A Tool for Visualising Cell Model Results
MInf Project Phase 2

James Hulme

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2014

Abstract

This project was the creation of a tool to help biologists visualise and analyse their data better. The project was developed in two phases. This document details the second phase. In this phase of the project: animation of cell data was added, along with functionality for annotating visualisations, normalising and exporting data, using plots as queries, and real time collaboration. The project was evaluated using a mixture of user feedback, relevant metrics, and experimentation. The findings indicated that the tool that was developed was an effective tool which fills a gap left by the currently available software.

Acknowledgements

Acknowledgements go here.

Table of Contents

Chapter 1

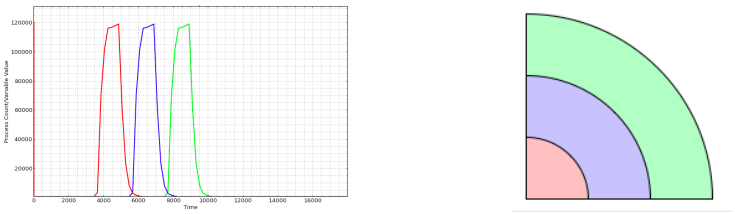
Introduction

Biologists often use computer models to help guide their research as modelling is much cheaper than experimentation. There are a number of tools available for biological modelling. These tools typically require a certain level of numerical confidence to create and interpret. Not all biologists have this numerical confidence. Some researchers find writing and interpreting models a challenge; this can make them less effective in their research. It is therefore necessary for the tools they use to help them relate the data to their field by incorporating domain knowledge.

One such tool that can be used for modelling is Bio-PEPA [1], an extension of the PEPA process algebra. Bio-PEPA is currently implemented as a plugin for the Eclipse IDE. Bio-PEPA visualises the model results as time-series graphs. There is one team of Src researchers in particular, who use Bio-PEPA and do not have the numerical confidence, as described above, to be comfortable using Bio-PEPA. This team was the original focus for the project. The purpose of this project was to extend Bio-PEPA's visualisation capabilities to allow the previously mentioned team, and other similar users of Bio-PEPA, to more effectively analyse their results.

A significant problem with Bio-PEPA's visualisation capability is that it is difficult to represent spatial change on a time-series graph (as can be seen in Figure 1.1). In Bio-PEPA you can have a species at different locations in the cell, for example, next to the nucleus, next to the cell membrane and throughout the cytoplasm. The movement of this species through the cell can by modelled by seeing the population of it in each location over time. This is difficult to visualise on a time series plot as three lines are too abstract. It requires the use of a biological metaphor for spatial information to be easily interpreted. In this case using a visualisation based on a cell can more intuitively show how the species moves through the cell. It is this type of inference that the Src researchers find challenging to do with Bio-PEPA currently.

Over the course of the project the scope has been expanded. The original objective was to assess which forms of visual representation are most helpful and informative to laboratory science. At the end of the first project phase the objective changed (to reflect that the project was about delivering a finished program and not about the results of many experiments) to be to develop a tool to visualise the results of dynamic



(a) Time Series Representation (b) Spatial Representation

Figure 1.1: One species at three locations in the cell represented traditionally on a time series graph and also spatially in a cell

time-series models of intra-cellular behaviour based on biochemical reactions. This objective was focused on visualisation to aid those researchers who are not numerically confident. The second phase of the project has added to this objective to also aid interpretation and collaboration. This change in objective is to make the tool better for all users. To reflect these changes the focus on the original Src team has been generalised. Evaluations are performed with other potential users who are also biologists. Another factor in deciding to focus more on these other potential users was one of the key Src researchers going on maternity leave during the second phase of the project. This made it infeasible to evaluate the tool with them. It had been hoped to still evaluate the tool with them at the end of the second phase of development but they were still on maternity leave.

1.1 This Document

This document details the work that was undertaken during the second phase of this project (MPP2). This is split into separate chapters.

Chapter 2 discusses the goals that had been identified and the new goals that have been added. This discussion includes what goals have been refined and why certain goals have been dropped.

Chapter 3 discusses the implementation work that has been performed as part of this project phase. The discussion includes challenges faced and solutions for those challenges.

Chapter 4 discusses the evaluation of the project. This is spread over multiple evaluation sessions and includes evaluations with potential users as well as self evaluations.

This chapter, the introduction, discusses the aims of the this project and gives a brief overview of the results. The current landscape of similar software is evaluated. There is also a brief introduction to the major external libraries that were used in this project.

The work that was implemented in this phase of the project consisted of:

- Animation of species moving through a cell.
- Enabling annotation of the visualisations.

- The ability to normalise and export the results data.
- A system for using plots as queries against a database of plots to obtain a similarity ranking.
- Enabling real time collaboration between two instances of the tool.

1.2 Where Does This Tool Fit In?

In the first stage of the project a review was performed of the features of a number of modelling and visualisation tools. This review included specialised software aimed at biologists and general software for anybody doing data visualisation.

As the project scope expanded to include goals not specifically related to visualisation it was prudent to perform another software review covering the new features, in particular software that allowed for collaborative editing. It is important to see what features are commonplace, which features are not commonplace but are useful, and which features are not useful. This analysis would then be used to guide development of the collaborative features of the new tool.

Four products were chosen for this second review: Google Drive [2], Pidoco [3], Lucid Chart [4] and SynBiowave [5]. Analysis of these software can be found below.

1.2.1 Previously Reviewed Software

The software packages reviewed at the start of the project were: Bio-PEPA, Uppaal [6], V-Cell [7], Cell-O [8], Copasi [9], Cell Designer [10] and WEAVE [11]. Bio-PEPA, V-Cell, Cell-O, Copasi and Cell Designer have been written for biological modelling. Uppaal is modelling software for general use, and WEAVE is a general data visualisation tool. A review of these tools can be found in the report for the first phase of development (MPP1).

All offered some level of visualisation, some simply graphs, others more complex visualisations. Bio-PEPA offered only line graphs. Uppaal had visualisations that highlighted where in a finite state machine view of the model the current state is. V-Cell had visualisation of the model in a hierarchical set of circles. It could also display spatial elements of the results data by displaying a heat model view of the cell. Cell-O was aimed more at multi-cell models and was able to show them moving and splitting; it also had visualisation of the model as a finite state machine. Copasi only had graphs, although the user had more control over the display of the plots. WEAVE had the largest visualisation capacity being able to display a variety of standard graph types along with more interesting ones, such as geographical maps, but it did not appear to have anything specialised for biological models. WEAVE is also the tool that gave the user the most control over the visualisation.

The existing biological modelling software seems to be focused more on the ease of modelling. The visualisation features on offer are typically quite basic. They also lack

the more innovative features that can be found in the newer general data visualisation software.

1.2.2 Google Drive

Google Drive which was previously Google Docs, is one of the most widely used collaborative pieces of software. Google Drive is used by a variety of user types. The focus of the review was on the word processor. Of the collaborative software reviewed this was felt to be the most user friendly. One of the nicest features was a cursor indicating where every user currently editing the document is typing. Each user is also given a different colour allowing you to know who is doing what in real time. Many people can edit a single document at once. As well as editing documents users can also leave comments on the document. Changes made by users appear near instantly to all users. The speed of editing is very important as it is frustrating as a user to have to wait to see changes others are making. It is also very easy to invite others to edit the document with you. Each document has a unique link and if a user visits that link they are taken to the document and can start editing. Different permissions can be granted to different users allowing some level of collaboration with people who you don't necessarily want to grant full write access. These users can then just look at it in real time and offer comments. Different parts of the editor have different levels of collaboration. All text that is changed is changed for all, but preferences like font choice are only changed for the user, unless another user edits any text. Also importantly from a User Experience (UX) perspective is that any conflicts that arise appear to be resolved without any user intervention. A history of what each user has done to the document is also provided and any unwanted changes can be rolled back.

1.2.3 Pidoco

Pidoco is a collaborative wireframing tool. It was not as user friendly for collaboration as Google Drive. Pidoco is much less instant. Sometimes manual refreshes were required to display the work the other users had done. Pidoco also supported multi user editing, however there was no way of seeing which users were editing a document. There was also a history of what changes each user had made. Pidoco has no messaging or commenting system which makes asynchronous collaboration more difficult. Sharing the work requires an email to be the sent to the user: they cannot simply be given a link. Again different parts of the workspace have different levels of collaboration. Any User Interface (UI) widgets that are placed are shared, but if one user zooms in on a particular area other users are not forced to that zoom level.

1.2.4 Lucid Chart

Lucid Chart is a collaborative diagramming tool. Lucid Chart lies between Google Drive and Pidoco in terms of collaboration speed. It is not as instantaneous as Google

Drive, but it does not require periodic manual refreshes like Pidoco. It also allows multi user collaboration and documents can be shared by link or email. Users can be granted read or read and write permissions on the document, like Google Drive, so you can collaborate with people you don't want to be able to fully edit. It has a chat system and users can comment on the document making asynchronous collaboration easier. Lucid Chart does not let you see what a user is doing in real time, but it does provide a full revision history so you can see what changes each user has made. Lucid Chart is different in that it appears to be fully collaborative. Even font preferences are synced across users: if one user clicks bold all users will start typing in bold.

1.2.5 SynBioWave

SynBioWave was a biological extension to Google Wave. It is the only example found of a platform for real time collaboration. Unfortunately it could not be made to work. The instruction page still refers to Google Wave. Google Wave has been discontinued since 2012 (some of its real time features found their way into Google Docs). An attempt was made to follow the instructions using Apache Wave (Google Wave was handed over to the Apache foundation) but the SynBioWave tool did not work. The user guide does seem to give a good indication of how the tool would work. The review is based on this. Wave was used to enable the collaboration. You would create a new wave and add your human collaborators. You would also add the SynBioWave agent as a collaborator. When the SynBioWave agent is added to the wave it adds a menu bar. Through this menu the human collaborators can call various functions; the agent then calculates the result and displays the visualisations. The agent and its results can be interacted with in real time. SynBioWave is modular and there are multiple agents available. Custom agents can also be created. The use of Google Wave allowed the biologists to focus on implementing domain functionality without having to worry about the code infrastructure for real time collaboration. Unfortunately, by relying on a third party for the backbone of the system, SynBioWave appears to have become unusable after Google Wave shut down.

1.2.6 Findings

From studying the alternative modelling tools available and their visualisation capabilities it was clear that the visualisation capabilities of Bio-PEPA had to be increased. It was also clear that all the modelling tools were limited in their visualisation capability and that increasing the interactivity and customization available would be very useful.

We can see that there are a number of tools now that offer the ability to collaborate in real time. Only one, however, could be found that was focused on synthetic biology. This one tool appeared to now be obsolete. Given how useful tools like collaborative document editors have been more commonplace it seems like a good idea to try and bring this functionality into the systems biology domain. This would give Bio-PEPA a competitive advantage and hopefully encourage more researchers to use it.

1.3 Previous Work

Early on in the first stage of the project it was decided to separate this project from the Eclipse plugin. It was felt that Eclipse is not the right tool to do data visualisation in.

The initial development stages were focused on getting the new tool from having zero functionality to matching the visualisation features of the Eclipse plugin. This involved writing an early version of the UI, parsing the Bio-PEPA results data, and plotting it using `matplotlib` [12].

The next stage of the project was to extend the functionality. The first new feature was intensity plotting where the colour of the line increases in intensity/opaqueness as the population of the species increases. The next feature added was visualisation of the model. Model visualisation used a system of nesting circles and rings to build a hierarchical view of the cell from its model components. Finally the user was given control of the plot, allowing them to alter whether lines are plotted or not, what colour the line is and the thickness of the line. Figure 1.2 shows how the main screen of the program looked at the end of the first stage of development (MPP1).

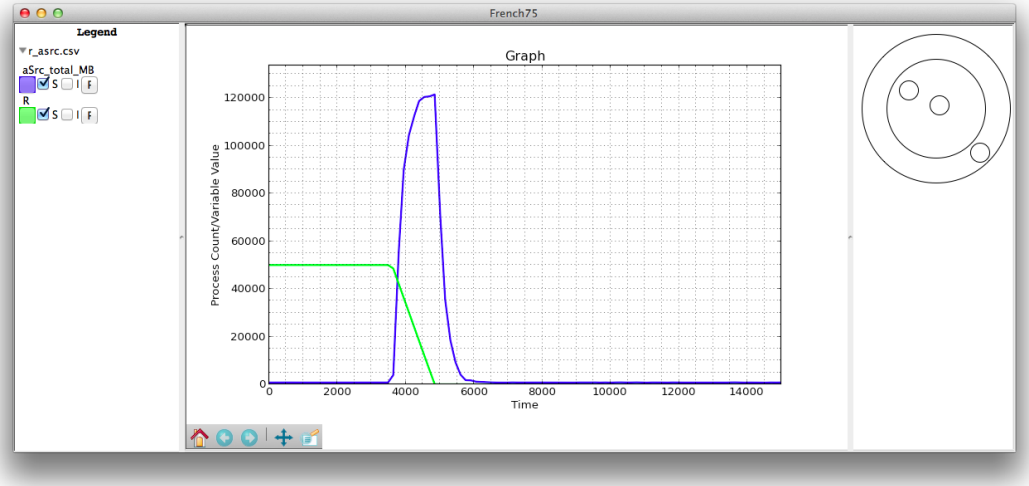


Figure 1.2: Main Screen of the Tool at the End of the First Stage of Development

Over the course of the first stage of work a number of evaluations were carried out with potential and actual users of Bio-PEPA. The findings from these evaluations were used to improve the tool.

1.4 Results

The tool that has been developed for this project is a fully featured tool with innovative features. It has been positively received by potential users. The tool offers more visualisation and analysis capabilities than the Eclipse plugin. Users can fully customize

the appearance of the visualisation. There is also the ability to add supporting information to the visualisation with annotations and the ability to attach files. Sessions can be saved and loaded allowing users to spread their work over time.

There are also features aimed at making it easier for biologists who aren't comfortable analysing graphs to analyse their work. This has been implemented as animated cell cross-sections simulating the movement of species through the cell.

Innovative features include using plots as queries to find similar plots allowing for discoverability from within the tool, and real time collaboration that allows two collaborators to work together and see each others changes in real time.

Their is focus on ease of use in the finished product with all features made as easy as possible to use with little guidance. Relying on recognition not recall.

By implementing more advanced visualisation features this tool should make Bio-PEPA a more attractive modelling tool. The visualisation features have been brought into line with the features in alternative modelling tools.

1.5 Libraries Used

This section details the libraries that were used to build this project. They are all Python libraries. Some libraries have been more important than others. This is detailed below.

1.5.1 matplotlib

`matplotlib` [12] is a graphing library for Python. The graph visualisation is built on top of `matplotlib`. The graph visualisation acts a wrapper around `matplotlib`. The user does not have to write scripts that make calls to `matplotlib`.

Features built on top of `matplotlib` are:

- Intensity Gradient Plotting
- Annotation infrastructure
- Data normalisation
- Line preferences

1.5.2 wxPython

`wxPython` [13] is the Graphical User Interface (GUI) library. The GUI and the animation visualisation are built on top of `wxPython`. The animation visualisation uses `wxPython's` drawing Application Programming Interface (API). This is detailed further in Sections 3.2 & 3.3.2

Features build on top of `wxPython` are:

- The UI
- Animation visualisation
- Animation annotations

1.5.3 SimpleXMLRPCServer

`SimpleXMLRPCServer` [14] has been used to enable communication between instances of the program. `SimpleXMLRPCServer` was used to avoid having to write client and server code that would send and receive messages over sockets. This allowed the development to focus on implementation of the API

Built on top of this library is

- Real time collaboration

1.5.4 Singleton

A singleton was used to act as a central repository for the data that visualisation session needs. A singleton was useful for this as only one instance of this repository was required. Pre-existing code was used for these to save time.

Over the course of the project two different singleton implementations were used. The first was taken from Paul Manta in this StackOverflow post - `http://stackoverflow.com/questions/6760685/creating-a-singleton-in-python`. This was not thread-safe. Towards the end of the project a different implementation was chosen. The code for this second singleton can be found here: `https://gist.github.com/werediver/4396488`. This implementation was threadsafe.

1.5.5 Miscellaneous

A number of other Python libraries were utilised in the project. They are all part of the standard Python library. They did not provide any significant relevant behaviour, instead they were used to aid implementation of the features in this project. More significant discussion of how any of these libraries were used can be found around the discussion of the features that utilised them. The libraries are: `uuid`, `re`, `platform`, `os`, `time`, `threading`, `subprocess`, `copy`, `random`, `numpy`, `itertools`, `collections`, `math`, `pickle`, `sys`, `inspect`.

Chapter 2

Goals

To help guide development during the project a number of goals were identified. Over the course of the project the goals have been refined, new goals have been added and some goals have been dropped.

2.1 Previous Goals

Through research into the problem a number of goals were identified. After performing a review of the existing software, the goals were refined. The initial user evaluations also fed into the refinement of the goals of the project. Some of these goals were completed in the first stage of development, the other goals were carried forward into the second phase of development.

The goals that were completed in the first stage of development are:

- Improve existing capabilities – The purpose of this goal was to add to Bio-PEPA's visualisation capabilities and bring these more into line with the other modelling software. This goal was completed in the first phase of development. The user was given extra control over the appearance of their graphs. Intensity plotting was also implemented.
- Visualise the model more intuitively – The purpose of this goal was to help the biologists to increase their understanding of what the model and the graph is showing them. This goal was completed in the first phase of development with the implementation of the hierarchical drawing of the model components.

The following goals were to be completed during the second phase of development:

- Provide visualisations that take into account the domain – The purpose of this goal was also to help the biologists further understand what the model and the graph are showing them. This has been implemented in the second phase of the project. This goal was merged with animation of the data.

- Animation of the data – The purpose of this goal was to make it easier for the biologists to interpret spatial data by showing the species in their results moving through the cell. This has been implemented: the user can now visualise species moving through a cell.

- Investigating which visualisation combinations work best – The purpose of this goal was to try and work out whether there were any combinations of visualisations that the program could offer that the biologists found particularly useful. This was originally planned to be performed in the second stage of the project, but due to the project focusing on non visualisation features the goal has been removed.

- Add the ability to annotate the visualisations – The purpose of this goal is to allow the biologists to add extra supporting information directly onto the visualisation. This goal has been implemented: the user can add annotations to the visualisations.

- Allow the ability to save and load the program state – The purpose of this goal was to allow the biologists to complete their work in multiple sessions. This goal has been implemented: users can save and load the program state into files, these can then be emailed to other users who can modify them.

- Provide a full session history to the user – The purpose of this goal is to allow the biologists to recover from any mistakes they might make. This has been implemented: the user has a full undo and redo history within the session allowing them to easily correct mistakes.

- Data Mining – The purpose of this goal was to have the program provide some level of automatic annotation and setting optimisation to reduce the workload for the biologists. Due to other features being added that reduce the need for this goal, the goal has now been dropped. The research for this goal was not lost however as a similar goal was added. Data mining to identify features in not a goal that should be abandoned either, instead it would be implemented if the development continued.

- Making meta data accessible – The purpose of this goal was to The plan for this has not changed since the first phase. This goal has not been completed in this phase of development. It was pushed back to allow development on real time collaboration and plots as queries features. It has not been abandoned, instead it would be completed if development was to continue.

2.2 New Goals

During the first half of the second phase of development new goals were identified.

2.2.1 Data Manipulation and Export

This goal was added after a meeting with the maintainer of BioDARE [15]. This web based tool is aimed at results from laboratory experiments, specifically experiments relating to circadian rhythm.

BioDARE is an online biological data repository. It allows its users to upload their experiment data. This is then available to all other users. BioDARE will generate static graphs of the uploaded data. These plots have little ability to be customized. Instead users can download the data and customize it in a separate application.

The maintainer explained that he had found that researchers often visualise their data after it has been normalised. This removes any issues where different species have different scales and makes it easier for the users to interpret the data.

BioDARE can also export the raw data allowing the researchers to use it in other applications if they desire.

