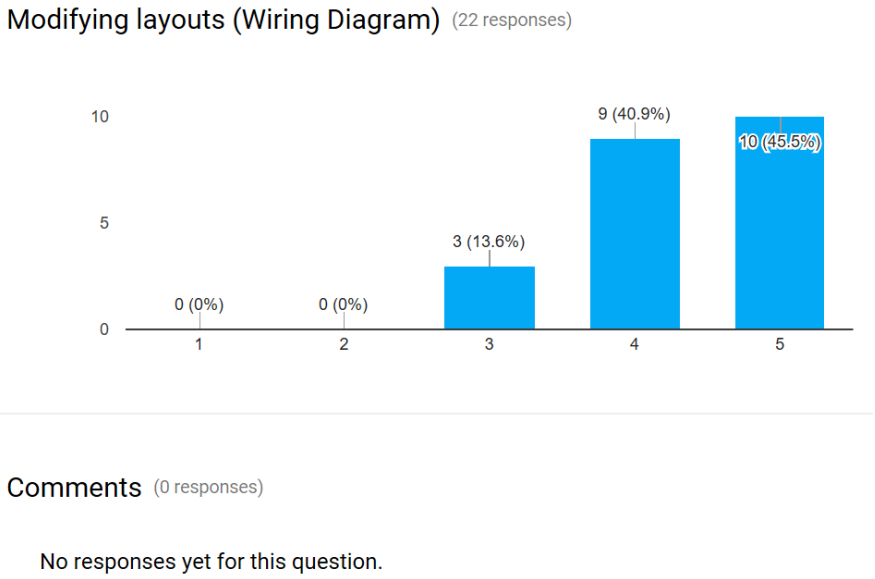


Figure I.21: User feedback on modifying layouts of the wiring diagram(after demo video)

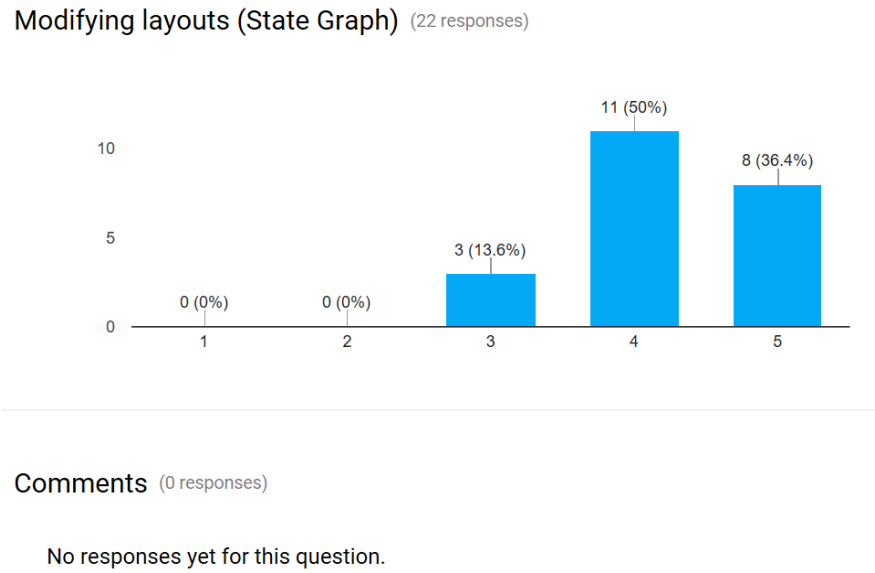


Figure I.22: User feedback on modifying layouts of the state graph (after demo video)

