

**Improving and Expanding the
Interface of an Existing MATLAB
Extension for Modelling
Biological Processes**

Luke Paul Buttigieg

Master of Science
School of Informatics
University of Edinburgh
2017

Abstract

This Masters thesis will present the approach taken when exploring ways of improving and expanding the interface of an existing MATLAB extension for use with Continuous Pi Calculus. The existing number of files visible when accessing the extension with a command line was reduced, and techniques from User Centred Design were applied when designing and implementing a Graphical User Interface for this MATLAB Extension, including a Cognitive Walk-through, a Heuristic Evaluation, Interviews and the Think-Aloud Protocol. We involved stakeholders and prospective users in all stages of the design and implementation of this project. Most of the functionality of the MATLAB Extension is now accessible from a Graphical User Interface, and User Support Documentation is now available.

Acknowledgements

I am grateful for the constant and unwavering support of my supervisor, Dr. Ian Stark, who guided me as I worked through this project.

I am thankful for the participation of the Systems Biologists and Bioinformaticians who took time out of their busy days to test the user interfaces I designed.

I am indebted to the academics at the University of Edinburgh, particularly those teaching Human Computer Interaction and Bioinformatics, who bestowed upon me the knowledge I needed to complete this project.

Last but not least, I would like to thank my parents, partner and friends back home and here in Edinburgh, for being there when things got challenging.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Luke Paul Buttigieg)

Contents

1	Introduction	11
1.1	Previous Work	11
1.2	Work Undertaken	11
1.3	Report Structure	12
2	Background	13
2.1	Continuous Pi-Calculus	13
2.1.1	Example	13
2.2	Continuous Pi-Calculus Workbench	14
2.3	MATLAB	15
2.4	CPi MATLAB Extension	15
2.4.1	Inspiration	15
2.4.2	Functionality	15
2.5	Human-Computer Interaction in Bioinformatics Software	16
2.5.1	Poorly Designed Software	16
2.5.2	Motivation for Improvement	17
2.5.3	User Centred Design in Bioinformatics	18
2.5.4	System Documentation	20
2.6	User Interface Design Principles and Methods	21
2.6.1	Interface Metaphor	21
2.6.2	Affordance	21
2.6.3	Gestalt Principles	22
2.6.4	Gutenberg Diagram	22
2.6.5	Contextual Help Information	22
2.6.6	Usability Inspections	24
2.6.7	Interview	25
2.6.8	Think-Aloud Protocol	25
2.6.9	Personas	25
3	Planning and Methodology	27
3.1	System Constraints	27
3.1.1	Command Line Interface Constraints	27
3.1.2	Graphical User Interface Constraints	27
3.2	Work Plan	28

4	Analysis of Existing Tools	29
4.1	Intrinsic Noise Analyser	29
4.1.1	General Observations	29
4.1.2	Design Observations	30
4.2	Kappa Calculus	31
4.2.1	KaDE	31
4.2.2	KaSim	32
4.3	Visual SPiM	33
4.3.1	General Observations	33
4.3.2	Design Observations	34
4.4	IQM Tools	35
4.4.1	General Observations	35
4.4.2	Design Observations	36
4.5	MATLAB Simbio	36
4.5.1	General Observations	36
4.5.2	Design Observations	37
4.6	CPi-IDE	37
4.6.1	General Observations	37
4.6.2	Design Observations	38
4.7	Summary of Design Observations	39
5	Code Re-factoring and Command Line Interface Redesign	41
5.1	Inline documentation and the Private Folder	41
5.2	CLI Modification	42
6	Graphical User Interface Design and User Evaluation	43
6.1	Design	43
6.1.1	Design Choices	43
6.1.2	Usability Inspections	44
6.1.3	Stakeholder Influence	46
6.2	Pre-Implementation User Testing	46
6.2.1	Test Procedure	46
6.2.2	Results	47
6.2.3	Test Improvements	49
6.3	Personas	49
6.3.1	Emily MacDonald	49
6.3.2	Ross McIntosh	50
7	Graphical User Interface Implementation and User Evaluation	51
7.1	Implementation	51
7.1.1	Implementation Choices	52
7.1.2	Differences between Mock-Up Designs and Implementation	55
7.2	Interviews	56
7.2.1	Question Structure	56
7.2.2	Answer Analysis	56
7.3	Post-Implementation User Testing	57
7.3.1	Test Procedure	58