This seemed like a very useful feature for users and it was added as a project goal.

2.2.2 Time Series Data as a Query

There has been a lot of research in recent years [16, 17, 18, 19, 20] into using time series data as a query against a database of other time series plots through some similarity measure.

This would be a very useful feature to add to the tool. By allowing researchers to use a plot as a query they would be able to discover other plots exhibiting similar behaviour. This could be the same species reacting the same to different species, or a species that is exhibiting similar behaviour. This could help the biologists discover alternative candidate species for research.

This feature would be most effective if there was an online repository of plots for the biologist to search. This would also allow them to discover potential collaborators. Online repositories of data do exist: BioDARE is one of them. These repositories are unlikely, at the moment, to store the data in a suitable way to allow for efficient search.

Given the potential usefulness of this feature, and the fact that none of the other modelling software had this capability it has been added as a goal.

2.2.3 Real Time Collaboration

An important area where all the reviewed software were lacking is collaboration, real time or not. The current work flow using the existing software is: perform your analysis, save it, email it to a colleague with additional files and notes, have them open it, they attempt to work out the visualisation is showing. It is a disjointed conversation.

It is not just the modelling software that did not offer collaboration. None of the visualisation software reviewed offered real time collaboration either.

Incorporating real time collaboration was added as a goal to offer researchers a better work flow.

Chapter 3

Phase 2 Development

This chapter details the development work that has been performed in the second phrase of the project. This includes conceptual problems faced and their solutions.

3.1 Design Decisions

During this phase of development a number of design decisions had to be made. The decisions broadly fell into two categories: UI decisions and architectural decisions. Ideally there should be a method behind such decisions. The decisions and the reasoning behind them are below.

3.1.1 UI Design

During the planning phase of the project (MPP) wireframe designs of the UI components were developed. The purpose of these wireframes was to allow for the UI to be evaluated, at least once, before development started. It is cheaper to fix problems early. Fixing the problem before any development is particularly cheap.

The previous wireframe evaluation that was performed was useful, but not for evaluating the UI. The main insights from the wireframe evaluation were to do with potential functionality. Very little was learned about the user's thoughts on potential UIs. The insights into functionality could be learned without the wireframes being mocked up. This is what happened in the evaluations during this phase of development.

During the second phase of development (MPP2) the UI was refined and new elements were added to it. It would have been desirable to again first create wireframes and evaluate them with the users before starting development. This approach was not feasible in this stage of development. It would have been necessary to hold more frequent user evaluations, which would have placed a greater strain on the user's time. The alternative would have been to keep the evaluations at the planned frequency and have the development bottlenecked by evaluating the wireframes. It was decided to not use

wireframe prototypes. Instead the UI was developed and then users evaluated it. The UI was then changed as required. The improvement of the architecture outlined in Section 3.1.2 made changes to the UI easier. After the improvement the UI contained much less information about the state of the tool. The UI is now effectively just calling the API. Having the UI separated means that a change to the UI no longer requires a refactoring of program logic.

3.1.2 Architectural Design

After the first stage of development there was little architectural design. The three main elements of the tool were the UI, the graph visualisation, and the model visualisation. Each component of the UI contained different parts of the visualisation data. Some parts of this data were duplicated across different parts of the program. As you can see in Figure 3.1 each component was passing information to the other components. Some of the items being passed around had references to to other components. At the start of this phase of development (MPP2) the features that were to be implemented were too complex for this 'lack of architecture' architecture.

The program was refactored to focus on an architectural model where all the data was centralised in a single location. This can be seen in Figure 3.2. If the original architecture was continued, with the new components included, the interaction would look like Figure 3.3. Trying to debug and add additional features would have been non trivial. The visualisation and UI components select what they need to display from this central location. The simplified architecture in Figure 3.2 greatly reduced the amount of information that was having to be passed between all the components. A refresh of the UI components is then all that is needed to display the most up to date information. This is similar to the Model View Controller (MVC) architecture. A simple diagram for MVC can be seen in Figure 3.4. In this case the controller and the view are not entirely separated. The model in this project is the session state that contains the information to be displayed. The distinction between the view and the controller is currently blurred with some aspect of control being handled by the view. Separating these would be a priority for future work.

A singleton was used to store the session state. A singleton only allows one instance of itself. This prevented the accidental creation of multiple session state objects. Using a singleton helped guaranteed that all session data would be kept together.

This work to move all the session data to being in a centralised location also enabled two new features: undo/redo, and saving and loading.

Undo and Redo is implemented by pushing and popping copies of this session state onto a stack. The copy of the session data that we want to display is the head of the stack. Undo and redo are important features for usability. This can read about in Section 4.4.2.

A problem was encountered when trying to copy the dictionary onto the stack. When pushing the dictionary onto the stack it would pushing a reference to the dictionary,

not the dictionary itself. This meant that any changes to the dictionary after it had been pushed onto the stack were also there in the stack. Python dictionaries have a copy method. Copy only does a shallow copy – any objects in the original dictionary will have their reference placed in the new dictionary. This was fine for some parts of the session dictionary, but for others it was not. Lines and annotations, which are custom objects, presented problems. This was solved by using Python's deep copy library. With deep copy a new copy is made of objects as well. Some elements of `wxPython` and `matplotlib` could not be deep copied, but this was fixed when the project architecture was changed to have the data in a single location.

Saving and Loading was an important feature to add. In some scenarios a user may not be able to complete all their analysis in one sitting and may want to come back to their work in the future. Without the ability to save and load the user would have to repeatedly add annotations, change preferences, and attach files.

Python has a module called `pickle` to serialise and deserialise data. When saving, the dictionary containing the centralised session data is pickled and written to a file and when loading the reverse happens. Because the program is now focused on the data model, once a previous session has been loaded, a UI refresh is triggered and the visualisation reflects the loaded data.

Saving the data also enables limited collaboration. The user can customize the appearance and add annotations. They can then save the state to a file and email that file to a colleague. The colleague can then load the file and see the user's work. The colleague can then correct any issues and add their own work. The colleague can then save this and email it back to the user. This is useful and is better than no collaboration, but it is entirely asynchronous.

Figure 3.5 shows a more detailed architectural diagram of the system.

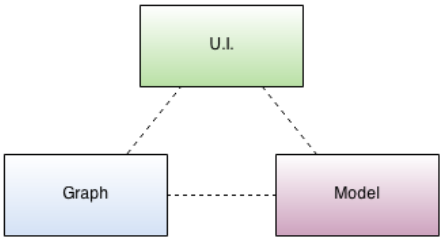


Figure 3.1: Architectural layout at the start of MPP2

3.2 Animation

Animation was a key goal of the project. The core aim of the project was to help biologists who are not comfortable with traditional time-series graphs. The goal was to provide them with visualisations closer to what they see in their domain. This has been accomplished by displaying spatial data in the shape of a cross-section of a cell.

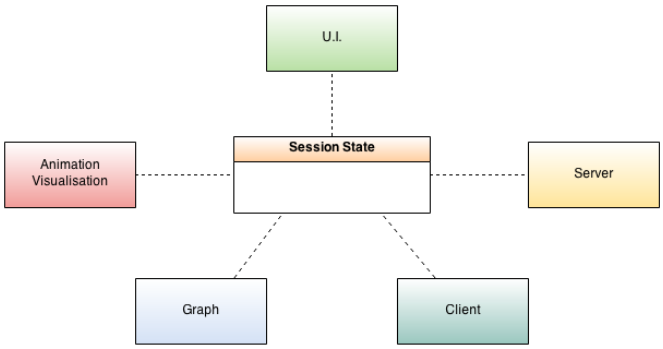


Figure 3.2: Architectural layout at the end of MPP2

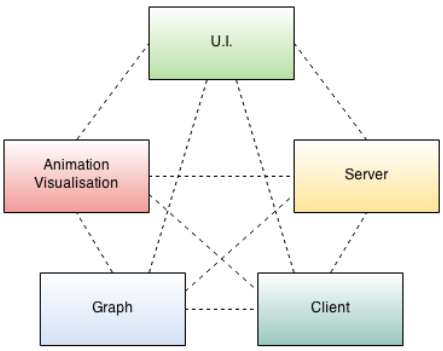


Figure 3.3: Architectural layout at the end of MPP2 if initial architecture was continued.

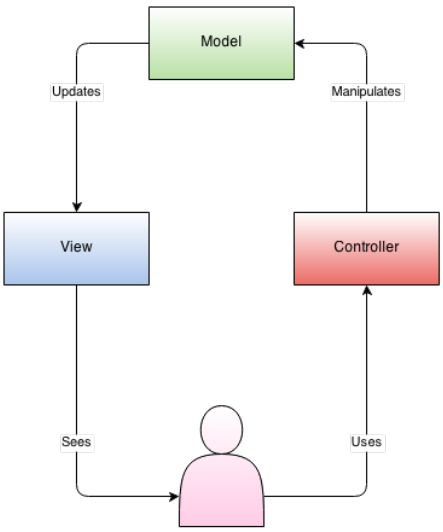


Figure 3.4: Basic MVC architecture

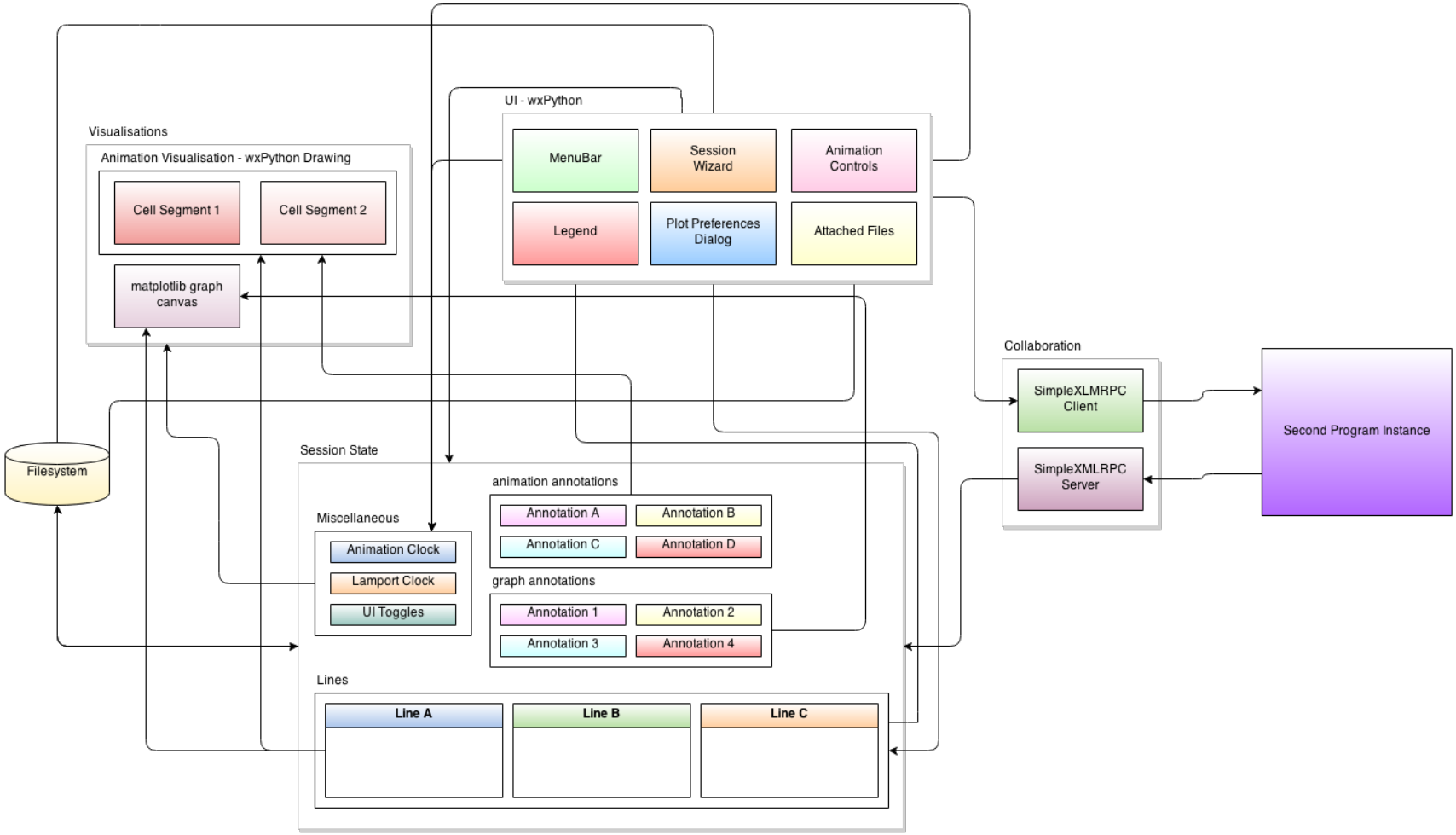
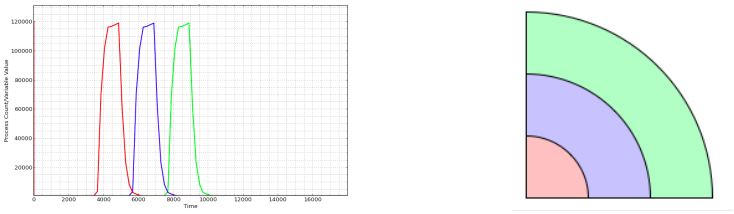


Figure 3.5: Detailed breakdown of program components and how they interact

The animation is ideally used to display species moving through a cell. This collapses what would be multiple different lines in a graph, that give no indication of their real position in the cell, into a single image. This can be seen in Figure 3.6.



(a) Time series representation (b) Spatial representation

Figure 3.6: One species at three locations in the cell represented traditionally on a time series graph and also spatially in a cell

Animation is just a sequence of static images which, when viewed frame by frame at a sufficient rate, will appear to be a dynamic image. There are two basic ways that animation can be implemented: rendering the entire animation before it is required, or generating a static image as a function of time and every frame calculating what the image should be and displaying that to the user. Both approaches have their advantages and disadvantages. The most appropriate depends on the situation. These options and how they would affect this project are reviewed below.

Rendering the animation before it is needed is not the correct way to do animation in this circumstance. One of the purposes of this tool is to allow for interactivity and customisation. This would potentially lead to the animation having to be rendered multiple times. The animation that was planned to be included in this tool is also not complex enough to warrant pre-rendering. Pre-rendering would be more suitable for more realistic animations. One would be doing this after the insight has been learned from the experimental data. The purpose of the animation here is to help the user to find this insight. For these reasons treating animation as generating static images as a function of time was chosen.

We had to decide how the animation would be displayed. `wxPython` and `matplotlib` are both used for visual display in this project. It made sense to implement animation using one of them. The capabilities of implementing animation was investigated for choices before the decision was made. The findings are below.

`wxPython` does not have built-in animation support but it does, however, have built-in drawing support through the paint device context. This allows for arbitrary drawing on UI panels. `wxPython` also provides some drawing primitives. Various primitives can be handled such as squares, circles, triangles and text. There is also support for drawing arcs. These primitives can be combined to form regions allowing more complex shapes to be drawn. During the planning phase (MPP) it was decided to draw cell cross-sections. These cell cross-sections were going to be arcs. `wxPython` providing support for arcs was perfect for this. This approach would be most suitable for generating images as a function of time.

`matplotlib` does have support for animation with the `matplotlib`.animation library.

This implements animation as a function of time. To display animations `matplotlib` requires: a function that generates the data to be plotted (which will change depending on the time value) and the interval of time between frames. `matplotlib` also has support for shapes. Each cell cross-section could then be treated as a separate `matplotlib` animation. `matplotlib` would also require a canvas to draw on. The appearance of this canvas would have had to be customised to make the animation look less like a graph, for example removing ticker lines from the axis. This would be a lot of scaffold code to implement animation.

`wxPython` and `matplotlib's` animation capabilities are quite similar, both have primitives for drawing the necessary shapes and both are implemented as drawing static images as a function of time. Given that the capabilities of each were similar, `wxPython` was chosen for animation to avoid the scaffolding code that `matplotlib` would have required. This enabled animation to be developed faster.

The animation visualisation will usually be one or more cell cross-sections. Each cross-section is split into compartments. These compartments are parsed from the model file. In the model file compartments are stored hierarchically. The structure of the cell in the model can be inferred from this hierarchy. Each compartment in the model relates to one segment in the cell cross-section. `wxPython's` built-in arc drawing was used to draw the cross-sections. The cross-section has to be drawn from the edge of the structure into the centre. Each arc is drawn on top of what is already there. If the larger outer arcs were drawn last then they would cover the inner arcs. The division of the cell cross-section is such that every visible segment has the same width. An example cell cross-section can be seen in Figure 3.7

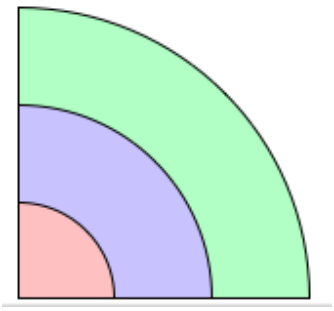


Figure 3.7: A cell cross-section formed of three segments.

Over the course of the animation the colour of the segment changes to reflect the colour in the intensity plot version of the line. This allows the researcher to compare the two visualisations and will hopefully help build their confidence understanding the graph plot.

Animation takes advantage of the data-centric model. When the play button is pressed a thread is created whose job it is to increment the animation clock and refresh all the UI elements. When refreshed the UI elements pick up on the change of clock and update what they display appropriately.

The first step towards animation was calculating at what points the line changes colour. This is done during the interpolation phase that was implemented in the first stage of

development. To do the interpolation the original plot is split into multiple separate plots with the results padded with `null` values to allow `matplotlib` to plot them as one. This is detailed further in the report for MInf Project Phase 1. The number of these `null` values is counted and that gives the time at which the colour should change. This is stored as an array of time, colour tuples. Each line also has an internal counter to say how many elements into this array the line is. When the animation clock is updated and the refresh triggered, each cell segment finds the appropriate colour change point in the line by going through the array and comparing each time to the animation clock until the time is found and updates the segment colour. This position is then used as the new starting point on the next refresh. This removes the need to look through past entries unnecessarily.

The user can control the animation clock through the time slider. Moving the time slider updates the animation clock to be the value of the time slider and triggers a refresh of all the cell segments. This ensures that the user is seeing up to date information. As described above, each line has a list of times at which it changes colour. These are sorted according to time. During 'normal' animation the previous position in the array is tracked so that the whole array is not searched every time unnecessarily. When the time slider is used this previous position might become a future position. To solve this the previous position is set back to the start of the array. On the next refresh the array will be searched from the start for the appropriate time and colour.

A nice UX feature that has been added is a line on the main graph that indicates the current time in the animation. This can be seen in Figure 3.8. When the animation is running the line moves along the graph. This again helps the user build a correspondence between the two visualisation types and may help with their understanding of what is being displayed on the graph.