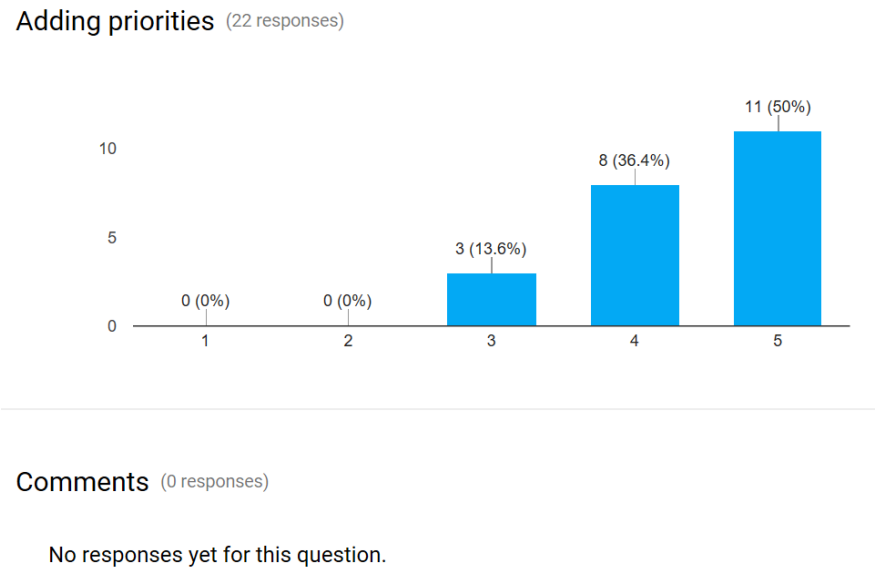


Figure I.23: User feedback on adding priorities to the network (after demo video)

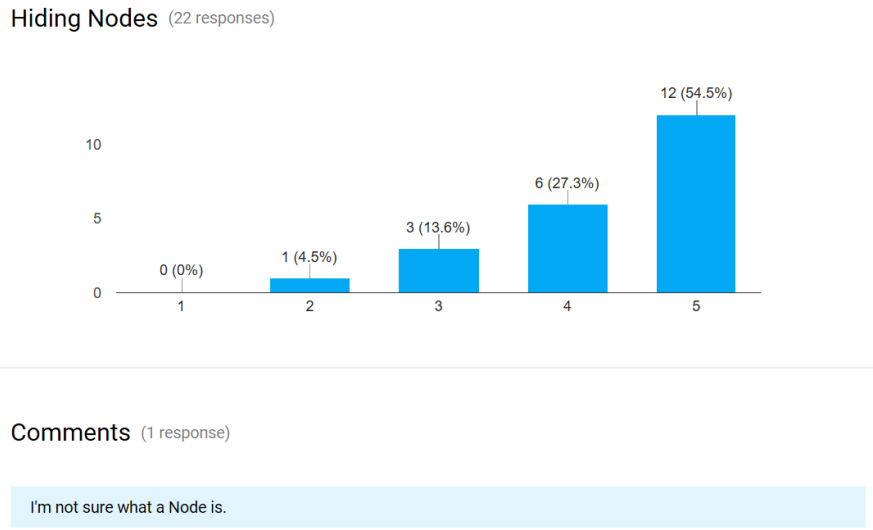


Figure I.24: User feedback on hiding nodes (after demo video)

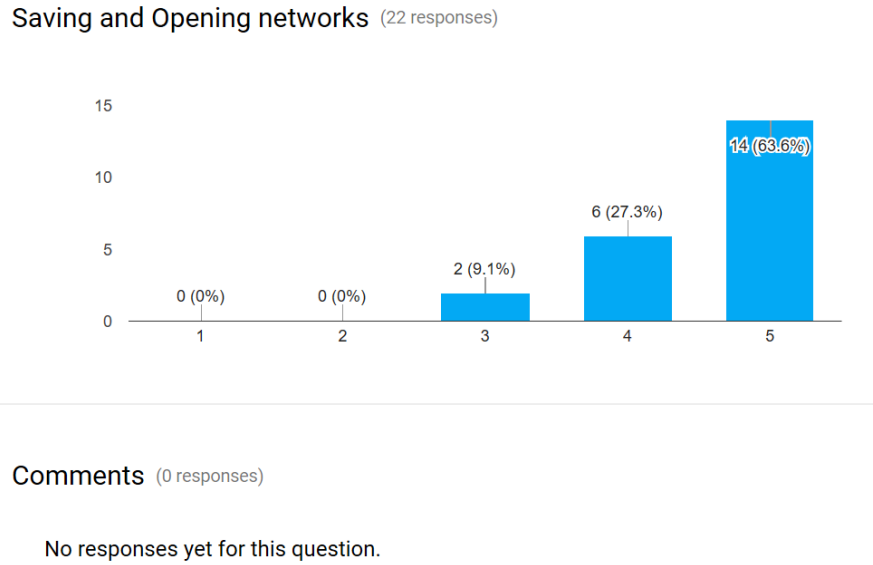


Figure I.25: User feedback on saving and opening networks (after demo video)