7.3.2	Results	58
7.3.3	Test Improvements	59
8	Project Evaluation	61
8.1	Background Research	61
8.2	Re-factored Source Code and CLI Changes	62
8.3	Design and Implementation of a GUI	62
8.4	Pre- and Post-Implementation User Testing	63
8.5	System Documentation and Walk-through	63
9	Conclusion	65
9.1	Achievements	65
9.2	Known Issues	65
9.3	Future Work	66
9.3.1	Additional Functionality	66
9.3.2	Interface Improvements	66
A	Code Dependency Report	69
B	Final Version of the Design Mock-Ups	73
C	Sample Consent Forms	81
D	Graphical User Interface Screen-shots	85
E	User Testing Plans	89
F	GUI Support Documentation	95
	Bibliography	111

Chapter 1

Introduction

Throughout this project, we have explored and tried different approaches of improving and expanding the user interface of an existing MATLAB [1] extension which facilitates the use of Continuous Pi-Calculus (henceforth CPi) [2], predominantly by creating a graphical user interface (henceforth GUI).

1.1 Previous Work

Researchers frequently build models of biological or biochemical systems to understand them, often in terms of first-order ordinary differential equations (henceforth ODE(s)). CPi is a recent biochemical process algebra which provides a framework for expressing reactions in terms of ODEs. MATLAB [1] is a widely used platform for scientific and numerical modelling. The CPi MATLAB extension (henceforth CPiME) [3] was created to enable CPi processing through MATLAB, by interfacing with a Haskell library [2] and command line tool known as CPi Workbench (henceforth CPiWB) [4].

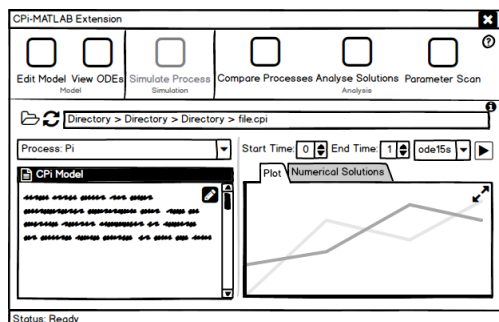
1.2 Work Undertaken

CPiME could previously only be used through a Command Line Interface (henceforth CLI), and the aim of this project was to improve this and expand on it, primarily by providing a GUI but also by improving the existing CLI.

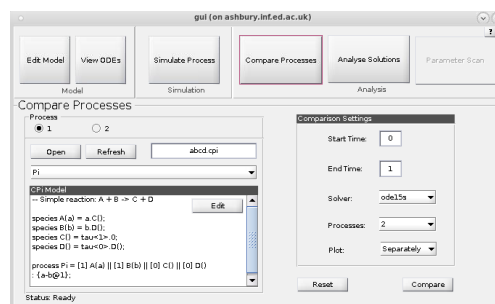
We started by conducting a thorough literature survey and analysing other tools to familiarise ourselves with this class of scientific software. We tried different User Centred Design (henceforth UCD) techniques which allowed stakeholders to be well involved in the design and implementation process including Cognitive Walk-through [5], Heuristic Evaluation [6], Think-Aloud Protocol [7], Interview [8] and Personas [9]. We believe that we created a GUI which implements many of the best practices, principles and methods we found, such as Interface Metaphors [10] and Affordances [11], Contextual Help Information [12], Gutenberg Diagram [13] and the Gestalt Principles [14]. We improved on this design by running pre- and post-implementation user tests. We iterated through four different designs, as seen in Figure 1.1a and two different implementation versions, as seen in Figure 1.1b. By creating a GUI, we might have

provided inexperienced CPiME users with a more straightforward way to experiment with the functionality available.