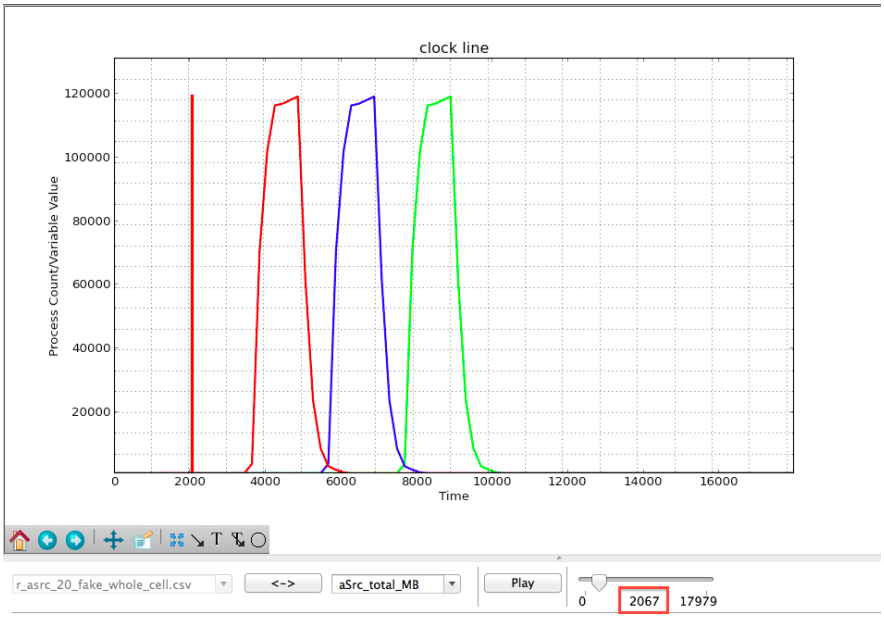
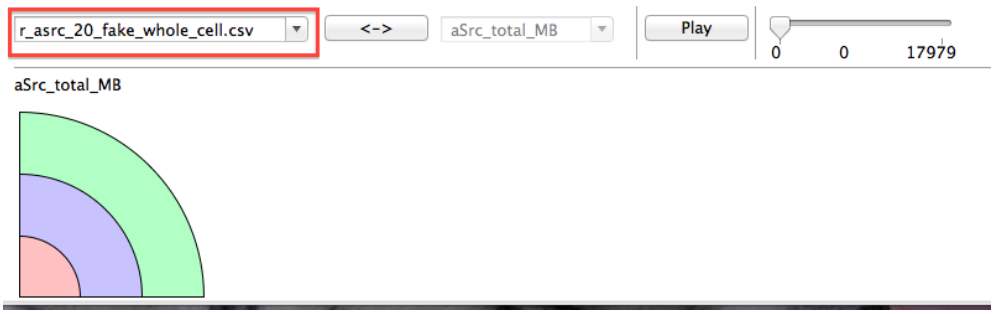


Figure 3.8: Vertical line that is drawn on the graph to indicate current animation clock time.

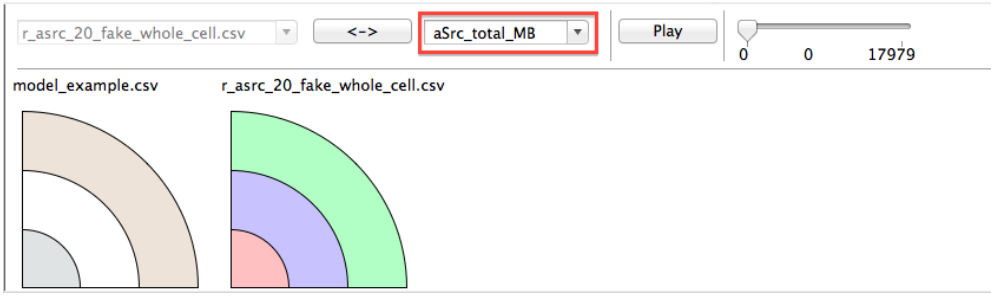
The user is also given control over what is displayed. The cell cross-sections can be drawn in a file-focussed manner or a species-focussed manner. In the file-focussed manner a cell cross-section is drawn for every species in that file. The user can switch between all results files that have been added to the session. In the species-focussed manner a cell cross section is drawn for every file that contains that species. The user can select between all species found across all the results files. These are illustrated in Figure 3.9

Figure 3.9 shows example cell cross-sections. Figures 3.9a and 3.9b show the cell cross-sections being drawn in the file-focussed manner. Both files contained results relating to the the species ``aSrc_total_MB". In Figure 3.9a this species was present at all locations in the cell. In Figure 3.9b the species is missing from the middle location in the cell. The middle segment has been left white to indicate this. Figure 3.9c shows the cell segments drawn in a species-focussed manner. One cell cross-section has been drawn for each file which contains the species ``aSrc_total_MB''. We can see that in one of the files the species was not present at all locations.



(a) Cell segments drawn by species from file r_arsrc_20_fake_whole_cell.csv

(b) Cell segments drawn by species from file model_example.csv



(c) Cell segments drawn by file from species aSrc_total_MB

Figure 3.9: Example cell segment animations

It had been hoped to allow the user to save the animation. This could have been implemented by either allowing the user to save individual frames, or allowing the user to save an animated `gif` of the whole animation. Saving to a `gif` was chosen as it was felt to be more useful, to a user, to be able to export the entire animation rather than a single frame of it. Allowing the user to save a single frame would be added in at a future date. To generate the `gif` we need to save all frames. As these frames are generated as needed, not pre-rendered it was required to run through the whole animation to save each frame. This meant that saving all the frames took as long as the animation's run time. At this stage in the save animation process, hundreds of static images have been saved. A shell script is then required to convert the saved frames into a `gif`. Given that this was a very slow and user unfriendly approach the decision was made to remove the ability to save animation from the project. It would, hopefully, be implemented if future work was to be performed.

3.3 Annotation

Annotation was an important feature to add. It is one of Grinstein's features that data visualisation software should have [21]. This is because the purpose of a visualisation is to aid analysis. If you have a print out of a visualisation you are able draw on it and highlight areas of interest. Users need to be able to do this on digital versions of their visualisations. As well as helping users analyse their data, being able to annotate also means that images for presentations can be prepared without having to save the visualisation and open it in an external program. If you did this and then wanted to change the visualisation you would have to re-annotate it. This is frustrating for a user and wastes their time. Being able to do the annotation from within the visualisation solves this problem as the raw data and the annotation data are together. Another benefit of being able to annotate is having another way of attaching additional information to the visualisation when sending it to a colleague. Having this information on the visualisation saves the colleague from flicking back and forth from an email or other supporting documentation.

Implementing annotation of the visualisations on offer in this tool had to be done in two parts. This is because there are two parts to the visualisation: the graph and the cell level animation. One is a static visualisation and one is a dynamic visualisation. Each of these required a different approach. These are detailed below.

3.3.1 Annotation of the Graph

Annotations of a static image are extra elements that are added by the user. We needed to decide what extra elements we are going to allow the user to add. We also needed to decide how to render the annotations.

For this project there are two options for drawing the annotations on the graph: with `matplotlib`, or with `wxPython`. `matplotlib` has support for annotations. In `matplotlib` annotations have two forms. The first form uses the `annotate()` and `text()` functions

which, respectively, add an arrow with optional text to the graph and add text without an arrow to the graph. matplotlib's second form uses its artists and patches library. With these libraries arbitrary shapes and different forms of arrow can be added. The other option was drawing the annotations with wxPython. As in Section 3.2 we can use device contexts to draw primitives on UI elements with wxPython. The primitives offered by the device context did not include arrows which are a key annotation type. In Section 3.2 wxPython was chosen over matplotlib partially because using matplotlib would have involved more scaffolding code. This was not a factor for annotation as the matplotlib primitives use the graph that has already been created for the displaying the results data. matplotlib was chosen for implementing annotations as it had a greater variety of primitives and required less code for the annotations.

matplotlib's support for elements on the graph is quite thorough. Text and arrows are obvious choices as they allow the user to indicate and explain areas of interest. matplotlib then allows for more arbitrary drawing. So as to not overload the user with annotation choices it was decided to limit the number of annotation elements that they could use. In addition to the text and arrows the user can also add circles as an annotation.

Users of the new tool are provided with four annotation types: arrow, text, arrow with text and circles. Buttons for each of these annotation types have been placed on the matplotlib toolbar. This can be seen in Figure 3.10. These four annotation types come from different libraries within matplotlib. An annotation class was defined to attempt to abstract their differences. This would also make it easier were more annotation elements to be added in the future.

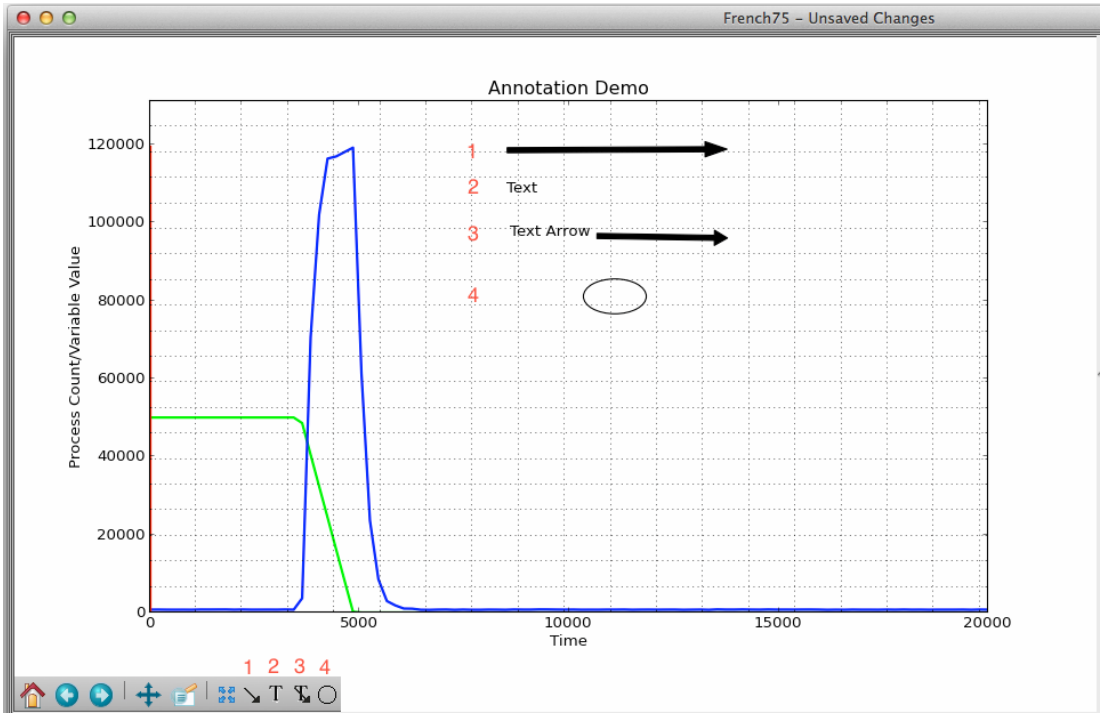


Figure 3.10: Example of each of the four graph annotation types and the toolbar buttons that correspond to them.

There were problems in making the annotation creation process user friendly. For all of the annotations the user needs to first press the appropriate button on the toolbar. The next actions depend on the annotation type. Text and circle annotations were intuitive as all they require is one click. The user clicks where they want the annotation to be placed and it will be drawn on the graph there. However the arrow annotation types required two clicks. The first click marks the start point (the tail of the arrow) and the second click is the finish point (the head of the arrow). This was not obvious. User evaluations revealed that the users did not know that it required two clicks and did not know whether the arrows would be drawn head to tail or tail to head. The technique for placing arrows was subsequently changed so that the first click still fixed the position of the tail of the arrow. However the behaviour after the first click changed, so now a temporary annotation is continuously redrawn that has the head of the arrow wherever the cursor is. This allows the user to see the arrow they are drawing. To indicate to the user that they need to click on the graph the cursor is changed after pressing one of the annotation buttons on the toolbar. The use of different cursors is a common technique to help guide users to perform actions.

It was important that annotations could be edited or deleted. Recoverability from error is an import Human Computer Interaction (HCI) guideline [22][23][24]. The annotations can not just be clicked as they are not a UI widget like a button. The solution to this was to have an array of annotations. When a user right clicks on the graph it searches through all annotations and selects the annotation that was closest to the click (if that distance was below a certain threshold). The selected annotation is then highlighted red, and a context menu appears to give feedback to the user that they have successfully selected an annotation. This can be seen in Figure 3.11. The context menu then gives the user the option to edit or delete an annotation. Editing an annotation only allows for editing text. For changing position, the annotation has to be deleted and redrawn.

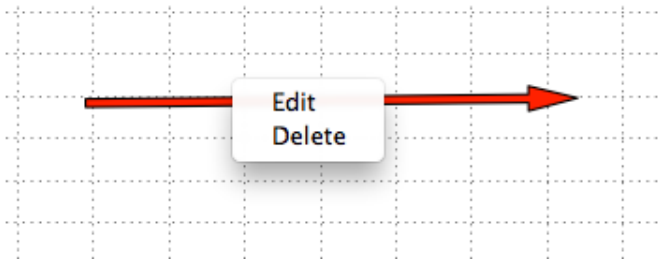
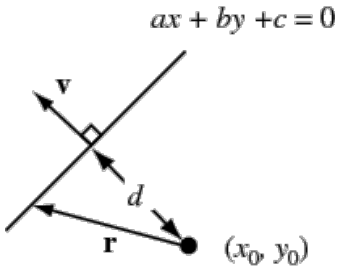


Figure 3.11: Context menu on selection of graph annotation.

To calculate the distance from the location of the mouse click to the annotation two problems had to be solved. The first was how to calculate the distance from a point to a line (This is only for the arrow annotations. The text and circle annotations use standard point to point Euclidean distance). An equation for this can be seen in Figure 3.13. Equation 3.1a is the equation for a line in two dimensions. This represents the annotation. The equation can be calculated from the start and end points of the annotation. Equation 3.1b is the position of the mouse click. Equation 3.1c shows the equation for calculating the distance from the point to the line. The second problem that had to be overcome was the difference in scale. In effect we have two coordinate

systems. The results coordinate system and the visual coordinate system. The difference is illustrated in Figure 3.14. The annotations in Figure 3.14a are further away from each other in the results coordinate system than the annotations in Figure 3.14b. However they are closer in the visual coordinate system. When selecting an annotation the user is using the visual coordinate system but the distance is calculated using the result coordinate system. This could lead to the wrong annotation being selected. The solution to this was to transform one coordinate system into the other. When calculating the distance from the point to the line, the values are scaled by the size of the graph.



(a) Diagram of the point to line distance problem

$$y = -\frac{a}{b}x - \frac{c}{b} \tag{3.1a}$$

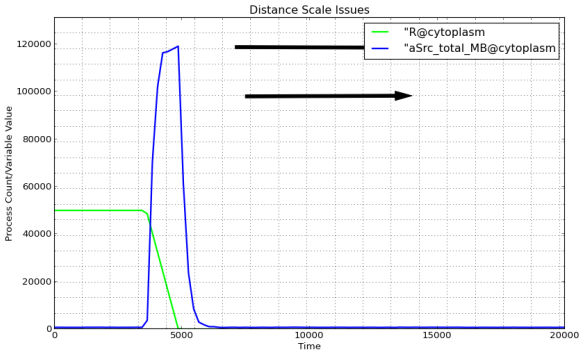
$$(x_0, y_0) \tag{3.1b}$$

$$d = \frac{|\, ax_0 + by_0 + c \,|}{\sqrt{a^2 + b^2}} \tag{3.1c}$$

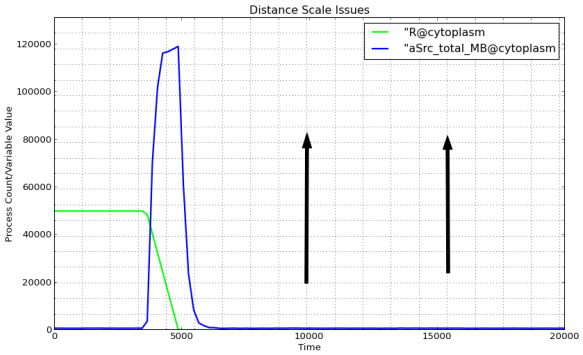
(b) Equations for calculating the distance from a point to a line.

Figure 3.13: Equations and diagram for dalculating the distance from a point to a line (taken from WolframMathWorld [25])

A similar issue to the difference in scale when calculating the distance from the mouse to annotation was encountered when switching the graph to display the normalised data (described in Section 3.6). When normalised the graph in the y-axis only goes between 0 and 1. The coordinates of the annotation are likely to be very much outside this range and the annotation would not appear on the graph. The solution, if the graph was normalised, was to normalise the positions of the annotations in the y-axis when drawing them. A side effect of this technique is that after the lines are rescaled. The annotations could be left pointing to nothing. This is unavoidable as the annotation has no concept of what it is pointing at. It may be pointing to the intersection of many lines; in this case it would be impossible to keep the annotation pointing at what it was originally pointing at after rescaling the axis. During user evaluations, when the annotation was not visible during data normalisation, the user was very confused as to why their annotation had disappeared. It was therefore decided that having the annotation be visible but most likely incorrect is more user friendly than just not displaying the annotations. Not displaying the annotations looks like a serious error.



(a) Annotations with distance of approximately 20000



(b) Annotations with distance of approximately 5000

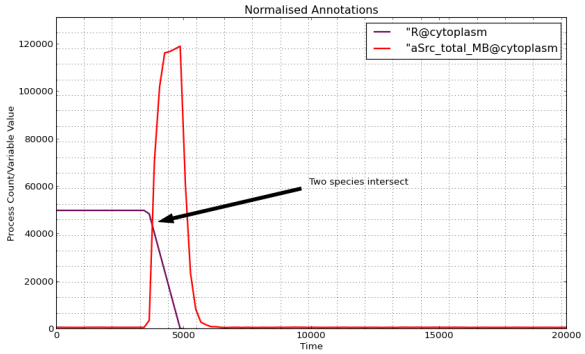
Figure 3.14: Two sets of annotations illustrating the problems when calculating the distance to an annotation

3.3.2 Annotation of the Animation

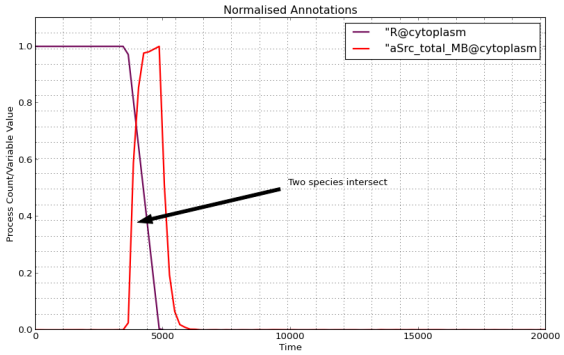
After completing annotation of the graph it was important to expand this to the animation panel, as this is the other visualisation option available to a user. Annotating animations posed more of a challenge than annotation of the graph and there were a number of issues to overcome.

1. How to implement the annotations? For the graph `matplotlib` has built-in annotation support. `wxPython` does have drawing support but not built-in annotation support. Annotating on the animation will need manual handling of the drawing on top of the animation visualisation. Manual drawing means that the automatic layout functionality that `wxPython` provides cannot be used.

2. When to display the annotations? When an annotation is drawn on the graph it is displayed at all times. The appearance of the graph does not change over time. However the appearance of the animation visualisation does change over time. The problem faced when annotating is whether to have annotations available at only specific times in the animation, or to have them there the whole time; and if they are going to appear and disappear, how can it be done without being distracting?



(a) Annotation pointing at the correct intersection of species



(b) Annotation pointing at nothing after data normalisation

Figure 3.15: Graphs illustrating what happens to annotations whilst data is normalised

3. How to give the user control over the annotations? When a user wants to edit or delete an annotation on the graph it is always there. However on the animation panel if the annotation is temporal, then it is not always visible for the user to edit or delete and it would be frustrating for a user to constantly have to search through the animation to look for annotations to change them.

A platform where users are able to annotate an 'animation' is YouTube [26]. YouTube's annotation interface can be seen in Figure 3.16. YouTube's approach is to allow the user to set what period of time an annotation is visible for. There is also a management panel where the user can see all of the annotations and edit or delete them. This approach solved issues two and three and was adapted for this tool.

On the right hand side of the tool there is a panel which lists all annotations that have currently been added to the tool, this can be seen in Figure 3.17. This provides the user with the persistent view of what has been added. When the user creates an annotation they are shown a dialogue that allows them to enter the annotation text and to choose a duration. This can be seen in Figure 3.18. The duration indicates to the user that the annotation will only be visible at certain times. The start and end times are given in model time. The duration displayed to the user is given as the real time that the annotation will be visible for.