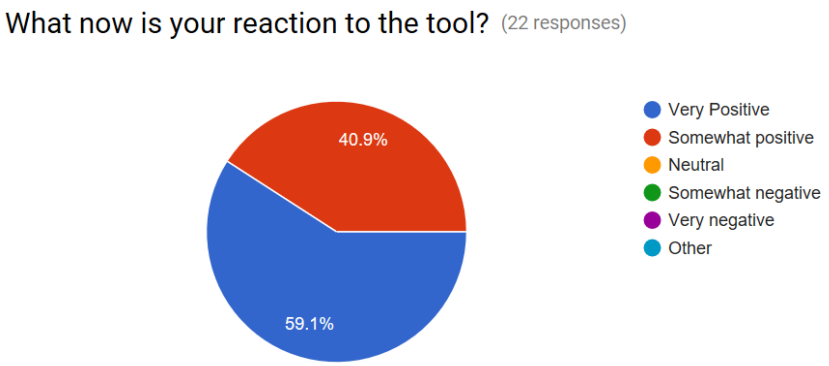


Figure I.26: User final reaction to the tool (after demo video)

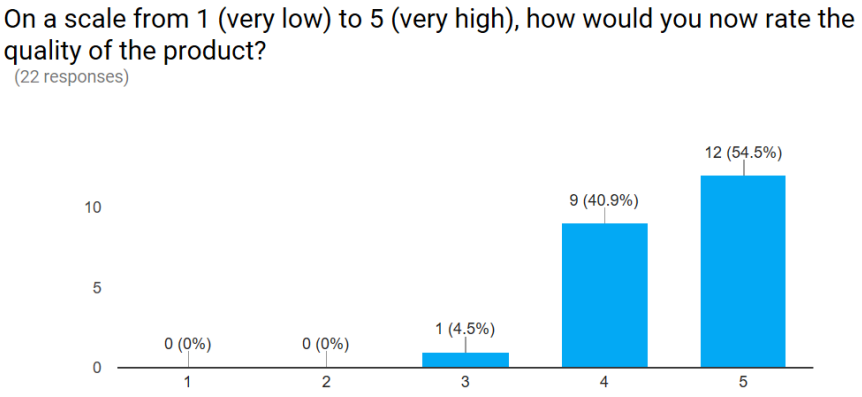


Figure I.27: User feedback of impression of quality (after demo video)

Appendix J

Fulfilment of requirements

As part of the evaluation of the system, testing was performed on the requirements of the system, the purpose being to determine the success of implementing the requirements set. The Table J.1 shows a quick overview if and how each requirement was implemented:

Table J.1: Requirements fulfilment table

Requirement identifier	Name	Justification if was or was not completed
F1	New network creation	This requirement was met; through the new button on the home screen (Figure F.1) a user is presented with a choice of networks (Figure F.2), on selecting a choice a new network is created.
F2	Save networks to file	This requirement was met; a user has the ability to save a new or opened network to file (via the save button in F.6)
F3	Open existing files	This requirement was met; a user can open an existing file that was previously saved (see Figure F.9)
F4	Edit existing files	This requirement was met; on opening a file a user can edit the network.
F5	Function manipulation	This requirement was met by the network editor (Figure F.6)
F6	Wiring diagrams	This requirement was met; a user can view the wiring diagram of network opened (see Figure F.3)
F7	State traces	This requirement was met; the tool provides the ability to view traces of a valid given network (see Figure F.16)
F8	State graph	This requirement was met; the system will generate a state graph for a given network F.13
F9	Export networks	This requirement was partially met. A network can be exported to image through the state graph and wiring diagrams. It can also be exported to a GraphViz formate from the state graph. It does not however support exporting to other network support tools.

F10	Import networks	This requirement was partially met. The tool provides the user with the option to import a JASON Based Graph-Exchange Formate file (jSBGN), However does not support importing from all possible sources.
F11	Hiding nodes	This requirement was met through the hide node function, whereby a user can hide nodes on the state graph (see F.13 to Figure F.15)
F12	Priority networks	This requirement was met through the network editor that provides the ability to add priorities to a asynchronous network network (see F.10)
D1	Long term storage	This requirement was met with networks being stored as XML file to disk
D2	Network type	This requirement was met through the use of an ID which is stored within the XML file
D3	Network entities	This requirement was met with the system storing the entity name within the XML file
D4	Network functions	This requirement was met with the storing the entity name's command within the XML file used to save/load networks
D5	State	This requirement was met, the system can store the list of traces for a given network, (However does not store long term)
D6	Traces	This requirement was met the system can store the state transitions for a given network, allowing it to generate the state graphs
E1	User guide	This requirement was met, accessed by the tool via the help button and provided in Appendix G
E2	Platform Compatibility	This requirement was met, the system was tested on a Windows PC, and all tests passed.