We have created two Personas which can guide future CPiME team members when developing future versions. We provided features which appeal more to non-technical users, such as a GUI, Support Documentation and Contextual Help Information and also technical users by providing richer in-line comments, and reducing the number of visible function files from twenty-eight to nine.



(a) A mock-up design of the "Simulate Process" screen, which is used to create graph plots and display numerical solutions of a CPi Model.



(b) The implemented "Compare Processes" screen, which is used to compare up to four CPi Model Processes.

Figure 1.1: We designed different versions of screen mock-ups and evaluated them by involving CPiME team members and prospective users. We then implemented the designs, and involved prospective users in testing them.

1.3 Report Structure

This report will first cover background knowledge necessary to understand the work performed during this project in Chapter 2. A literature survey of Human-Computer Interaction (henceforth HCI) in the Bioinformatics domain will also form part of this background chapter. The work-plan is then presented, with the project constraints in Chapter 3. Next, Chapter 4 will cover an analysis of other software tools and packages. Subsequently, we will be describing the work performed when improving the CLI in Chapter 5. We describe how we designed the GUI in Chapter 6 and implemented it in Chapter 7. We include an analysis of the answers received when we interviewed System Biologists and Bioinformaticians to understand their behaviour and expectations to software they use in the latter chapter. Finally, we evaluate the entire project in Chapter 8. We conclude in Chapter 9, but summarising our achievements, known issues and suggestions for future work.

Chapter 2

Background

This chapter covers the background knowledge necessary to understand the work done in this project. It describes the separate technology components which were used to build CPiME. It also includes a literature survey of HCI studies and the application of UCD in the Bioinformatics domain, which we reviewed to enhance our understanding of the field. We used this literature as a starting point when planning how to approach the task of making the user interface (henceforth UI) easier to use.

2.1 Continuous Pi-Calculus

CPi [2] is a biochemical process algebra that provides a framework for expressing reactions in terms of algebraic definitions.

There are two components of an algebraic definition — chemical *species* that represent the individual molecules, and *processes* that represent the overall systems, by forming descriptions in terms of parallel compositions of the species.

2.1.1 Example

To illustrate the language syntax, we can consider a simple example which involves four species: A, B, C, and D. A reacts to form C, while B reacts to form D. C decays at rate unity until it holds no concentration, otherwise known as an inert species. In this case, the reaction sites for A and B hold a mutual natural attraction at rate unity.

This example can be represented in CPi as follows:

```
species A(a) = a.C();  
species B(b) = b.D();  
species C() = tau<1>.0;  
species D() = tau<0>.D();  
  
process Pi = [1.0]A(a) || [1.0]B(b) || [0.0]C() || [0.0] D() : a-b@1);.
```

In this example, site *a* is where A reacts to form C, while *b* is where B reacts to form D. We can model spontaneous or silent actions, such as degradation, using *tau* in CPi.

C decays into an inert species, denoted by 0. On the other hand, D is static, and we can represent this silent action with rate 0.

All the changes of the four species of this systems example are described in Process Pi. We can use the `||` operator to denote parallel composition in the process definition. A non-negative, real number value is associated with them, representing the concentration prior to any temporal changes, is associated with the species in the process definition. There is also a colon (`:`), inserted after this parallel composition definition, which indicates the start of a global affinity network for the process.

Briefly, an affinity network is a finite, weighted graph which represents the attraction between any two reaction sites of the system. The nodes represent the reaction sites; i.e. where the reaction actually takes place. The weight of the edge represents the natural attraction between two reaction sites — an absence of an edge denotes no natural attraction. When using CPi, global networks must be defined for each process. These can be empty or else contain any number of attractions, separated by commas. The scope of attractions defined in these networks is global, and therefore visible to all other parts of the system. This is contrary to local networks which can be defined for each species in a model; these are only visible in the species they are defined in.

In the example process Pi, the reaction sites a and b have a "unity" attraction set. Species A and B are initially given a "unity" concentration. Species C and D have no concentration in the initial stage, since they arise from the reaction of A and B respectively.