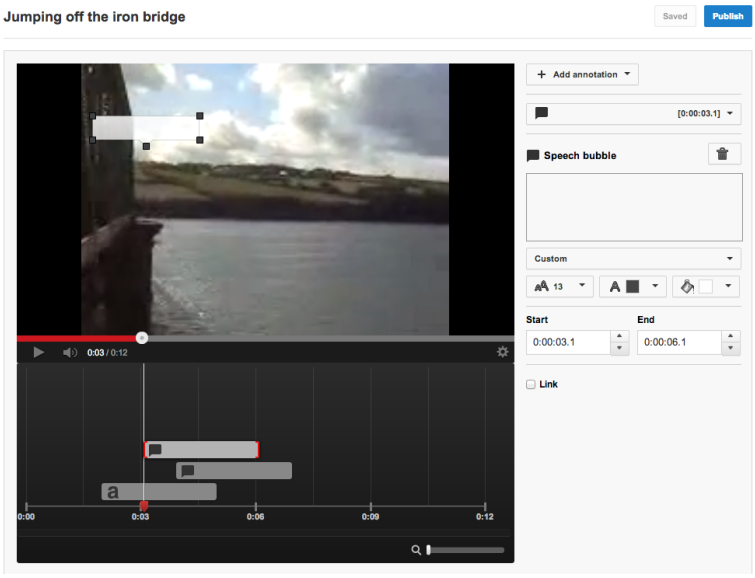


Figure 3.16: YouTube video annotation interface

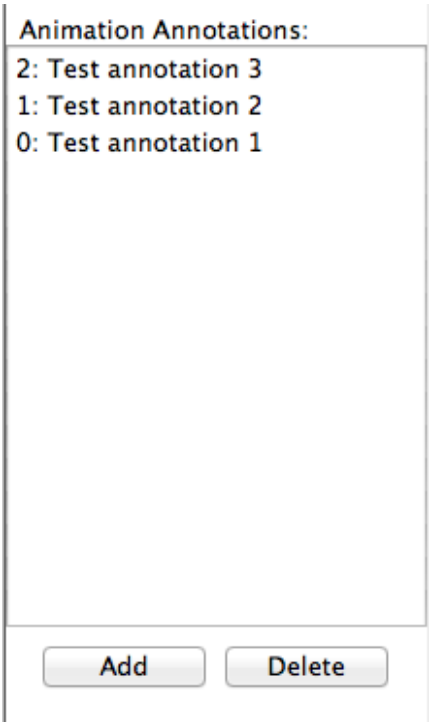


Figure 3.17: Animation annotation list

The first problem – how to display the annotations – has also been solved. Drawing in `wxPython` takes place on panels. Each cell cross-section is placed on its own panel. This drastically limits the available space. The panels are not wide enough to have the annotation text drawn on. The solution was to assign each annotation a number and that number is what is used to annotate the cell. The number is then displayed in the annotation list box allowing the user to read the appropriate annotation. This can be seen in Figure 3.19

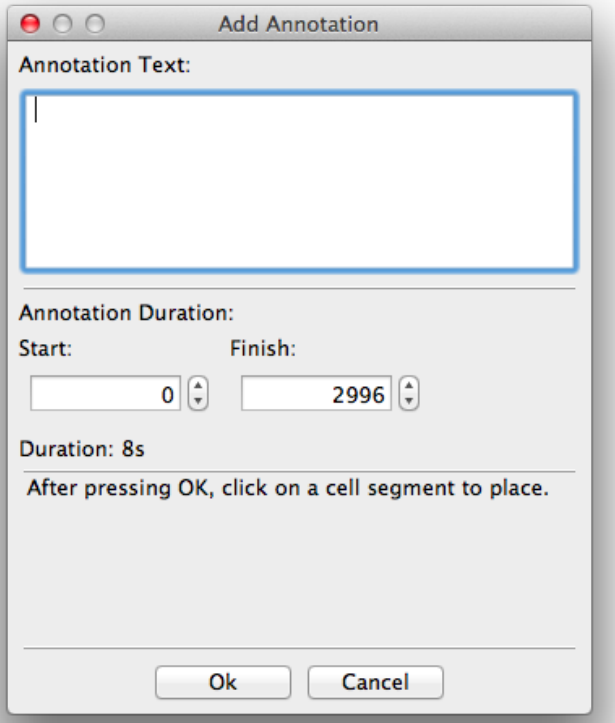
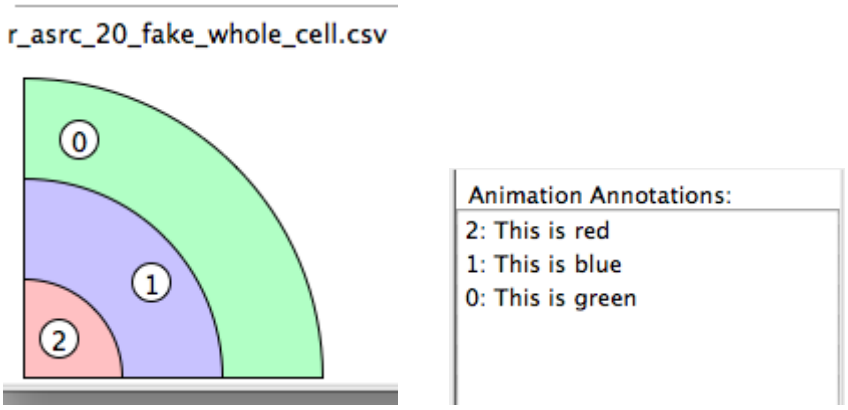


Figure 3.18: Animation annotation dialogue



(a) Annotated cell cross-section (b) Corresponding annotation List

Figure 3.19: Animation annotations

3.4 Search

Being able to use a time series graph as a query against a database of time series data was one of the new goals added to the project. It was felt it would be of great use to the biologists as it would allow for discoverability within the tool. It had been identified

as a potential goal whilst researching data mining in time series data. It was added as a definite project goal after the user group backed it as a feature they would very much like to have. It is also a feature in the tool that incorporates cross-domain knowledge in an area of active research.

This was one of the more challenging features implemented in this project. This was due to it requiring the implementation of extra functionality on top of work done by researchers in a relatively new field.

Some problems needed to be overcome before using plots as a query could become useful:

- How to cope with different scales?
- How to cope with events happening at different times?
- How to represent the plot to allow for efficient search?
- How to determine similarity between two graphs?

Early techniques used simple similarity measures such as Euclidean distance, but these gave poor results [27]. The reason for the poor results is that Euclidean distance does not take into account scale or time. Figure 3.20 illustrates a case where Euclidean distance would give a poor similarity measure. The two lines are identical, but the green line has been shifted in time. Euclidean distance will not recognise that they are similar. The same problem would happen if one line had been shifted in the y-axis above the other.

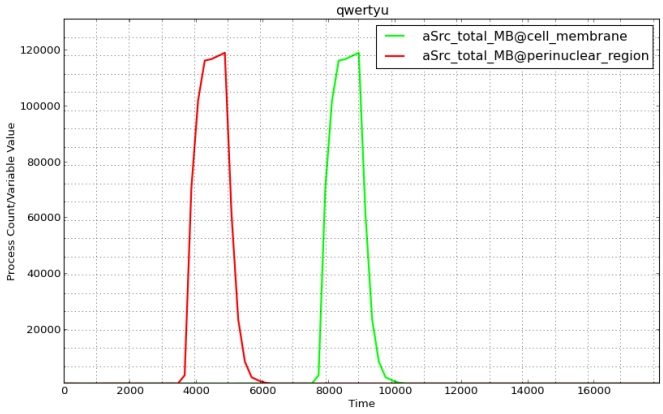
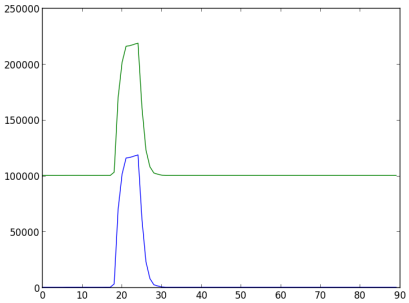


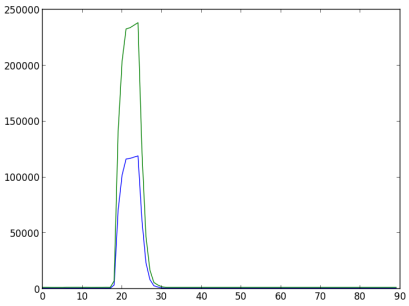
Figure 3.20: Identical plots shifted in time

It is important to define what it means for two lines to be similar. It is obvious that in a case such as Figure 3.20 that the two lines are similar. Both lines are the same shape and they are exhibiting the same behaviour. When determining similarity of plots we should take a time invariant approach. Perhaps less obvious is the case where we have offset in the y-axis as shown in Figure 3.21. Figure 3.21a shows two identical lines where one line had an additional 100000 population level. Figure 3.21b shows two lines where one line is double the other line. Figure 3.21a is effectively the same case as Figure 3.20. Figure 3.21b is different, but it is clear, looking at the graph, that they

are exhibiting the same behaviour. When determining the similarity of plots we should take a scale invariant approach [28].



(a) Identical plots shifted in population level



(b) Identical plots scaled in population level

Figure 3.21: Graphs illustrating different ways lines can be offset in the y-axis

Approaches that have been researched to determine similarity between time series data have included shape based approaches and dynamic programming techniques [29, 30, 31]. Not all of these approaches have included dimensionality reduction. The dimensionality reduction transforms the input data into a vector space with fewer dimensions. Similar values in the data space will, in theory, be transformed to the same value in the vector space. The need for dimensionality reduction is explained later in this section. After researching alternatives an approach was adapted from the work of Lin and Li [32].

Lin and Li's approach uses normalised sub sequence data to reduce the dimensionality of the dataset. The reduced data can then be used to efficiently find the most similar plots in the database. The input data is converted into sub lists to ensure that all features will have been captured. These lists are then normalised and have their dimensionality reduced. This is where Lin and Li's work left the topic. Work from the field of information retrieval was then applied to provide the similarity ranking. This approach is explained in further detail below.

The first step was to take the input data and return all the sub lists of this data. To do this a sliding window was used. The size of the sliding window will determine the feature size in the reduced vector. Lin and Li use a size of eight for the sliding window. The same value was chosen in this implementation. If we have input data $[1,2,3,4,5]$

and windows size $n = 3$ (for illustrative purposes only) then we get the sub lists as $[[1,2,3], [2,3,4], [3,4,5]]$.

The second step is to normalise each sub list. Each sub list is normalised to be zero mean and unit variance. This places each sub list onto the same scale. This removes the effect of the offsets seen in Figure 3.21. To normalise to zero mean and unit variance we need the mean (μ) and the standard deviation (σ). We then have:

$$\left[\frac{x-\mu}{\sigma},\ \forall x \in sublist\right]$$

The third step is to reduce the dimensionality of the data. Lin and Li's approach also involves converting the data from a continuous representation into a discrete representation. By normalizing the sub lists to be zero mean and unit variance we allow a Gaussian distribution to be fitted to the data. This can be seen in Figure 3.22. This approach does assume the Gaussian distribution is a suitable distribution. Lin and Li found it to be a suitable distribution but do note that other distributions could be investigated; this was outside the scope of this project. For each data point we use the Gaussian function and map it to a letter. This can be seen in Figure 3.22 which shows a line formed of eight datapoints, our sublist, that have been normalised to be zero mean and unit variance. Each datapoint has been mapped to one of the three sections of the distribution divided by the breakpoint and has been replaced with the letter that represents that section. Breakpoints are used to divide the Gaussian distribution into sections with equal probability. To split the distribution into three sections two breakpoints are needed. The values of these breakpoints can be found in statistical lookup tables. If our breakpoints divide the distribution into three sections then each datapoint can be mapped to one of these three sections. Lin and Li divide their distribution into three sections and represent each section with a character. They found three sections to be suitable, again alternative numbers of breakpoints could be used but this is outside the scope of the project. This reduces the dimensionality of the dataset as multiple real values are mapped to a single letter.

After this stage each sublist has been transformed into an eight character long string. The plot is represented as a list of strings. These strings represent the features of the plot.

Lin and Li now call for the removal of local duplicates within the list of features. If we have runs of a certain feature then that should be reduced to a single instance of the feature. For example, using a feature size of three, if we have the reduced representation "*AAA*", "*AAA*", "*BBA*", "*BBB*", "*BBB*", "*AAA*", "*AAA*" it should be reduced to "*AAA*", "*BBA*", "*BBB*", "*AAA*" not "*AAA*", "*BBA*", "*BBB*". We are not removing all duplicates of the feature. This reduces the number of entries in the representation and provides an efficiency benefit. It will also remove the effect seen in Figure 3.21b where Euclidean distance would give a value indicating dissimilar if one line is a scaled version of the other. This is because we have just removed long features with the same behaviour to be unit length features.

The plot is now represented as a list of strings without local duplicates. Our strings

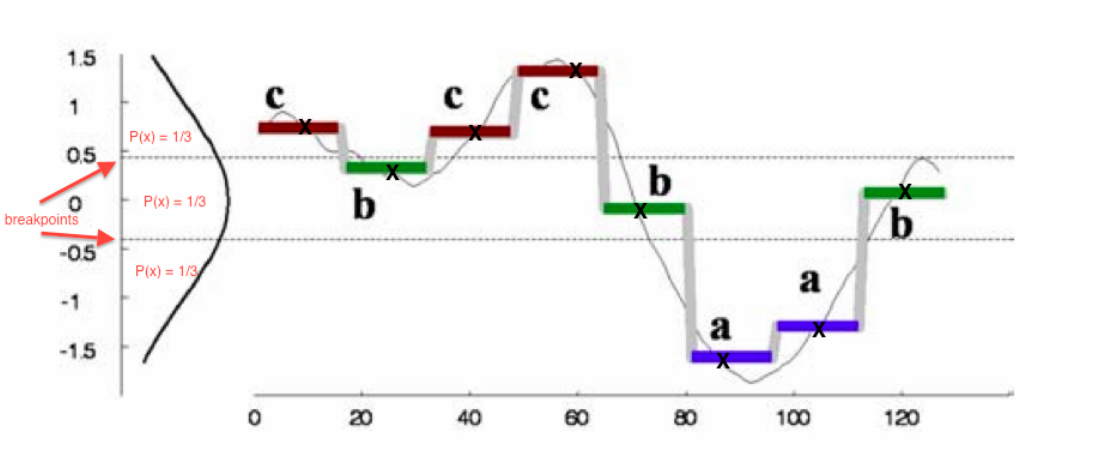


Figure 3.22: Fitting a Gaussian distribution to time series data, taken from [32].

all have a length of eight characters and the strings are composed from a three letter alphabet. This gives us a vocabulary of 3^8 possible features. The index of the plot is a 3^8 element vector where each entry in the vector contains the count of the feature in the line. This is a very wasteful representation as most of the entries in the vector will be zero. We can replace this with a sparse representation where we store only the features that make up the line and their count.

This representation of the plot also solves the problem of features being temporally located in the graph. In the plot index there is no ordering of the features, in effect all the features happened at the same time. This is a common representation in information retrieval and is referred to as bag-of-words. A more suitable name in this domain would be bag-of-features.

Now that we have a representation of the plot that is scale and time invariant and allows for efficient comparisons we can find a method to calculate the similarity. Lin and Li leave similarity as future work and use their representation to perform clustering, classification and anomaly detection. It was therefore necessary to find a technique, potentially from a different domain, to find the similarity.

We could simply just use Euclidean distance again, using the two vectors as the input, but this does not do anything to weight rarer features. If two plots have rare features in common then that is much more significant for similarity than if they share very common features. A similarity measure that takes into account the importance of a word is therefore desirable.

It was decided to implement term frequency-inverse document frequency (tf.idf) weighted cosine as the similarity measure. This takes into account term frequencies to provide a higher weight to rarer vectors. tf.idf is very well researched in the field of information retrieval and text mining. Given the representation is equivalent to a bag-of-words it seems sensible to apply tf.idf to this domain. A description of how tf.idf weighted cosine works can be found in Section 3.4.1.

The tf.idf weighted cosine similarity [33, p. 243] can then be calculated between the

query plot and each individual plot in the database. These similarity scores can then be ranked.

Lin and Li do note that for their approach to work the data must be suitably long. They did not quantify what suitable long is. The implementation in the tool is performed on the interpolated data. The interpolated data is all ~ 1000 elements long. This is believed to be suitably long.

3.4.1 Tf.idf

tf.idf is a technique for calculating how important a word is to a document. tf.idf requires a corpus of document vectors. With the corpus data tf.idf can take as input a query vector and transform that into a vector of weights, where the weights are the tf.idf scores of each word. We want to assign a higher tf.idf weight to those words which are rarer. The rarity of the word applies only within the corpus of documents that is being used. A larger corpus of documents will lead to a more accurate scoring of the rarity of the word.

Equation 3.2a is the equation for finding the similarity between a short query term and a longer document. This equation would be applied between the query term and every document in the corpus. This would then give a similarity ranking. Below is an explanation of the components of the equation and how they affect the weighting of a word.

- $tf_{w,Q}$ – The number of times word w appears in query Q. If a word is repeated in the query it is likely to be important.
- $tf_{w,D}$ – The number of times word w appears in document D. If a word is repeated in the document it is likey to be important.
- k – A squashing factor. The first occurrence of a word is more important than repeats.
- $|D|$ – Length of the document
- $|C|$ – Number of documents in the corpus
- df_w – Number of documents in the corpus that contain word w
- $avg|D|$ – Average length of all documents in the corpus.
- $tf_{w,D} + \frac{k|D|}{avg|D|}$ – Normalising factor, repeated words are less important unless the document is longer than average.
- $\log \frac{|C|}{df_w}$ – Increases the weighting of words that are rare across the corpus.

Equation 3.2b is the equation for calculating the tf.idf weighting of a single word in document vector. This equation would be applied to every word in the document vector and transform the document vector into the document's weight vector.

Equation 3.2c is the equation for tf.idf weighted cosine similarity. This gives a better measure of similarity when the query is a document as is the case here [34][slide 11].

$$s(Q,D) = \sum_w tf_{w,Q} \times \frac{tf_{w,D}}{tf_{w,D} + \frac{k|D|}{avg|D|}} \times \log \frac{|C|}{df_w} \tag{3.2a}$$

$$D_w = \frac{tf_{w,D}}{tf_{w,D} + \frac{k|D|}{avg|D|}} \times \log \frac{|C|}{df_w} \tag{3.2b}$$

$$S(D_1,D_2) = \frac{\sum_w D_{1,w} \times D_{2,w}}{\sqrt{\sum_w D_{1,w}^2} \times \sum_w D_{2,w}^2} \tag{3.2c}$$

tf.idf allows for efficient search through the use of an index and an inverted index. The index is a mapping from document to words and the inverted index is a mapping from words to documents. If we look up a document we find what words are in it. If we look up a word we find what documents contain it. When a new plot is added to the database these two indexes are updated. With these two indexes all the components of tf.idf can be calculated for very little cost. For example $tf_{w,D}$ would involve looking up document D in the index; this will return to us all the words in document D and their counts. We can then quickly find the count of word w in D.

3.5 Real Time Collaboration

More and more programs are allowing their users to collaborate in real time. Some of these pieces of software are discussed in Section 1.2. Only one piece of software, with real time collaboration capabilities, could be found that was biology focused. Implementing real time collaboration in this tool is therefore a great advantage.

There are two broad ways that real time collaboration could have been implemented: in a distributed manner or by having a single centralised instance. A single centralised instance is the approach taken by all the software reviewed. The approach taken in this tool is the distributed approach.

The single centralised approach would have been simpler. With the distributed approach you have to ensure that each instance of the program has the same ordering of events and is displaying the same state. In the centralised approach there is a single repository which has all history of what has happened. The order that the single program instance thinks events happens in is the order that those events happened in.

Ideally the collaborative model used in this project would have been the centralised model. This would however have required a significant refactoring of the project. It would have required a rewrite so that all the processing would take place on the server with the users just interacting with a simple client. Given the time constraints on the project and the fact that it had been developed, up until this point, as a standard desktop application it was felt it would be easier to implement distributed collaboration.

The collaboration that has been implemented in this project is only between two program instances. Future work would include expanding the collaboration to work for a more arbitrary number of users.

The first stage in allowing distributed collaboration was enabling the different program instances to talk to each other. There are a number of techniques to allow for this. Data could have been sent directly through sockets. Another option would be to create a web server and send `HTTP GET` requests and pass parameters. It was decided to use an existing library to handle the communication. The library chosen was `simplexmlrpc`. `simplexmlrpc` handles the implementation of communication via sockets. Messages sent via sockets will often be split into multiple parts. A manual implementation has to be able to handle receiving of arbitrary length messages in multiple parts. It makes sense to use a library where this is provided. `simplexmlrpc` provides a Remote Procedure Call (RPC) server. Methods are then registered in the server to make them publicly accessible. `simplexmlrpc` also provides a method for a client to connect to the server. The client can then call methods that the server has made public.

One problem that was encountered with the `simplexmlrpc` library was that when sending the entire session state during the handshake it was unable to serialise the more complex objects. This was a similar problem as faced when implementing undo/redo in Section 3.1.2. The solution to this problem was to first serialise the data to be sent using `pickle`. The pickled data is then reserialised by `simplexmlrpc` and is sent successfully. Serialising the data twice is not optimal, but it was necessary for the data to be sent.

Figure 3.23 shows the communication required for two collaborators to work on the same visualisation session. There is a 'handshake' stage where User A sends their IP address to User B. This allows User B to connect to User A's session. User B's client then requests User A's session which User A's server sends back. Both users are now working on the same session state. Each user can now send changes back and forth.

This approach was not without its problems. A fundamental problem with having the collaboration follow a distributed model is how to ensure that both program instances have the same ordering of events. Figure 3.24 shows this. In this example User B makes a change before receiving the change from User A. After both changes have been received User A will see User B's change and User B will see User A's change. To solve this problem Lamport clocks [35][p.624] were used. Any action that effects the session state increments the Lamport clock. The Lamport clocks are then sent with every message. When a message arrives the server checks the Lamport clock and works out where in the history it should go. After reordering the history all the visualisations are redrawn. This also has to be done in the instance of the program that is sending the change. This can be seen in Figure 3.25. When changes are received the history of events is reordered to ensure each user sees the same state. User A sees User B's change and User B sees their own change. Figure 3.26 shows how using a centralised model would remove the need for reordering of events.

The undo history is used to enable the event reordering. When changes are pushed onto the undo stack they are stored as a tuple with the Lamport clock. When a change comes in that may require events to be reordered the undo stack is searched and Lam-

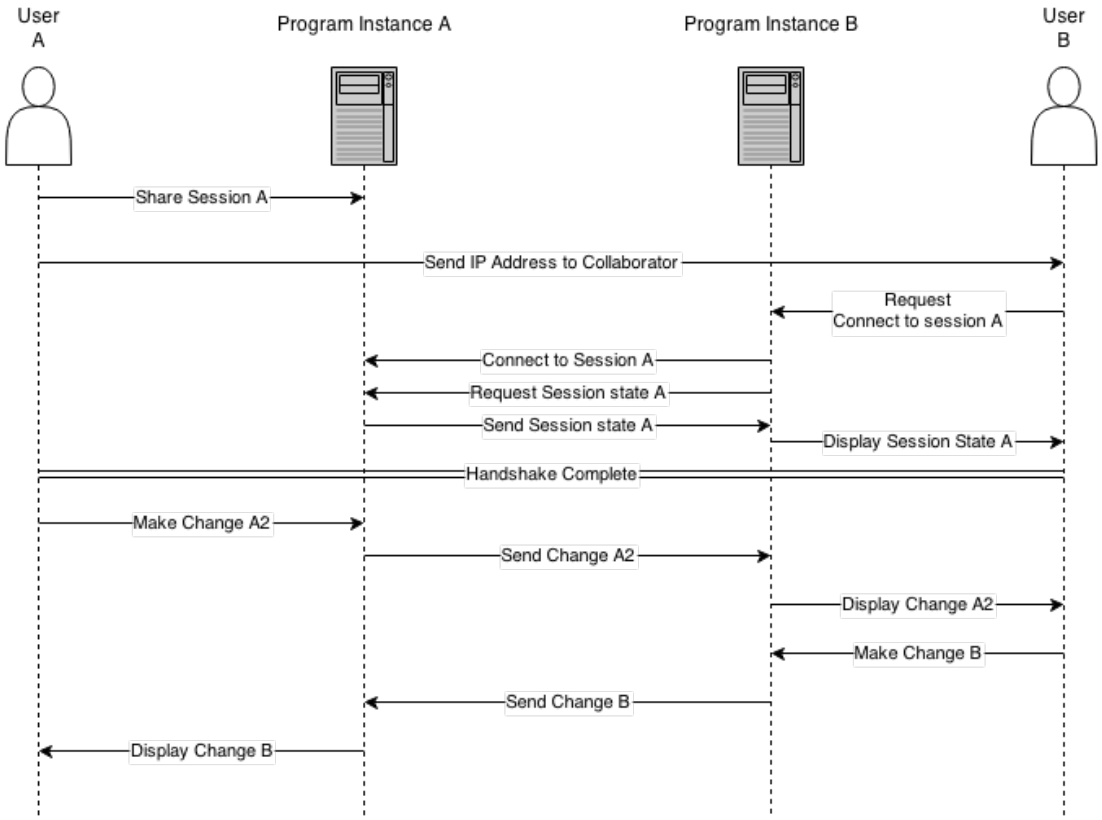


Figure 3.23: Collaboration handshake stage

port clocks are compared. The new data is then inserted between the two appropriate Lamport clocks.

Another issue that was encountered was related to the initial lack of architecture. The session data was often directly modified by methods that were bound to UI elements: they were called when the UI is interacted with. These bound methods were also the methods that called the RPC client to send the message to the collaborator. If the server on the other end then called the original bound method to perform the action it would have resulted in an infinite loop. These methods would also often do more than just modify the state. For example when modifying an annotation the method will find the appropriate annotation and edit it. The client would then send a message saying delete annotation 1. The server would not be able to call the original method as it doesn't take a parameter. The solution to this was to split the methods up more. Now the more typical flow is the UI bound method calling another method that modifies the state. The server is able to call the same method. This removes the need for duplicated code.

3.6 Data Manipulation and Export

In a meeting with the developer of BioDARE it was discovered that biologists found the ability to export their data very useful. This allows them to use their data in external applications as required.

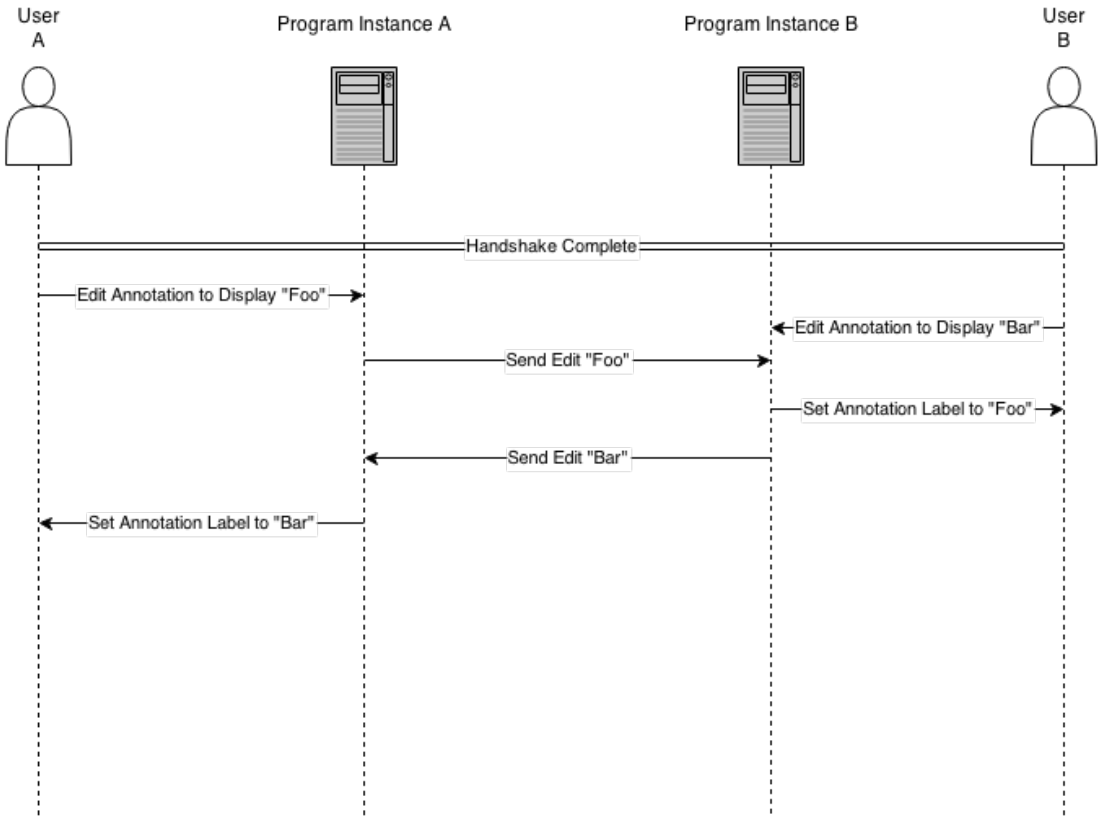


Figure 3.24: Out of sync collaboration

Bio-PEPA's CSV format is non standard. The first lines of a results file contain meta-data about the experiments. In the worst case these lines have the potential to cause an external CSV parser to parse the data incorrectly. In the best case the user has to delete the meta-data manually. Allowing them to export the data, without the meta-data, makes it easier for them by removing potential errors and unnecessary steps. The normalised data is also exported alongside the original data.

Another feature that the developer of Bio-DARE recommended was allowing the biologists to normalize their data. Biologists will often normalise their data in the y-axis. This puts all of the data on the same scale. This makes the data much easier to analyse if there is a significant difference in the scale. Without it the user would be required to swap between different zoom levels. Normalising the data makes the user's analysis much easier.

There are different normalisations that can be applied. The data can be normalised to be zero mean and unit variance as it is for Section 3.4. Another normalisation method is to normalize all the data to be between zero and one. This can be calculated using the minimum and maximum data values from the species in the results. BioDARE allows the user to choose what type of normalisation they would like to use on their data. For this project zero to one normalisation is the only normalisation that the user can have displayed on the graph. Figure 3.15 shows the difference in appearance between a graph and its normalised version. It was decided to only offer one normalisation option to make the choice simpler for the users. They do not have to think about which

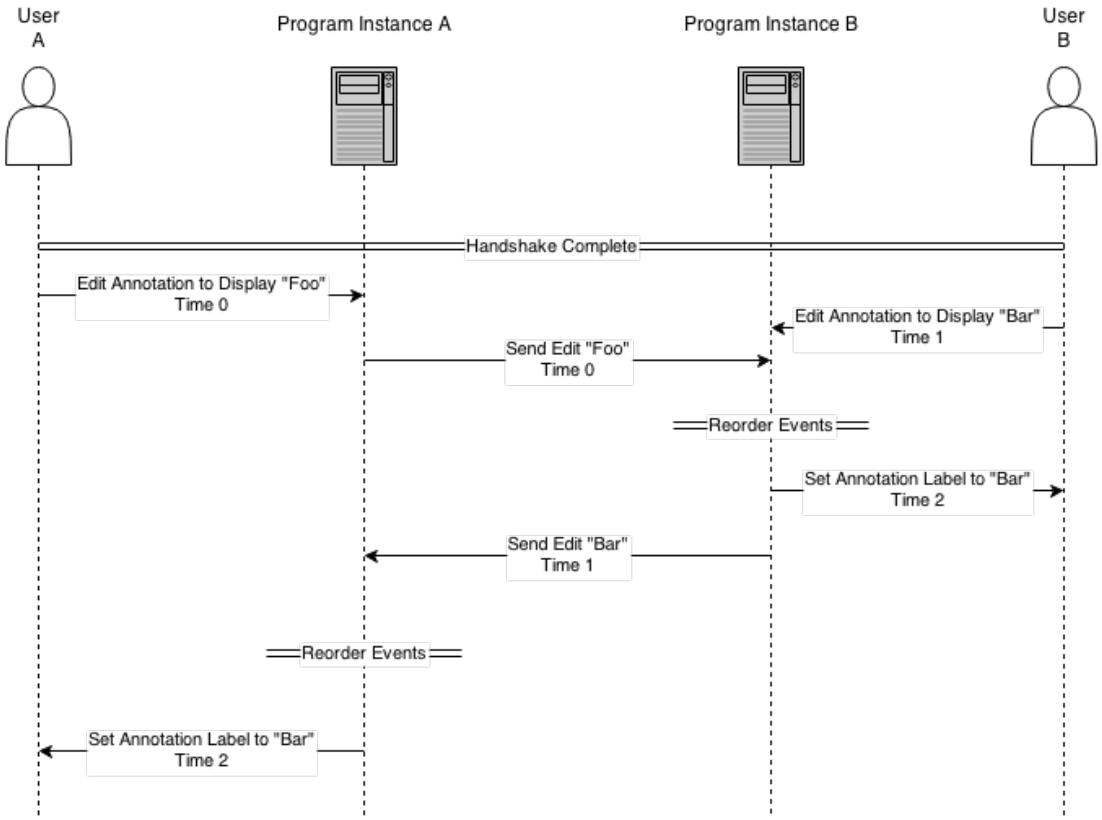


Figure 3.25: Use of Lamport clocks to keep collaboration in sync

normalisation they should apply. In the future if desired more normalisation options could be added. Currently if they want to use a different normalisation the user can export the data and normalise the exported data using their desired technique.

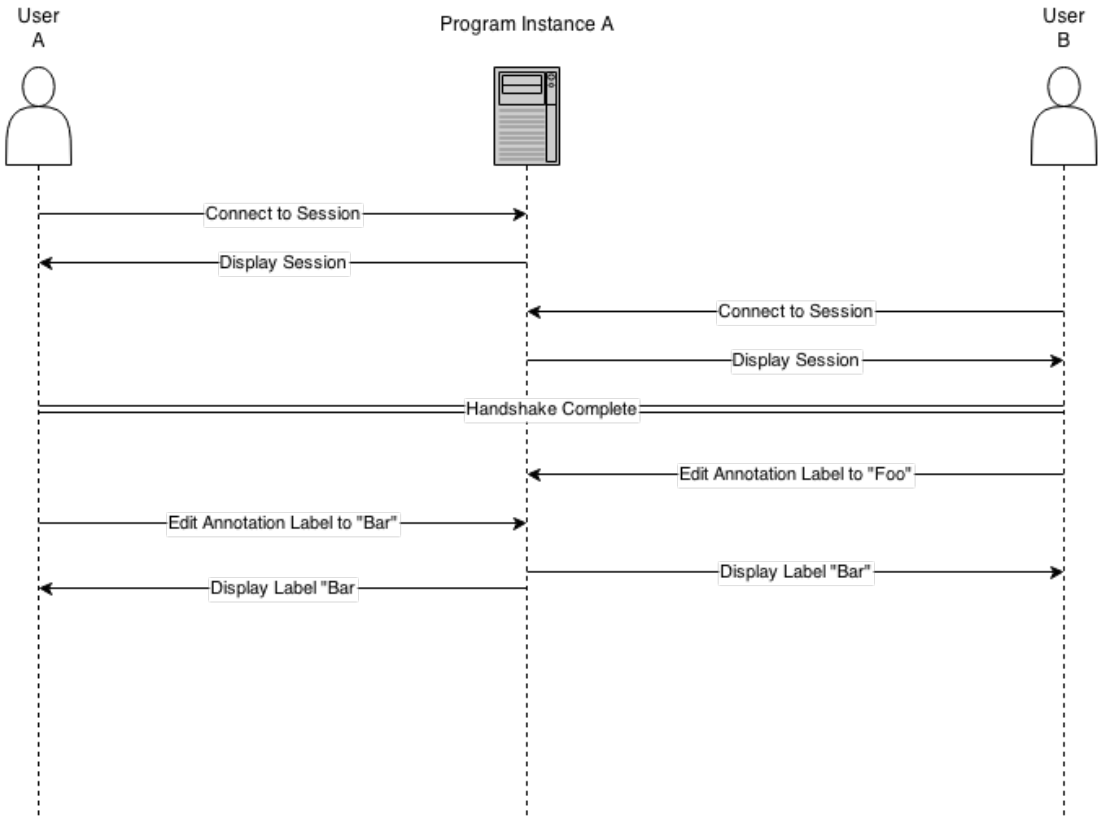


Figure 3.26: Communication flow in a centralised collaboration model.

Chapter 4

Evaluation

This section relates to the evaluation of various aspects of the project. The Src researcher whom the project originally focused on was off on maternity leave for the duration of this phase of the project so it was not possible to perform any evaluations with them.

4.1 First Evaluation

The first evaluation in the second phase of the project occurred in November 2013. The user group was made up of two biologists: one who had taken part in the first evaluation meeting and one person who had no knowledge of the project.

4.1.1 User Group

The original evaluation strategy was to use insight-based evaluation. This is a technique proposed by Chris North [36]. He proposed giving the program to non-expert users. They then see what they can learn from visualisation. These insights are then evaluated by an expert user. The program can then be evaluated by the quality of insights that the non-expert users were able to glean from the visualisation.

As the domain expert was not available, insight based evaluation was not possible. A more traditional approach had to be taken. Before the evaluation a typical scenario that a user might encounter was prepared. The task was to open a file, annotate it and run the animation visualisation, and attach supporting documentation. The task was prepared at two levels of instruction. The first level was a paragraph of text that described what was to be done. The second level was a step-by-step list of instructions to perform. These tasks can found in Appendix **??**. The user group was observed as they attempted the task and were offered assistance when required. Afterwards the user group was given a questionnaire to fill in about their experience. After filling in the questionnaire there was a discussion on their answers and any further thoughts that they had. The questionnaire and results can seen in Appendix **??**

The task was prepared at two levels to try and gauge how easy the program is to use. The users were first presented with the textual description and if they had been unable to complete the task with this then they would have been given the step-by-step instructions instead. The users were able to complete the task from the textual description alone. This is a good sign that the new tool is usable.

Some issues were encountered:

- The users were unfamiliar with `MacOS` – Both users were unable to locate the menu bar as it is not attached to the program as in Windows. Future evaluations were intended to be performed using Windows. There were, however, issues in setting up the program on Windows. Future evaluations happened on `MacOS`.
- The users were unclear as to what was going to happen when annotating. When annotating the graph with an arrow the user had to click twice to place it but there was no indication of this, nor was it clear to them which way the arrow would be drawn. This was subsequently fixed. Different cursors are used to give feedback to the user that they should click, and rather than just relying on two clicks with no information as to where the arrow is going to point, after the first click (which places the tail of the arrow) an arrow is drawn that follows the cursor until the second click placing the annotation.
- Lack of ability to edit, move, or delete annotations. Once an annotation was placed it was there permanently. The ability to edit annotations was always planned, but had not been implemented in time. But the amount of frustration it gave the users was very high. It was a principle in all three of Norman [23], Neilson [24] and Schniederman's [22] lists that a user should be able to fix mistakes. Since the evaluation, editing and deleting of annotations have been implemented. This means any mistakes can be corrected.
- Initially the users were confused by what all the buttons on the `matplotlib` toolbar did. After discovering the tooltips, and seeing what effect the buttons had, they were comfortable with them. If a user were to do something they did not intend they are able to undo it. All the `matplotlib` built-in buttons on the toolbar can be undone and redone from the toolbar. Any buttons implemented for this tool are covered by the undo and redo functionality implemented across the whole program. Being able to recover from their actions on the toolbar means no hindrance to discovery and so needs no further action. It would be desirable to have the two undo methods unified but a way to do this could not be found.
- The users were confused by some of the terminology. In particular they could not distinguish between "save graph" and "save model". These items in the menu have now been grouped more carefully to help the user distinguish them. A related issue was worrying that "save graph" was going to override the results file. To rectify this the menu items that create new files have been renamed "export ...".
- The users struggled to start a new session. When asked for a title they did not know what the title was going to be used for. When trying to add files, rather than use the add files button in the dialogue, they tried to use the file menu.