The high-level description of a chemical species and their individual behaviours can be compiled into a set of ODEs. These ODEs represent what happens when they are combined, including all possible interactions and any new chemical species that emerge as the system reacts.

2.2 Continuous Pi-Calculus Workbench

Banks *et al.* created CPiWB [4] as a CLI tool for CPi. Written in Haskell [15], it can validate sets of CPi definitions, and also construct first-order ODEs for a specified process. CPiWB can also solve CPi ODE systems, and simulate them. Either temporal simulations of the concentration can be performed, or phase plots in two dimensions can be created for any two species in a process.

A screen-shot of the CLI of CPiWB can be found in Figure 2.1, which shows a simple CPi Model loaded into the system.

```

Welcome to the Continuous Pi-calculus Workbench (CPiWB).
Type "help" for help.

CPiWB:> load models/abcd.cpi
Loading: models/abcd.cpi
Done. Type "env" to view.
CPiWB:> env
A(a) = a.C()
B(b) = b.D()
C() = tau<1.0>.0
D() = tau<0.0>.D()
Pi = [1.0] A(a) || [1.0] B(b) || [0.0] C() || [0.0] D()

CPiWB:> █

```

Figure 2.1: The CPiWB Command Line Interface.

2.3 MATLAB

MATLAB [1] stands for MATrix LABoratory and is a widely used programming language developed by MathWorks Inc. It targets scientists and engineers and offers expansive Mathematical functionality built in. It includes a variety of ODE solvers [16] and solution simulation functionality [17], making it especially appropriate for CPi analysis. Several packages have previously been identified by Rhodes [3] which allow biochemical modelling inside MATLAB. Examples of these include MATLAB SimBiology [18] and IQM tools [19]. These packages served as inspiration for CPiME's design and implementation.

As part of this project, we followed three online MATLAB courses for us to fully grasp the MATLAB syntax and use best practices when developing with it [20]. The courses were called "MATLAB Onramp Preview", "MATLAB Fundamentals" and "MATLAB Programming Techniques", and they were valuable sources of coding help.

2.4 CPi MATLAB Extension

Rhodes stated that CPiWB's and CPi-IDE's shortcomings influenced the design decisions of CPiME [3]. CPi-IDE was one of the previous attempts at creating a GUI for use with CPi, and we analysed the tool in detail in Chapter 4.

2.4.1 Inspiration

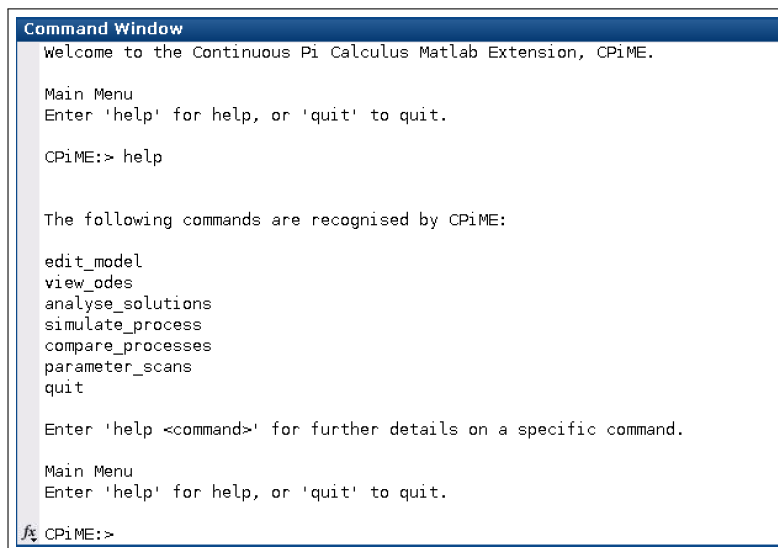
Experimenting with parameter values in CPiWB is said to be cumbersome as the user needs to reload the file manually after each change of the parameter values. The tool does not offer any text editing functionality, and is also said to be missing any form of comparator between CPi processes. That said, CPiWB was still integrated into CPiME using foreign calls through C code [21], and it is used to generate the ODEs from the CPi model.