Having two routes into the visualisation seemed to be confusing them. Now the file menu open file has been removed. To create a visualisation the user has to go through the new session wizard.

- When placing species in the cell one of the users did not understand what they were being asked to do. One of the users did understand. To fix this user input has been removed from the process. The species locations are parsed automatically from the model. This has required the model file to be added to the session by the user.
- They liked the animation feature and thought it would be very useful. One of the users did their PhD in transport and expressed a desire to have had this feature during the PhD. They did feel that it wouldn't be useful directly for papers, but that it would be useful when deciding what to include in a paper.
- One of the users asked if there was a map of the cell. When presented with the model visualisation they thought that it did look nice, but they were unsure of its usefulness. The model viewing has since been merged into the animation visualisation.
- The results from the questionnaire indicated that both users thought the tool's appearance was good. The tool was average in difficulty to use – neither easy nor difficult. The annotation buttons on the toolbar were clear as to what they did. It was obvious how to attach supporting files. Both users thought that it is very useful to attach files to the session so that they can be easily emailed to a colleague. They thought it would be useful to have the graph automatically annotated, but they wanted to the ability to disable any automatic annotations.

4.1.2 Personal Evaluation

At this stage in the development the program was in a state where some existing functionality had been broken. This had not been noticed during development. This highlighted architectural flaws in the code. There were multiple paths through the program that data was taking. Code was also duplicated across the program. The majority of these bugs have since been ironed out and the duplicated code removed. The code has a better architecture. At the time of the evaluation with the users not all the features could be tested with them – mainly the plot preferences dialogue. These features have since been fixed and they were evaluated by the users at the next meeting.

Having the users use the program also highlighted a number of usability problems: menus being badly organised and named, features such as annotation relied on assumed knowledge to work them. All this created an unfriendly environment for the user. This was due to losing sight of the need for usability during development and when testing new features not removing the knowledge of the code from my mind. After this evaluation the three lists of usability [22][23][24] were focused on again and the code was reviewed and the principles applied.

The positive feedback on animation and annotation, two of the core new features, was

excellent validation of the new work.

4.2 Evaluation 2 - Start of Second Semester

The second evaluation in the second phase of the project occurred in February 2014. There were two evaluations this time. One with the same group of biologists as before and another with an Informatics researcher who uses Bio-PEPA.

Since the previous evaluation work had been done to fix the issues encountered and two new features had been added: annotation of the animation and data normalisation and export.

4.2.1 User Group

As with the first evaluation a task was prepared at two levels of instruction. The task can be seen in Appendix **??**. Afterwards the users were given a questionnaire to fill in. The questionnaire and results can be seen in Appendix **??**. They were also given a chance to use the program without following a task, giving them a chance to explore. Again, after the questionnaire a discussion was held.

The findings of the evaluation were as follows:

- There were inconsistencies in the text within the tool that is intended to guide users. This did not appear to cause confusion, but it was noticed by both of the users. This was fixed. All the user facing text in the project was checked and changed if inconsistent.
- The users kept trying to drag annotations on the graph after creation. The users wanted to be able to drag the annotation to a new position to fine-tune the position. They attempted to do this by left clicking on the annotation and dragging it. This issue has not been solved. Currently if the user places an annotation in the wrong location they can delete the annotation and put a new one in the correct place. For the user this is not the most desirable solution, but it was felt that the time could be better spent in other areas of the project. This would, hopefully, be implemented in future work.
- The users found the new model visualisation much more informative. This is a positive as the reason for implementing model visualisation was to help the users understand what happens in the model more. The users did however think that adding in labels of the compartments would be useful as they may not be familiar with the model. The labelling of compartments has since been implemented.
- The users found the task to be easy to complete. This is a good result. The tool needs to be easy to use. They did not give full marks for ease, so there was still room for improvement.

- The users found it much easier to start a new session using the wizard. This was a task they had struggled with during the previous evaluations. There were still aspects that they did not discover such as being able to select multiple files. Guidance text has since been added to the wizard pages.
- The users indicated that they found annotating animations easy and intuitive. This is positive. It is a new feature and the users indicated little need for change.
- The users uncovered some bugs in the system. The bugs included: not all types of annotations could be placed on the plot and the colours of the cell segments not being updated after changing the colour of the line. All bugs discovered have since been fixed.
- The users were very much in favour of being able to use plots as queries. This provided justification for pursuing this as a project goal.
- The users were very much in favour of being able to collaborate in real time. This provided justification for pursuing this as a project goal.
- The users found it easy to normalise and export the data and indicated that they thought it would be a useful feature to have.
- The users were pleased by the addition of undo functionality. They appeared to be much more willing to make mistakes knowing that they could undo their mistakes.
- The users were pleased that since the last evaluation the ability to modify and delete annotations had been added.

4.2.2 Bio-PEPA Developer

The evaluation with the Bio-PEPA developer was similar to the evaluation with the user group. The developer was given the same task and after performing the task there was a discussion about their performance and their feelings about the tool.

- It was not clear to the user what the new model visualisation was displaying. This has been solved by adding text to the model visualisation page explaining what the user is seeing.
- The user uncovered a bug in how annotations are rendered when the graph is viewed with the data normalised. The problem and the solution are discussed in Section 3.3.1.

4.2.3 Personal Evaluation

The results from this evaluation were positive. The users seemed much more comfortable using the tool. This was hopefully down to the efforts made to improve the usability. There were still too many bugs being discovered by the users. This was addressed by spending a significant amount of time on tracking down and fixing bugs.

The positive reception to the ideas for plots as queries and real time collaboration led to them being added as project goals.

4.3 Evaluation 3 - End of Second Semester

The third and final evaluation in the second phase of the project occurred in March 2014. There were two evaluations this time. These were the same as the previous evaluation: the user group and an Informatics researcher who uses Bio-PEPA. Given that this was the final evaluation a further focus was placed on evaluating the users' preferences between this new tool and the Eclipse plugin.

Since the previous evaluation a sizeable amount of work had gone into improving the stability. The new features that had been added were plots as queries and real time collaboration. These could not be evaluated by the users. Plots as queries could not be evaluated as it has not been tied into the UI. Real time collaboration could not be evaluated with the users as it would have required another laptop. The layout of the program had changed and this was tested by the users.

4.3.1 User Group

Like the other evaluations the users were given a task to complete. The task can be found in Appendix **??**. Afterwards there was a questionnaire and a discussion. The questionnaire and results can be found in Appendix **??**

The findings were as follows:

- The users liked the appearance of the tool. Since the previous evaluation significant effort had been placed on making the user interface more aesthetically pleasing. The users strongly preferred the new UI. This justifies the effort put into the UI.
- The users indicated that they found the task easy to complete. They were not confused by the new layout.
- Both users indicated that they would rather use this than the Eclipse plugin. They did both give the proviso that they have zero experience with the Eclipse plugin beyond what they were shown in the evaluation. In the evaluation they were shown an example of a graph that the plugin generated. Both users preferring to use this new tool rather than the Eclipse plugin was excellent validation of the project aims.
- Both users were unable to give feedback about features they felt were missing or features they would like to see as they had not spent enough time with the tool. Further evaluation time with ethnographic studies would be required for this.
- When creating an animation annotation the users set the duration to be after time zero. They were confused afterwards as to why the annotation was not

immediately visible on the cell. It had been hoped that it would be obvious that if the current clock was not in the duration of the annotation then it would not be visible. This was solved by updating the clock to be the start time of the annotation being created. This means it is always visible to the user that an annotation has been successfully placed.

- The slowness of the tool caused the users some frustration. They would frequently want to perform the next action, while the program was still trying to update the visualisations. This is a problem that has been discovered with matplotlib – it is quite slow. This has not been resolved.
- The users found the model visualisation more helpful now that labels had been added.
- The users appreciated the addition of helper text to various parts of the UI.

4.3.2 Bio-PEPA developer

The user was presented with the same task as the user group. The task can be found in Appendix **??**. After completing the task they were asked to fill in a questionnaire. After filling in the questionnaire there was a discussion about their performance. The questionnaire and results can found in Appendix **??**

- The user felt that the appearance of the tool was adequate, although they far preferred the new UI. They did not rate the appearance as highly as the biologist users. They were unable to explain why they had not scored it higher in appearance. They said they would like to see more headings on the different parts of the UI in the future. This would be resolved if development were to continue.
- The user indicated that they found it difficult to complete the task. In previous evaluations they did not find it as difficult. This was due to the lag in the UI when updating the visualisations.
- When asked whether they would use this tool instead of the Eclipse plugin the user said it depended on what task they were trying to perform. If they were preparing images for publication they would use this new tool. If they were just analysing the model they would use the Eclipse plugin. This was to be expected. Future work would hopefully be done to integrate the workflow between the Eclipse plugin and this new tool. It important to note that this user is not the target user. The biologists are the target audience, their preferences are more important for the evaluation.
- The user would have liked to see the model visualisation being able to handle models that were not necessarily hierarchical. This would hopefully be looked into for future work.
- The user would have liked to have been able to evaluate the tool with more models which covered a variety of scenarios. This was not possible in this evaluation strategy. A direction for future evaluations would be ethnographic studies.

- The user did not discover the context menu for annotations, they had to be told about this. No solution has been found to make this feature more discoverable.

- The user uncovered an issue that made it very difficult to right click on text or circle annotations. These annotation types use point to point Euclidean distance. This meant that the user had to right click near the start point of the annotation. This start point is not obvious at all. This problem has not been fixed. For the circle annotation it would be possible to use the equation of a circle to check if the mouse click is near any point in the circle. For the text annotation it would be very difficult to solve the problem as the text annotation has no concept of its size.

4.3.3 Personal Evaluation

The final evaluation definitely revealed that the project has had positive outcomes. The users were all in favour of the project and would use it in their workflow. The reception of the new UI layout was very positive. Fewer bugs were discovered than in previous evaluations. There are still usability flaws and further, more in depth evaluation is, ideally, needed.

4.3.4 Validity of These Results

The findings from the user evaluations should be taken with a pinch of salt. The tool was evaluated by a small number of users. They did not spend a significant amount of time using the tool. This limits the meaningfulness of the findings. There were not enough biologists or potential users available to perform a significant number of evaluations.

In particular the user's opinions on whether the tool was easy to use or not, and their ease with completing the task are not independent between evaluations. Asking a set of users to repeatedly use a tool and perform a task will lead to them becoming more familiar with the tool and the task and future attempts will be easier for them. Their opinion on ease of use may be built on the experience across evaluations. This is an unavoidable problem when evaluating multiple times with the same set of users. There were not enough biologists available to use different users in each evaluation.

The users were also not able to spend a significant amount of time using the tool. The evaluations performed were a good initial set of evaluations and uncovered a lot of the usability flaws. An ethnographic study would involve users in their natural environment using the tool over a long period of time. This would be a very in-depth and time consuming evaluation technique.

4.4 Self Evaluation

The feedback from potential users was a very important part of the evaluation. There are other important aspects of the evaluation. Metrics exist for evaluating the quality of a UI and its usability. The quality of the visualisations must also be measured. The performance must be evaluated. These aspects of the evaluation can be found in this section.

4.4.1 Evaluation of the Architecture

In Section 3.1.2, the architecture of the tool is discussed. During the second phase of development the program was given an architecture similar to MVC, but with little distinction between the view and the controller. This was a step in the right direction. The program would, however, have a much better architecture if it was fully MVC. This would have made the implementation of real time collaboration much easier. Having a MVC architecture would also make future development easier. This is discussed in Section 5.4.4.

The program architecture suffered towards the end of development when real time collaboration was added as a feature. The code was not fully MVC and collaboration had not been a potential feature during the architectural design. The collaboration feature introduced an entirely new route through the program. Due to the project time table there was no time to refactor the architecture. This led to collaboration being somewhat bolted on. Some of the functionality necessary for real time collaboration was put into the singleton object. This was less than ideal. The better solution would have been to create a controller and put the functionality there instead. This was not possible due to time constraints.

4.4.2 Meeting HCI Principles

Norman [23], Shneiderman [22] and Neilson [24] each have a set of principles for UI design. Whilst planning this project it was decided that these principles would form a useful evaluation metric. The principles and how this project meets or does not meet them are detailed in this section. There are overlaps between the sets of principles. If there is an overlap it has been indicated.

Striving for consistency is a principle in all three lists. It suggests that platform conventions should be followed, similar actions should have similar effects and terminology should not change across the UI. Conforming to platform conventions has not been strictly followed in this project. The intention was to be cross platform. An attempt was made, with respect to the menubar, to follow `MacOS` guidelines. However there are not enough menu options to make this obvious. Terminology has been kept as consistent as possible across the UI. Efforts were made for similar actions to elicit similar results, however this was not always successful. There is a big inconsistency

between built-in `matplotlib` functionality and functionality implemented by this tool. For example, there are two different undo mechanisms.

Enabling shortcuts is principle in Shneiderman's and Neilson's principles. Shortcuts are there for expert users to save them time. There are standard shortcuts, i.e. `Ctrl-S` for save. Standard shortcuts can be considered user friendly. Custom shortcuts are not user friendly, but they enable an experienced user to speed up their workflow. Standard and custom shortcuts were implemented in this tool according to `wxPython` specification.

Offering informative feedback is in all three lists of principles. This is so the user is kept aware of system state. They should be receiving feedback from every action. Feedback was, sometimes, a natural byproduct of the features; for example, when loading a results file the feedback that the load operation has been successful is that a graph appears on the screen. If the graph does not appear then something has gone wrong. Other forms of feedback have been explicitly added. These include:

- When adding annotations the cursor changes to indicate to the user that they can interact with the graph in a different way.
- The title bar text changes to display "unsaved" when the user makes a change and then changes back to "saved" when a successful save has been performed.

Designing dialogues to yield closure is a principle in Shneiderman's principles. It calls for encapsulating sequences of actions with dialogues. Finishing the dialogues allows the user to clear their mind. The new tool makes significant use of dialogues to improve the user experience.

Offering simple error handling is a principle of Shneiderman's and Neilson's principles. The user should be prevented from arriving at an error state. This was a problem throughout the project. During development, any error states that were found were removed. This was done through exception catching and more defensive programming. If new error states were discovered the same techniques would be applied. If the user performs an action they did not mean to, which could be classed as an error, they are able to undo it.

Permitting easy reversal of actions is a principle in Shneiderman's and Neilson's lists. It is intended to give users a way of escaping error states by reverting to previously working states. This has been implemented through undo and redo. Copies of the data dictionary are pushed and popped onto the stack. Copies are pushed onto the undo stack on any atomic change the user makes. This gives the user a full session history to go back through.

Supporting internal locus of control is one of Shneiderman's principles. It calls for making the user feel in control of the system and having the user instigate actions rather than respond to actions. The user is the instigator of all actions in the tool. The sense of control will be lost when collaborating as actions will be performed that they did not instigate. This is unavoidable and is mitigated by the fact that the users have to instigate a collaborative session.

Reducing short-term memory load is a principle in Shneiderman's and Neilson's lists. It calls for keeping UIs simple. It is important that users should be able to *recognize* what action needs to be performed, not *remembering* what action has to be performed. This has been done by clearly and effectively labelling UI elements and by having a shallow structure. The user is not forced to go through deep menus and dialogues to perform actions. When implementing UIs that are optimised for recognition it is important not to end up in a Norman door situation [37][p.87]. A Norman door is a door that has been particularly poorly designed and is labelled push or pull, rather than made clear through the design of the door. Some minor Norman door situations have found their way into the project. The most obvious one is in the session wizard on the results file selector page. The user is able to select multiple files. During evaluations the user did not discover this, so helper text was added telling the user that multiple files can be selected. It had been assumed to be obvious, as it is a feature in many programs that you can select multiple files.

Constraining the user is one of Norman's principles. It suggests finding ways to restrict what actions a user can perform during use of the program. This can be used to help guide the user through the program. This has been accomplished in this project through the use of wizards and by enabling and disabling various UI elements.

The first evaluation of the second phase of the project unearthed that the users struggled to choose the correct action as there were multiple ways of performing the same action that had slightly different use cases. For example, at one point results files could be added via two different menu items. They were not constrained enough. These multiple paths have been removed. Now there is only one way, initially, to open results files. To help guide the user further, UI elements are enabled and disabled as appropriate. Now when the program is first loaded the only action a user can perform is to load a session or start a new session, or join a session. Afterwards other UI elements are enabled to allow the user to start using the tool effectively. This guides the user through their use of the program.

To help guide the user when they first use the program a session wizard is created. This replaced a series of separate menu items that the user previously had to navigate. The new session wizard would typically be the entry route in the first time a user runs the program. It was therefore important that it helped them understand what it is they are doing. The process of setting up the session should also be as easy as possible so that the user does not dislike the prospect of using the program. The session wizard can be seen in Figure 4.5.

Providing mappings to the real world is one of Norman's and Neilson's principles. It suggests that the program should 'speak the user's language' and use words, phrases and concepts that are familiar to them. The cell cross-section animations are an example of that in this program. It maps the results data into a form that is more familiar to biologists.

Making use of affordances is one of Norman's principles. Is calls for using visual clues to help guide the user. Different mouse cursors were used to indicate to the user that different functionality could be performed. The principle of affordances in linked to the principle of reducing short term memory load, as affordances enable recognition.

Having an aesthetic and minimalistic design is one of Neilson's principles. It calls for not having unnecessary information in the UI as it competes with the relevant information for the user's attention. This is quite a subjective criteria, however the user feedback would seem to indicate that it has been met.

Providing help and documentation is one of Neilson's principles. It calls for having documentation available for the user in case they need it. This project has no user documentation. This is because the time that could have spent writing documentation was better spent implementing innovative features. If time was not a factor in the project then user documentation would have been written.

The new tool has been quite successful at meeting these principles. We can consider therefore that it has a good interface, with room for improvement. The area that most needs improving is enabling recognition without the use of text explaining to the user what to do. More use should be made of visual cues. This also saves the user having to read lots of text.

4.4.3 Visualisation Quality

Edward Tufte's The Visual Display of Quantitative Information [38] is a seminal work in the field of data visualisation. In his book Tufte outlined a number of metrics that can be used to evaluate the quality of a visualisation. Given that this project is primarily a data visualisation tool it is important that visualisations are of a high quality. Tufte's metrics have been used for evaluating the quality of the visualisations in this project.

Tufte provides a checklist for graphical excellence [38][p.13]. These are set of points that describe what a visualisation should be. They therefore can be a useful evaluation metric for visualisations.

According Tufte graphical displays should:

- Display the data. This was accomplished in the new tool.

- Make the user think about the visualisation, not how it was created. This is a very subjective metric. It is believed to be more relevant to more complex infographics rather than graphs.
- Avoid distorting the data. This was accomplished in the new tool. Data is not distorted. The displaying of the normalised data, in some circumstances, fall foul of this guideline. When the data was normalised some plots appeared to grow in size. The shape was preserved, but the scale was changed. This gives an appearance of a larger change. This is unavoidable when normalising data. It is also a user initiated action, it is not hidden from them.
- Present many numbers in a small space. This was accomplished. Thousands of data points are plotted on the graph.
- Make large datasets coherent. This is believed to have been accomplished, the data is plotted in line graph form, which is, typically, easy to interpret. One of the aims of the project was to make the datasets coherent for those who are not comfortable with traditional graphs. For these users, the cell level animation makes the data coherent.
- Encourage the eye to compare different pieces of data. This is another metric that seems more relevant to complex infographics. It was accomplished in this tool. Multiple species can be plotted on the same graph. It is natural to compare the data.
- Reveal the data at several levels of detail. This was accomplished. We have the traditional graphs and cell cross-sections. The graph provides detail on the structure and the cell cross-sections provide a broader overview of the data.
- Serve a clear purpose. This tool does have a clear purpose. The goal of the project was to "Visualise the results of dynamic time-series models of intra-cellular behaviour based on biochemical reactions". This goal has been met. This is discussed further in Section 5.1.
- Be closely integrated with statistical and verbal descriptions of the dataset. The ability to add annotations to the visualisations accomplishes this goal. The annotations provide the verbal description and the the graphs provide the statistical description.