Rhodes cites CPi-IDE's lack of scalability as a significant limitation which he attempted to overcome when creating CPiME. CPi-IDE reportedly becomes unresponsive and crashes when attempting to use models which have a large number of species and affinities. Therefore, CPiME does not contain any code of CPi-IDE at all, but inspiration was drawn from it.

2.4.2 Functionality

CPiME offers six different functions, as seen in Figure 2.2. It allows a user to: edit a CPi Model definition text file, generate and view ODEs, simulate a process and plot a graph of its temporal differences, compare two processes, analyse solutions by allowing logical statements to be checked against the solutions, and parameter scanning. Generally, most functions require a user to select one or more CPi Model definition text files upon launch, which have a `.cpi` file extension, and contain the CPi representation of the model. Most functions then call CPiWB to return the first-order ODEs for the chosen model and process. The ODEs are solved using one of a number of available MATLAB ODE solvers, and the different functions use this solved system to perform

their respective functionality. Some functions of CPiME already used a GUI; for example, selecting a process involves a list dialog box, and selecting a file uses a file sector dialog box. However, the GUI was not unified and only appeared in some parts of the application.



```

Command Window
Welcome to the Continuous Pi Calculus Matlab Extension, CPiME.

Main Menu
Enter 'help' for help, or 'quit' to quit.

CPiME:> help

The following commands are recognised by CPiME:

edit_model
view_odes
analyse_solutions
simulate_process
compare_processes
parameter_scans
quit

Enter 'help <command>' for further details on a specific command.

Main Menu
Enter 'help' for help, or 'quit' to quit.

fx CPiME:>

```

Figure 2.2: The CPiME CLI screen, showing the available options after the help command was run.

2.5 Human-Computer Interaction in Bioinformatics Software

As the field of Bioinformatics grows and matures, so have the approaches and tools used to extract, analyse and disseminate biological data [22]. We reviewed literature related to this area and summarised it into sections of knowledge relevant to this project. The aim is to understand the current state of HCI in Bioinformatics, and to identify suitable courses of action for this project.

2.5.1 Poorly Designed Software

According to Macaulay *et al.*[23], Bioinformatics tools suffer from many of the usability difficulties other scientific software is said to have.

2.5.1.1 Origins of the Problem

Macaulay *et al.*[23] said that funding for scientific software development is nearly always limited, usually being part of a larger research grant bid. Software Developers in this domain usually need to have scientific skills to understand the end-user requirements. In fact, it is necessary for scientists to be involved for a Software Developer to truly understand their requirements, but unfortunately it is said that this involvement only happens occasionally. Academic software developers tend to focus mostly on the

local setting of the laboratory for which the software is written. Additionally, funding bodies and cultural expectations are said to create barriers to creative design because most require open source software. Managing the expectations of software users is also problematic, especially when software developers are operating with the limited resources commonplace in scientific software development.

2.5.1.2 Anti-Agile Development

In more recent work, Morrison *et al.*[24] described many of these usability issues and noted that scientists' lack of understanding of computational methods are impacting the outcome of their research objectives. It seems that while scientists have managed to increase the volume of data being created after an experiment, the strategies and technologies of processing and interpreting the data have not caught up with the increase. Paradoxically, scientists are actually relying more on this software to interpret the data efficiently. This is said to be one of the leading bottlenecks in genomic discoveries, with software hindering rather than aiding the scientist's research efforts. The sheer size of the data returned is also making collaboration more difficult, especially because these Bioinformatics tools are not often designed to be collaborative tools. This limitation was also noted in previous work by Barker *et al.*[25], who claimed that efficient data integration was one of the major challenges facing Bioinformatics and systems biology software.

2.5.1.3 Limited and Sporadic Record Keeping

Scientists are reportedly [26][27] not keeping record of the data analysis procedure and how they interact with bioinformatics tools in the same way that they record wet-lab experiments. As a result, the way tools are used in analysis work flows, such as the parameter settings used, are often lost. When scientists keep notes, they are often in the form of "cheat sheets" which demonstrate their limited understanding of how the tools work. They also do not keep track of the output of the system, relying on digital output instead. Morrison *et al.*[24] also observed that scientists rely on default parameter values when using Bioinformatics tools, which are at times unsuitable for the data being analysed, because they do not fully understand the algorithm powering the tool. Command Line interface tools are often said to be difficult to understand and use, and frequently publishers leave out useful system documentation. Ultimately, the combination of these issues mean that scientists rely on computational experts to perform their work, and are concerned when they are unable to access this expertise. To conclude, gaps exist in the study of the factors which make it difficult for scientists to use Bioinformatics software. Therefore additional, undocumented issues could also be having a usability impact.

2.5.2 Motivation for Improvement

Javafery *et al.*[28] claimed that initial tools were not user friendly nor well designed. Hence, given the novelty of the tools, developers made little effort to make the tools more approachable and easy to learn. As more scientists started using the tools, Javafery *et al.*[28] said that the need for Bioinformatics systems which are easier to use

for new users and more capable for experts became evident. Bioinformatics tools are said to be especially challenging because users try to perform various complex tasks with them. Medical researchers and practitioners are expected to make broader use of better designed Bioinformatics applications. Training costs are also said to have decreased by improving usability. Migrating users to newer versions of software is also claimed to be easier when a tool is better designed. Thus, they recommend that software developers analyse and understand the users' experience to successfully design and release Bioinformatics tools.

2.5.2.1 Impact on Research Quality

Additionally, Al-Ageel *et al.*[29] state that people's perception and experience when interacting with Bioinformatics tools can have a great impact on the understanding of complex data being analysed. They claim that in spite of advances in technology, ultimately it is humans who perform most of the data mining by joining the different parts of an analysis together, and hence make decisions in an analytic Bioinformatics study. Realistically many users of Bioinformatics or systems biology tools are said not to have a computational background, which motivates a greater investment in usability engineering. This echoes the earlier assertions by Bolchini *et al.*[30] and Mirel *et al.*[31] that the software tools scientists use can vastly influence the direction and success of their research.

2.5.2.2 Commercial Advantages

Separately, Mayhew *et al.*[32] claimed that this is an area which companies are targeting for competitive advantage, and first noted that better usability allows users to adapt to new versions of existing systems quicker, spend less time in training, achieve better quality of work and decrease the risk of data input errors. He states that improving usability allows users to take advantage of the best balance of compromise and trade-off by providing a simple UI for powerful functionality. Other work by Senadheera *et al.*[33] graded existing Bioinformatics tools according to their perceived "user-friendliness" and suitability for inclusion in open and distance learning programs of Bioinformatics, which again confirms the commercial value of improving the tools' usability.

2.5.3 User Centred Design in Bioinformatics

HCI and UCD deserves greater attention according to Pavelin *et al.*[34], who claimed that a cultural shift is required in the field of Bioinformatics for stakeholders to embrace UCD approaches and make a greater effort in designing systems users actually want to use. They claim that although the Bioinformatics community is managing to supply accurate data which is valuable to users, the interfaces to access the data are seldom well designed and often not straightforward to use.

2.5.3.1 Overview and First Sample Approach

UCD is described as a design philosophy which focuses on the user throughout the development process of the software application, including when designing, testing

and implementing the application [34]. The goal of this philosophy is to produce a usable application tailored to the specific needs of a set of users. By including users in the design and development process, software developers can better understand their user's needs and how they will interact with the software once it is created. This is said to be especially relevant in the case of Bioinformatics software, where the end users are often formally trained in Biology and have acquired complimentary, rather than dedicated, computer skills.

Figure 2.3 shows an overview from Pavelin *et al.*[34] of the UCD process. This strongly influenced the project plan, although time constraints limited the scope for multiple iterations.