We can see that the tool has met most of these guidelines. This indicates that the visualisations offered are good quality visualisations.

The Golden Ratio is suggested by Tufte as the proportion to use for the size of a graph [38][p.189]. The golden ratio is also believed to be aesthetically pleasing to humans. The aspect ratio of the graph in the tool approximates the golden ratio.

Data Ink is a metric for determining how much redundant information is in a visualisation [38][p.93]. This has not been formally calculated as the definition of redundant information is not well defined, especially with respect to annotations. The concept of

reducing redundant information is still an important aspect and was taken into consideration. There is no unnecessary decoration on the graphics. Only relevant information is displayed.

Lie Factor is a metric for whether a visualisation is misrepresenting the information [38][p.57]. It is calculated by $\frac{\text{size of effect in graph}}{\text{size of effect in data}}$. This was identified as an evaluation metric because correctly representing the data is vital. The visualisations available in this tool do not misrepresent data. The visualisations are plotting the data. Lie factor is more of an issue in fancier graphics. The displaying of the normalised data, in some circumstances, fall foul of this guideline. When the data was normalised some plots appeared to grow in size. The shape was preserved, but the scale was changed. This gives an appearance of a larger change. This is unavoidable when normalising data. It is also a user initiated action, it is not hidden from them.

4.4.4 Finished Product

As well as evaluating the program as a whole, it is necessary to evaluate the major components.

4.4.4.1 The User Interface

Figure 4.1 shows the main screen of the UI at the end of the development phase. Figure 4.2 shows the previous version of the UI. The widgets have, now, been placed in a more organised and consistent manner. This style has been continued through the rest of the screens in the UI. A heading to the legend panel has been added. This indicates to the user from the start where the legend will be placed. It has also been much more prominent in an effort to draw the users attention to it. The legend panel has been moved over to the right hand side to remove the boxed in feeling that the previous UI gave.

Figures 4.3 and 4.4 show the dialogues that are presented to the user for creating an animation annotation and changing the preferences of a line. There is a consistent style to the dialogues to help the user feel comfortable. In both dialogues related widgets are grouped together. It was not felt necessary to use wizards for these functions. Wizards imply a set of steps that need to be followed. For the plot preferences dialogue there is no concept of order. For the annotation dialogue there are not enough steps to warrant a wizard. The plot preferences dialogue in Figure 4.4a spawns another dialogue when the user initiates a change colour action. The spawned dialogue can be seen in Figure 4.4b. This is one of the most user unfriendly aspects of the entire UI. Whether you choose a new colour or not, the only way to exit the dialogue is by pressing the red close button. This is a standard cancel action. It does not inform the user that their action is going to be successful. This confused all users. This is a default dialogue provided by `wxPython`. Ideally it would have been fixed, but there was not time.

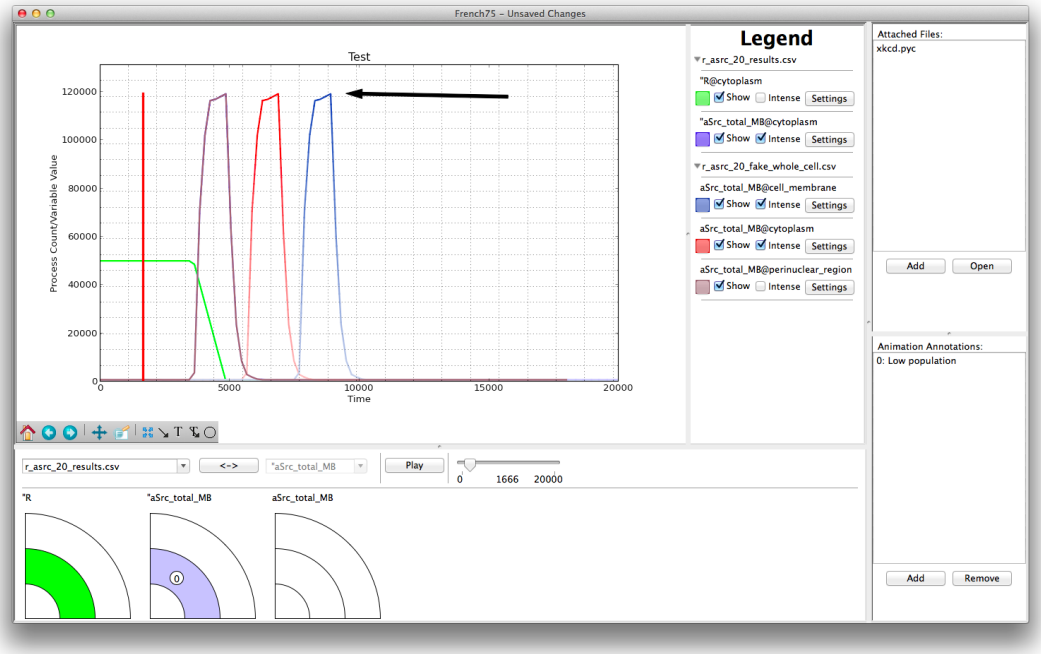


Figure 4.1: Final UI

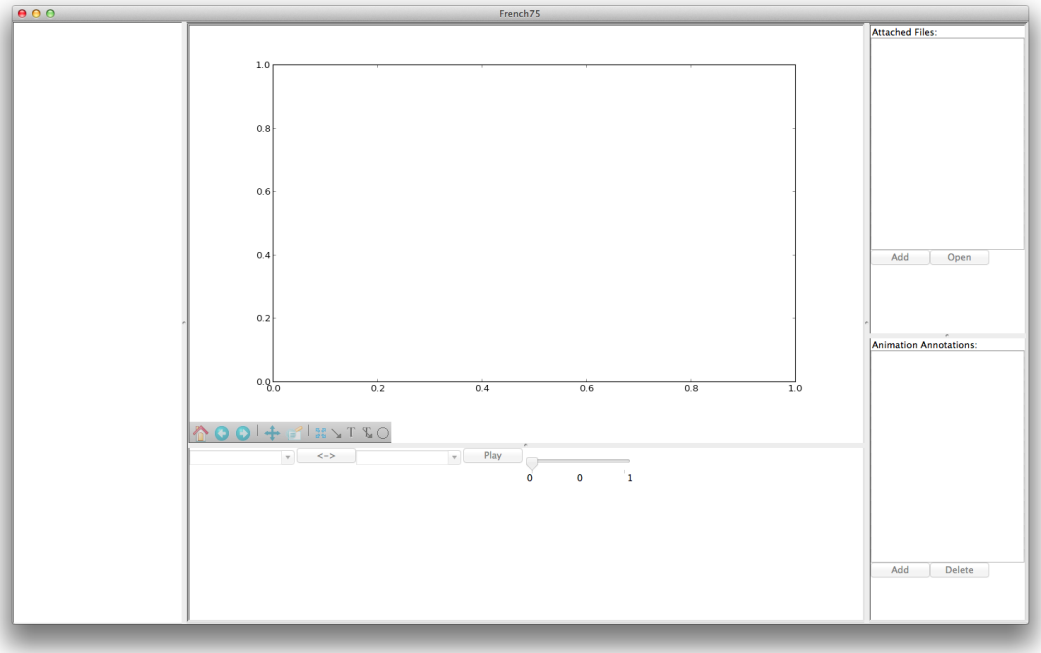


Figure 4.2: Earlier UI

Wizards were used to guide the user through the process of setting up a new session. The wizard can be seen in Figure 4.5. A predecessor of the wizard can be seen in Figure 4.6. The old dialogue was not user friendly. The user is presented with a long list of boxes that need to be filled. This is overwhelming to a user. It is also quite

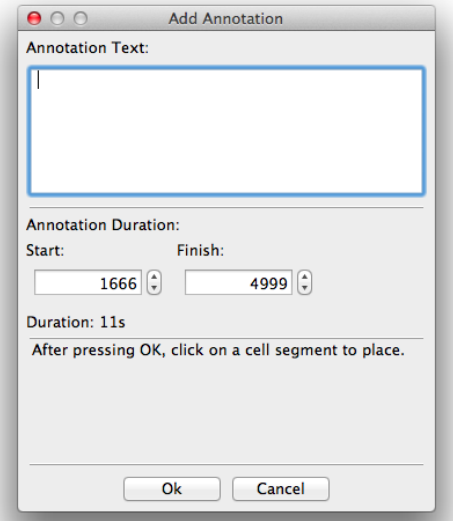
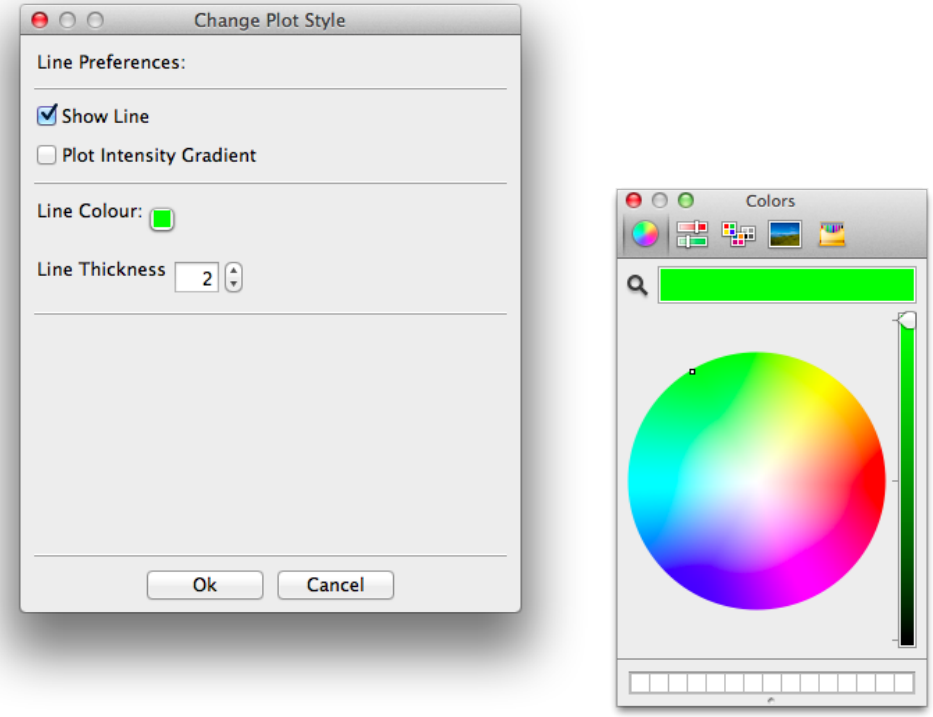


Figure 4.3: Add animation annotation dialogue

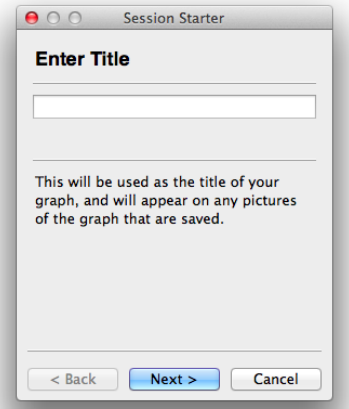


(a) Line preferences dialogue (b) Colour chooser dialogue

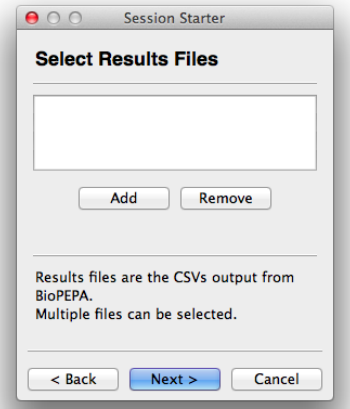
Figure 4.4: Line settings dialogues.

ugly. The new session wizard fixed this. The wizard follows the style of the other UI screens, providing the user with a consistent experience. Having different pages in the wizard for each step of the new session process means that the user does not need to

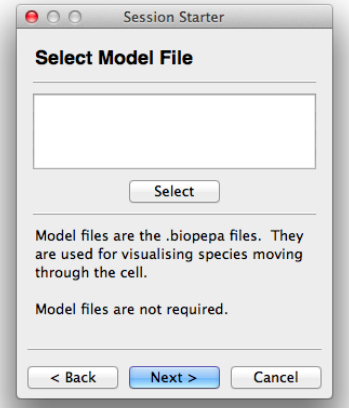
remember the whole process. They can focus on the current step.



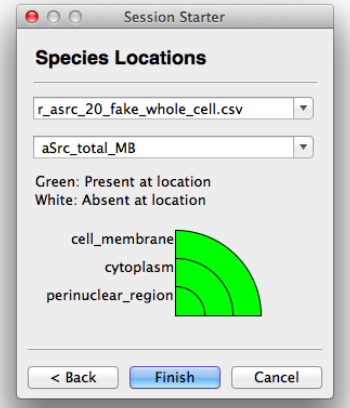
(a) Wizard title chooser page



(b) Wizard results selector page



(c) Wizard model selector page



(d) Wizard model viewing page

Figure 4.5: The session wizard that guides a user when setting up a session.

Form validators are used on the title fields and results fields. The validators ensure that a title and at least one results file are included. This stops the user from creating an invalid session. There is also help text on each page of the wizard instructing the user on what actions they should take.

The model viewer page in Figure 4.5d is similar to the cell cross-sections used in Section 3.2. There are two drop down boxes. These drop down boxes control what is displayed in the cell cross-section. Species in files can be selected. The cell cross-section displays which compartments in the cell the species is present in. The compartments are parsed from the model file. This means that the model file is a requirement for model viewing and cell level animation. This cell cross-section has replaced the previous model visualisation. The new model visualisation has two purposes. First, the user can sanity check that they have matching results and model files. If a section of the cell

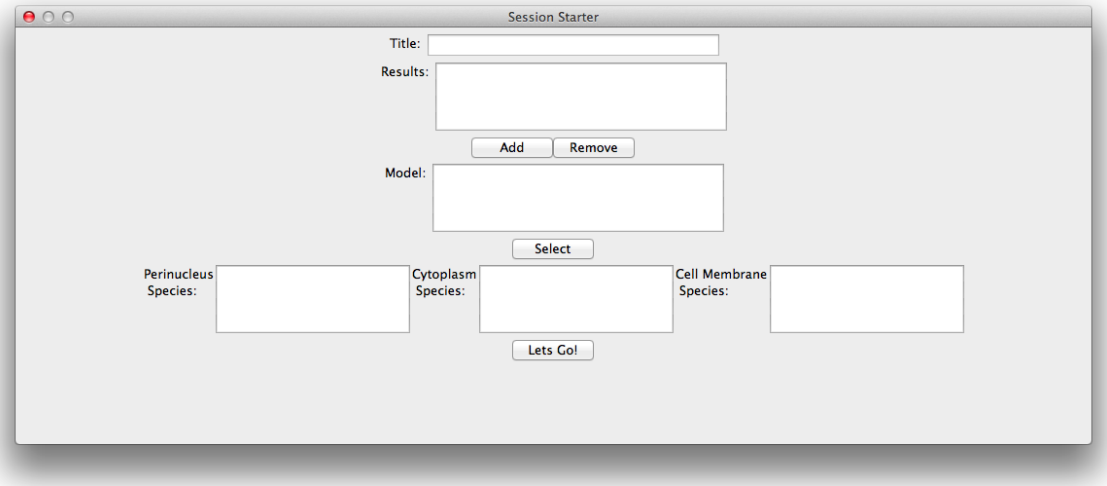


Figure 4.6: Session dialogue, the precursor to the session wizard.

has been left white then it acts as a cue to the user: if they were expecting the species to be present at that location then they know something has gone wrong. Second, they can see how the model is structured. The sections of the cell are labelled allowing the user to know what areas of the cell a species is present in. This will hopefully increase their confidence with visualising and analysing the results from the models. Before the compartments were parsed from the model file automatically there was an user unfriendly system where the user had to input where in the cell a species is. This was time consuming, difficult to use, and quite brittle. At the time it assumed that there would be three compartments for every species, which is not a valid assumption.

4.4.4.2 Collaboration

It was not possible to perform an evaluation with any users of this feature. Only one laptop was available for the evaluation sessions. Although it is possible to run two instances of the program and start the collaboration feature, it is is not a representative demonstration as the two instances must be switched between and the changes are not seen in real time.

The evaluation of this feature has therefore been focussed on the responsiveness of the collaboration and the coverage of the functionality that can be collaborated on.

The implementation of real time collaboration is quite effective. Almost all functionality that can be in a solo session has been implemented for collaborative work. The only feature which cannot be used in the collaborative mode is the large plot. It was not possible to implement this feature because of the issue of closing the large plot dialogue. The event that closes the dialogue would be the event that sends the message to the collaborator. The collaborator would then have this method called. There is then an infinite loop. The displaying of a large version of the plot is a minor feature. Not having access to it in the collaborative mode does not limit the user in their work.

The initial implementation of collaboration was not responsive enough. It used blocking communication, effectively providing a lock around each action that effects the visualisation. This involved waiting for a response from the collaborator to say that the change had been applied. The UI would be appear locked to the user for the duration of this message. The collaboration features were only tested on internal networks where that duration would be minimal. Even with minimal duration it was a frustrating experience being locked out of the UI. Communication to users across non internal networks could only be slower and therefore more frustrating to the user. For this reason non blocking communication was implemented. A thread is now created for each message sent. This allows the UI to not be locked out. It did require losing the mutual exclusion guarantee. This made the UI more responsive, but sacrificed guaranteed correctness. Section 5.3.1 discusses this further.

4.4.4.3 Search Results

Using plots as queries was a difficult feature to evaluate. The ideal evaluation would have a large set of plots that were representative of the data generated by Bio-PEPA. The set would have a subset of labelled similar plots. We could then evaluate by using a plot, with known similar plots, as a query and seeing how many of the similar plots are ranked highly by tf.idf weighted cosine.

In lieu of a real data set, an artificial set was used. A random markovian walk was used to generate a corpus of time series data. This corpus did not have a labelled set of similar plots, so similar plots were generated. The query plot was taken and mutated in various fashions. The mutations matched the criteria that were discussed in Section 3.4. The similar plots generated through mutation can be seen in Appendix **??**.

The experiments were run on this artificial corpus. A number of experiments were performed:

- Experiment 1 – One iteration of tf.idf weighted cosine with 10000 plots and no labelled similar plots. The purpose of this experiment was to see what would be returned as similar if no similar plots were defined.
- Experiment 2 – One iteration of tf.idf weighted cosine with 10010 plots including 10 similar plots generated through mutation. The purpose of this experiment was to see how many of the labelled similar plots are returned as being similar.
- Experiment 3 – 100 iterations of tf.idf weighted cosine with 1010 plots in the corpus. The purpose of this experiment was to repeat Experiment 2 with multiple iterations.
- Experiment 4 – 1000 iterations of tf.idf weighted cosine with 110 plots in the corpus. The purpose of this experiment was to repeat Experiment 2 with even more iterations than Experiment 3.

The results were extremely positive. The findings from each experiment are discussed in detail below.

Experiment 1: The results from this experiment can be seen in Table 4.1. These results have discarded the match between the query and itself. The table shows the top 10 most similar plots found by tf.idf weighted cosine and the measure of their similarity. The plots of these graphs can be seen in Appendix **??**. We can see in some graphs, particularly Figure **??** and Figure **??**, that there are structural similarities between the query plot and the result plots. In some other graphs, particularly Figure **??** and Figure **??**, this similarity is less obvious. This could be an issue of scale that is not obvious to the human eye. It could also be a failing in the method for determining similarity.