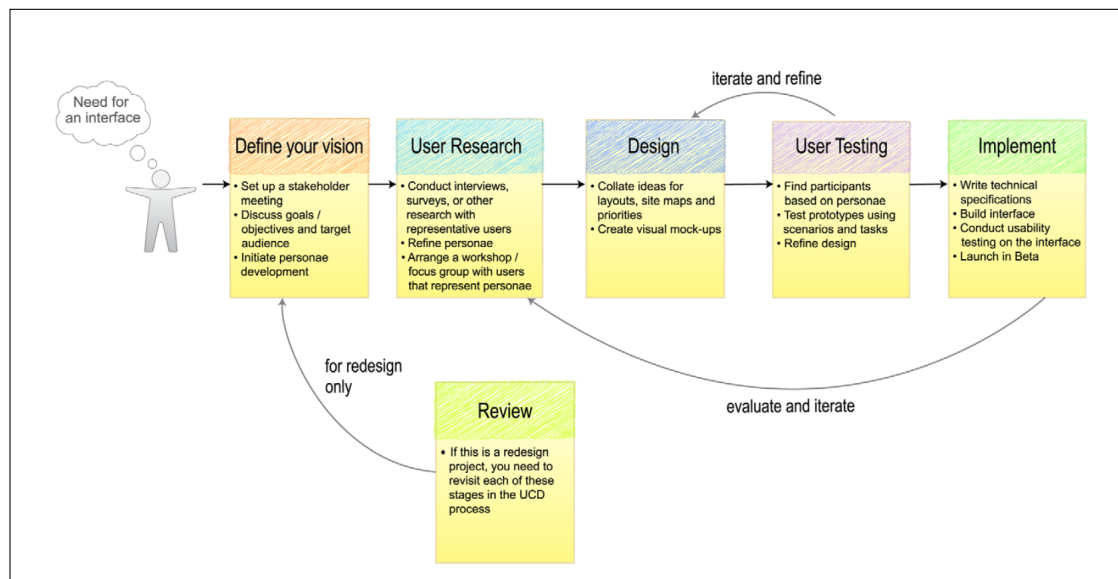


Figure 2.3: A suggested overview of the user-centred design process [34]. The needs of the user are said to take precedence, ensuring that the final implementation is acceptable for the user and fulfils their needs. This process is said to be iterative. It is interesting to note that user testing for the interface design is recommended to take place prior to the implementation.

Pavelin *et al.*[34] state that although it might seem that the entire development process becomes longer since the design process involves far more work in a UCD approach, this is offset by software developers ultimately spend less time implementing the system. In fact, Mirel *et al.*[35], who focused on applying a UCD technique called Heuristic Evaluation [6] to Bioinformatics tools, stated that this was ultimately a low cost method for identifying issues with a UI design, and far less expensive than dealing with issues once the software development is complete.

2.5.3.2 Second Sample Approach

More recently, de Matos *et al.*[36] confirmed that involving domain experts in the design process leads to an effective approach to UCD. They listed HCI and UCD best practices which they uncovered in their study, some of which have the potential of universal application; for example, they found that software developers should time-stamp data

they are exporting from a system, as this would make it easier for a researcher to recall the context of this data at a later date. They also noted that users want an easy way of forwarding information to their peers.

They also recommend the use of consent forms prior to carrying out the usability study, and notifying the participants of the right to withdraw from the study at any time. They note that multiple activities can be part of a UCD approach, depending on the requirements of the project and the availability of information.

Their UCD approach involved:

1. Holding a stakeholder meeting during which they gathered and formalised the scope and requirements of the project.
2. Creating personae which represent a major user group. These descriptions guided them when taking decisions about product features, navigation and interactions.
3. Interviewing users to confirm that the personae align to them.
4. Creating a work flow map which condenses the information gathered in the interview and personae describing.
5. Organising workshops with domain experts, to confirm the most important elements of the design and information architecture.
6. Creating paper prototypes to identify issues with the design.
7. Creating a technical specification document defining what needs to be developed.
8. Refining the prototypes after testing the usability using an interactive prototype.