Rank	1	2	3	4	5	6	7	8	9	10
line id	256	7017	8341	1380	4997	5462	6648	2298	595	190
similarity	0.1804	0.1765	0.1629	0.1623	0.1608	0.1601	0.1584	0.1575	0.1566	0.1562

Table 4.1: 10 most similar plots from Experiment 1

Experiment 2: The results of this experiment can be seen in Table 4.2. The results have discarded the match between the query and itself. The table shows the top 10 most similar plots found by tf.idf weighted cosine and the measure of their similarity. This experiment contained the 10 mutated similar plots. The 10 mutated plots are the entirety of the top 10. Applying tf.idf to Lin and Li's work [32] appears to have been justified. The three plots that were purely scalings of the original data have been returned as identical. This is the correct behaviour of the algorithm as it is a scale invariant algorithm. The eleventh most similar plot was plot 256, which was the most similar in Experiment 1.

Rank	1	2	3	4	5	6	7	8	9	10
line id	10002	10001	10000	10003	10008	10009	10005	10004	10007	100006
similarity	1.0	1.0	1.0	0.8980	0.8902	0.8525	0.8349	0.7787	0.7086	0.6649

Table 4.2: 10 most similar plots from Experiment 2

Experiment 3: In all 100 experiments the top ten most similar plots found by tf.idf weighted cosine, excluding the query itself, were the mutated similar plots. This is a positive indication that the approach for determining similarity was a valid one. It does, however, indicate that using artificial data may have drawbacks.

Experiment 4: In all 1000 experiments the top ten most similar plots found by tf.idf weighted cosine, excluding the query itself, were the mutated similar plots. This is a positive indication that the approach for determining similarity was a valid one. It does, however, indicate that using artificial data may have drawbacks.

Findings: The evaluation technique does have its limitations. The data is artificial and may differ significantly from real data. This was unavoidable as there was not

enough real data to compare it to. The experiments with multiple iterations were limited in how many plots could be included in the corpus. The generation of the plots was a very computationally intensive process and limited the amount of data that could generated for experimentation.

The mutations used for defining similarity are quite clean operations. In reality the similar plots would likely be fuzzier matches than were used in this experimentation. This would require actual data to investigate. This has possibly affected the results, especially in Experiments 3 and 4.

Despite the limitations of the evaluation technique, the results of the experiments showed great promise for this feature. The results justified the inclusion of the feature and indicated that further investigation would be worth while.

4.4.4.4 Animation

Animation of cell cross-sections requires the model file to be provided. Ideally the user would just need to provide the results file, but the results file does not include all the necessary information. It was felt to be more user friendly to require the model file than having the user enter a lot of information by hand.

Users can use the model viewer, in the session wizard (Figure 4.5d), to preview animation. Once a session has been started the animation toolbar is made available. The toolbar contains a play/pause button. This enables discovery of two features. The line drawn on the graph indicating current time and the time slider. While the animation is playing the user can see the time slider and the line moving. This provides a cue to them that they can move the slider to control time.

The species/file selectors and the button to switch between file-focussed and species-focussed modes are less obvious in their behaviour. Users were unable to work out what these features did. After having the features explained they were comfortable with them. Discoverability was the issue with these features. A solution was not found. User documentation would be a fix for this.

The appearance of the time slider in the UI is a particular weak point in the UI. Many attempts were made to stretch the appearance of the slider widget but these were fruitless.

The performance of the animation was more than adequate. The animation of the cell segments is very smooth. The animation of the line on the graph is also fairly smooth, although choppiness in frame transitions has been noticed. The performance is better than the usual plotting speed of the graph. This is because only a select part of the graph is being redrawn – the vertical line.

Feedback from the users has been very positive for the animation feature. It meets the goal of providing visualisations more familiar to the users.

4.4.4.5 Annotation

Annotation of the graph and the animation had to be implemented separately. They also have to be evaluated separately. One of the flaws of the current annotation implementation is the separation between the different annotations. This is not user friendly as they have to learn two different systems, each with different behaviours.

Annotation of the Graph: Placing annotations is now quite user friendly. The annotation type is chosen from the toolbar and then the graph is clicked to place the annotation. Arrow annotations have an arrow that follows the mouse allowing the user to see the appearance before placing the annotation. This was not possible for the text and circle annotations.

Selecting annotations is not user friendly enough. The need to right click to select an annotation is assumed knowledge. It has not been made obvious enough to the user. No user was able to initiate the context menu without being told about it. No solution was implemented. One fix would be to remove the right click aspect, and instead use a hover menu.

Editing text and deleting provide the user with control over the annotation. More control needs to be provided in the future, in particular the ability to move annotations.

Annotation of the Animation: Placing annotations is reasonably user friendly. The dialogue acts as a guide to the user. Having to click on a cell cross-section to place the annotation was hoped to be obvious as it follows the same behaviour as with the graph annotations. This was not the case. Text has been added to the dialogue to make this more obvious.

There is also too much of a disconnect between the animation panel and the panel containing the list of annotation. This was felt necessary due to space constraints. A more desirable approach would be similar to the graph annotations which are handled entirely from the toolbar, not a separate panel on the UI.

4.4.4.6 Normalisation and Export

This is quite a user friendly feature. This has been evidenced by the users having no trouble with the feature. Normalising the data uses a checkbox menu item to toggle on or off. This provides a visual clue to the user that it is a toggle-able state. It should indicate as well that if they press the menu item and do not like the results that they can press the menu item again and revert to the previous appearance.

Normalisation and export are both operations on the data and so have been grouped together on the menu. The main issue with normalisation was how annotations should behave. The approach was described in Section 3.6. This was an adequate solution and keeps the state visible to the user.

4.4.4.7 Performance

The speed of the tool is not great. It becomes sluggish in some situations. This is mainly due to `matplotlib`. `matplotlib` is the de facto standard for graphing in Python. This, amongst other reasons (detailed in the MPP report), led to it being chosen for the graph visualisations in the project. When features such as plotting the intensity of the gradient were added, `matplotlib` was pushed to its limits as it was being asked to plot thousands of lines. It was discovered that `matplotlib` is not optimised for speed, it is instead optimised for producing graphs of a publishable standard. This poses a dilemma: researchers want to be able to generate quality graphs, whilst the tool wants to offer real time interactivity and customisation. There are tweaks that can be applied to make `matplotlib` faster. There are also alternative graphing libraries that could be considered. Threading was also looked into but caused segmentation faults.

4.4.5 Testing and Logging

The development of the project involved little formal testing and logging. The intention had been to develop tests, but these tests never came to fruition. The development work was too tempting. The lack of tests caused problems later in development. New features being implemented, which would often involve refactoring, would sometimes break existing functionality. These bugs might then not be discovered until the next round of user testing. This was not what the users wanted to be presented with. Having tests would potentially have revealed these bugs.

Another oversight was the lack of logging in the program. This only became a problem towards the end of the project as it got more complex. It would, sometimes, be difficult to recreate a bug that the users had discovered. This made the bug harder to fix. The problem reached its peak during the implementation of real time collaboration. Trying to debug the distributed communication without formal logging was challenging. Print statements were used as a simple form of logging that was somewhat helpful. Having a more structured logging framework in place from the start would have eased debugging throughout the development process.

Chapter 5

Conclusion

This section contains a final look at the project. The goals were looked at and it was decided whether or not the project met them. There is also a detailed analysis of where future work in the project should be focussed. There is also a discussion of any challenges that were faced and any problems that were unable to be resolved.

5.1 Comparison to Objective

The aim of the project was: "to develop a tool to visualise the results of dynamic time-series models of intra-cellular behaviour based on biochemical reactions". This goal has been met. The results can be visualised statically and dynamically. The cell level visualisation applies domain knowledge familiar to biologists to help them understand their results. Additional features to help the user work more efficiently and effectively have been added. With respect to meeting the project goal the tool has been successful.

The users are now able to load the results from their dynamic time-series models of intra-cellular behaviour based on biochemical reactions into the program to be visualised. There are static and dynamic visualisations. These take the form of standard graphs and animations of an abstract view of cells.

The user is able to interact with these visualisations. They can customise them and attach additional information not present in the results files.

The user is also able to manipulate their data and export it.

Prototype features have been added for applying data mining techniques to allow the user to use their data to search databases of time series data and allowing users to collaborate in real time.

Evaluations with potential users have shown positive reception to this new tool and its features and a willingness to use this tool instead of the Eclipse plugin.

5.2 Challenges Faced

Some challenges were faced in this project that were not discussed elsewhere in this report. These are presented below.

5.2.1 wxPython Being Cross-Platform

Python was chosen for this project to allow for the project to be run on any system. A cross platform GUI toolkit was chosen for the same reason. `wxPython` was chosen (details of why can be found in the MPP report). Initially this was successful. The program was running on `MacOS` and `Linux`. As the project was developed further some discrepancies were uncovered between `wxPython` on different platforms. Initially this was coped with by writing different versions of parts of the code for each platform. Eventually this became impractical. Development support for `Linux` was halted. Currently the project has only been tested running on `MacOSX Mountain Lion`.

An attempt was made to test the program on Windows, but there were problems installing the necessary Python libraries.

If this project was to be undertaken again, a different approach for the UI would be chosen. A browser based approach would be most suitable. This is discussed Section 5.4.3. `HTML` would become the primary source of the UI. The user's browser would render the UI. Different browsers should be able to render the same UI on different platforms, as long as the `HTML` is standards compliant.

5.3 Unsolved Problems

5.3.1 Complete Ordering Without Blocking

An initial version of the collaboration used blocking communication. In a blocking mode it was possible to globally order the events. This is described in Section 3.5. Blocking collaboration is far from optimal. A slow network would make the collaboration unusable. The program would lock out the user for the duration of the message and response. Non blocking communication was added later through the use of threads when sending messages. Now a thread is spawned for each message sent between the client and the server. This prevents the UI from being locked out during the communication process. This had the undesirable side effect of not being able to guarantee correct ordering. No solution has been found for this, apart from switching to a centralised model as described in Section 5.4.3.

5.3.2 Distributed Undo Forgetting States

When doing a visualisation without any collaborators the undo and redo functionality works as expected. In the collaborative mode however there is a bug. If we have the user and the collaborator, changes the user makes are sent to the collaborator. If the user issues an undo command their undo history is correct. The collaborator's undo history appears to forget states. This leaves the collaborator and the user seeing different visualisations. This bug appears to be related to the issues that arose when first implementing undo and redo. Python's deep copy had to be introduced. Despite numerous attempts to solve this bug a solution has not been found to resolve the issue before the project deadline.

5.4 The Future

By the end of the project the program that had been developed was in a state that could be used by users. There is more functionality than the Eclipse plugin and it is stable. There is, however, work which could be done if development were to continue. This is detailed below.

5.4.1 Plots as Queries

A prototype system for using plots as queries was developed (Section 3.4). Early results were promising, but further work needs to be performed. The first step would be working with a team of biologists and creating a large evaluation set. This would be a labelled set of plots that are similar or not similar. This will allow the effectiveness to be accurately measured. This would also include tuning of the parameters used, such as feature size and vocabulary size. Other systems for using time series data as queries could also be developed and then the different systems can have their effectiveness compared.

There should also be an online repository of time-series data. This would be processed for use in the search algorithms. This would be done with the technique in Section 3.4. tf.idf, information retrieval, and machine learning in general, all rely on having a large amount of data. One user will typically not have enough data to give the best results. A central repository of data would allow all users' data to be used to gather statistics on the data to be used in the algorithms. Other users' data could also be displayed in the results. This would allow users to discover related work and potential collaborators. Having a central repository also takes some control of the data out of the hands of the user. Part of the reason for not integrating this feature with the UI during this project was that it would have been difficult to track the location of the experiment data. If the data had been uploaded to a central repository then there would be no problems tracking it, as the user would have no say in the location.

5.4.2 More Data Mining and Machine Learning

Enabling search was an early step into what can be done when machine learning and data mining is applied to time series data. Other areas of interest include classification, clustering and anomaly detection [28, 27]. All of these can be built on top of the work that was done in Section 3.4.

These are all features that would allow the user to analyse their work in a more detailed manner.

Again, all of these work best with a large amount of data and so would need to be tied into an online repository.

5.4.3 Put it in the Cloud

All of the collaboration software reviewed (see Section 1.2) worked using a model where there is a single server. This is a much simpler model for collaboration as the only ordering is the ordering on the server.

Adopting this model the for tool's architecture would require some refactoring of the project. The code would need to be separated completely into a server and a client. The server would do the the work: parsing the data, performing the intensity calculations and handling the creation of annotations. The client would just display the information from the server.

Work towards this goal would also hopefully include rewriting the project so that the client could be a browser-based client. A lot of problems arose from `wxPython` not being as cross platform compatible as had been hoped. Rewriting the client to be browser-based should solve the cross platform problems.

The cloud backend would also, hopefully, include a repository for storing the attached files. Currently there is no way for attached files to be sent to a collaborator. They were considered too large to be sent through the implementation of collaboration described in Section 3.5. Without a central file repository we would need some other way of sending files. This comes close to violating both Lett's and Zawinski's laws [39]. These laws state, respectively, that all software evolves until it can send email and that all software evolves until it can read email. They are a sign of feature creep in software. Email would have been a potential solution on how to transfer the files between users, by building an email client into the program. A central file repository avoids the need for this feature creep. Previews of the files could be part of the real time features with thumbnails and title being transmitted. The collaborators can upload and download the full file as necessary.

5.4.4 Improve the Architecture

In Section 3.1.2 the initial lack of architecture is discussed. The architecture was improved and now resembles a MVC architecture. This work on the architecture should

be continued in the future. Making the architecture fully MVC would greatly enhance the work towards moving the tool to the cloud (Section 5.4.3).

Having a MVC architecture would also make it much easier to change interfaces. New UIs could be developed and just connected to the model which would be running in the cloud.

5.4.5 More Customization

Currently, the user is given control over the appearance of lines on the graph. This could be expanded further into the tool. Currently annotations cannot be customized, they can only be black. More annotation options could be offered. Also, different normalisation methods could be included as discussed in Section 3.6.

5.4.6 More collaboration

In addition to changing the model of collaboration from being distributed to centralised, further work should be done towards enabling communication between users. Currently collaborators will have to use a phone or an external chat client to discuss the work. A chat client could be integrated into the program. This would enable communication without needing to switch between applications. This chat client could also be specialised to include features that would be relevant to visualising data.

5.5 Final Remarks

The goal which was “to develop a tool to visualise the results of dynamic time-series models of intra-cellular behaviour based on biochemical reactions” has been met. This has made it a suitable replacement for the Eclipse plugin.

On top of the visualisation functionality, a suite of innovative features have been implemented to allow users to analyse their data quicker and more effectively. They are also now able to collaborate in real time.

The functionality now offered by the tool is unique in the field. No other software aimed at biologists allowed for real time collaboration, and no software allowed them to use their data to find similar data.

There is work required in these innovative features before they are fully ready for the user, but the effectiveness of these of the prototypes demonstrate the potential. The new software can be considered a significant improvement for biologists.

Bibliography

[1] Bio-PEPA. Bio-pepa homepage. http://www.biopepa.org.

[2] Google Drive. homepage. drive.google.com.

[3] Pidoco. homepage. https://pidoco.com/en.

[4] Lucid Chart. homepage. https://www.lucidchart.com.

[5] SynBioWave. homepage. http://www.synbiowave.org.

[6] Uppsala Universitet and Aalborg University. Uppaal. http://www.uppaal.org/.

[7] University of Connecticut Health Center. V-Cell: Virtual Cell Modeling & Analysis Software. http://www.nrcam.uchc.edu/index.html.

[8] Azevedo Lab, University of Houston and MBI, German Cancer Research Center. Cell-O: Biological Simulation Framework. http://www.cell-o.org/.

[9] Mendes Group, Virginia Bioinformatics Institute and Department Modeling of biological Processes, University of Heidelberg and Mendes group, University of Manchester. COPASI: biochemical network simulator. http://www.copasi.org/.

[10] The Systems Biology Institute, Tokyo, Japan. CellDesigner: A modeling tool of biochemical networks. http://www.celldesigner.org/index.html.

[11] Institute for Visualization and Perception Research, University of Massachusetts Lowell and Open Indicators Consortium. WEAVE: Web-based Analysis and Visualization Environment. http://www.oicweave.org/.

[12] matplotlib. Python graphing library. http://matplotlib.org/.

[13] wxPython. http://www.wxpython.org/.

[14] SimpleXMLRPCServer. http://docs.python.org/2/library/simplexmlrpcserver.html.

[15] BioDARE. homepage. https://www.biodare.ed.ac.uk/.

[16] DinaQ. Goldin and ParisC. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation. In Ugo Montanari and Francesca

Rossi, editors, *Principles and Practice of Constraint Programming CP '95*, volume 976 of *Lecture Notes in Computer Science*, pages 137–153. Springer Berlin Heidelberg, 1995.

[17] Kaushik Chakrabarti, Eamonn Keogh, Sharad Mehrotra, and Michael Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228, June 2002.

[18] I. Popivanov and R.J. Miller. Similarity search over time-series data using wavelets. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 212–221, 2002.

[19] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Rec.*, 23(2):419–429, May 1994.

[20] Béla Bollobás, Gautam Das, Dimitrios Gunopulos, and Heikki Mannila. Time-series similarity problems and well-separated geometric sets. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, SCG '97, pages 454–456, New York, NY, USA, 1997. ACM.

[21] Georges Grinstein. Keynote on Integrating Visualization and Analysis. In *Vizbi Conference*, 2012. `http://vizbi.org/2012/Presentations/Georges_Grinstein_Keynote.pdf`.

[22] Ben Shneiderman. Designing the User Interface. `http://faculty.washington.edu/jtenenbg/courses/360/f04/sessions/schneidermanGoldenRules.html`.

[23] Donald Norman. Summary of Don Norman's Design Principles. `http://www.csun.edu/science/courses/671/bibliography/preece.html`.

[24] Jakob Nielsen. `http://www.nngroup.com/articles/ten-usability-heuristics/`.

[25] WorlframMathWorld. Point-Line Distance–2-Dimensional. `http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html`.

[26] YouTube. Video Sharing Website. `www.youtube.com`.

[27] ChotiratAnn Ratanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michail Vlachos, and Gautam Das. Mining time series data. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 1049–1077. Springer US, 2010.

[28] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Comput. Surv.*, 45(1):12:1–12:34, December 2012.

[29] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In DavidB. Lomet, editor, *Foundations of Data Organization and Algorithms*, volume 730 of *Lecture Notes in Computer Science*, pages 69–84. Springer Berlin Heidelberg, 1993.

[30] Tetsuya Nakamura, Keishi Taki, Hiroki Nomiya, Kazuhiro Seki, and Kuniaki Uehara. A shape-based similarity measure for time series data with ensemble learning. *Pattern Analysis and Applications*, 16(4):535–548, 2013.

[31] Eamonn Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 406–417. VLDB Endowment, 2002.

[32] Jessica Lin and Yuan Li. Finding structural similarity in time series data using bag-of-patterns representation. In Marianne Winslett, editor, *Scientific and Statistical Database Management*, volume 5566 of *Lecture Notes in Computer Science*, pages 461–477. Springer Berlin Heidelberg, 2009.

[33] W. Bruce Croft, Donald Metzler, and Trevor Strohman. *Search engines : information retrieval in practice / W. Bruce Croft, Donald Metzler, Trevor Strohman.* Boston, Mass. ; London : Pearson Education, [2010], 2010., 2010.

[34] Victor Lavrenko. Vector space model of ir. *Text Technologies Course Slides*, 2013/2014. `http://www.inf.ed.ac.uk/teaching/courses/tts/pdf/vspace-2x2.pdf`.

[35] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition, 2011.

[36] Chris North. Keynote on Usability & Evaluation. In *Vizbi Conference*, 2010. `http://vizbi.org/2010/Presentations/Chris_North.ppt`.

[37] Donald A. Norman. *The Design of Everyday Things*. Basic Books, reprint paperback edition, 2002.

[38] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press USA, second edition, 2001.

[39] Jeff Atwood. Why Does Software Spoil? `http://blog.codinghorror.com/why-does-software-spoil/`.