2.5.4 System Documentation

When designing a Bioinformatics tool using user centred design principles, the developer must tailor the application for the task being carried out.

Barlett *et al.*[37] listed a number of best practices in their work, which are suitable for Bioinformatics analysis tools. They state that an overall lack of cohesive information describing the use of tools in conjunction with others is a common source of frustration for scientists. Their first suggestion is to create better system documentation of the type of analysis information the tools can provide, details of the information resources create and how different resources fit into the whole picture.

Secondly, they claim that it is advantageous to list both the reasons behind performing the Bioinformatics analysis, and separately, how to perform it in the system documentation. They recommend layering the systems documentation, presenting increasingly specific and detailed information as a user goes deeper into an area of the documentation. The very outer layer of documentation should include an overall view of all the functionality available and its use in context. They also recommend using a breadcrumb trail in the application, which allows a user to easily identify the stage they are in and also to backtrack with ease. In the end, they admit that their recommended design choices were not proven to improve the efficiency of Bioinformatics analysis and that an additional experiment would have to be developed to confirm their hypothesis.

2.6 User Interface Design Principles and Methods

We will now introduce the relevant UI design principles practices which we used, or referred to, in this project.

2.6.1 Interface Metaphor

An interface metaphor is used to make complex computer notions easier to understand by giving them a real world label, even though this might not be entirely correct, realistic or fully representative of the underlying logical concept [38]. Interface metaphors are popular and have now become ubiquitous in a modern operating system (henceforth OS) and applications. An example of an interface metaphor would be the "computer desktop" with "windows", as in Figure 2.4. There is no "desktop" in a computer screen, but the blank area which usually contains clickable icons (usually referred to with another metaphor as "files") has been named as such as it represents the active work-space of a computer user, just like the user's active work-space in the physical world. Other popular interface metaphors include a "folder" used to organise "files", a "trash can" used to delete "files", a floppy disk icon used to save "files" and "tabs" used to organise UIs, much like file dividers in the physical world.

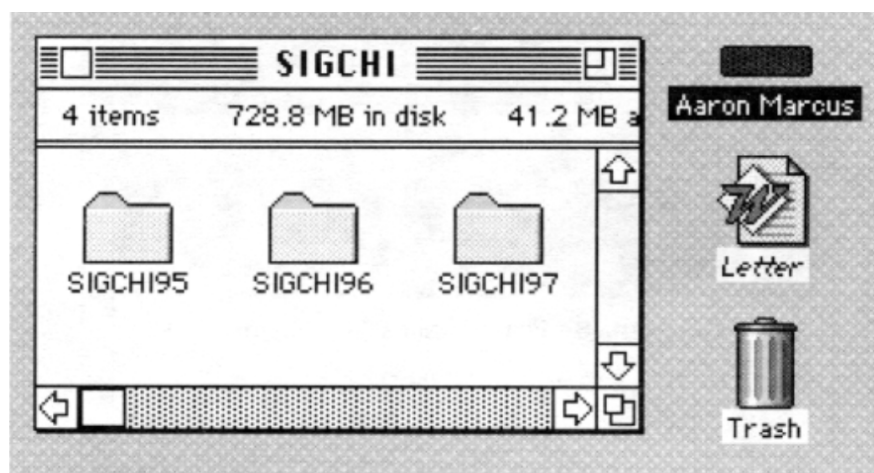


Figure 2.4: An early example of the "desktop" interface metaphor with "windows", showing "files" and "folders", and a "trash can" [10].

2.6.2 Affordance

An object's design can suggest how it is to be used, thus clearly and easily indicating how to use it. For example, a flat panel on a door indicates that the door is to be pushed, while a handle indicates that the door needs to be pulled instead. This is known as an affordance [11]. Since GUIs are not made out of physical objects, affordances in UI design are actually only perceived affordances, arising when a user is familiar with the represented object in the real world. For example, a user knows that a button in a GUI is to be clicked because buttons in the real world are pressed. GUI affordances leverage user's real world knowledge to indicate how they should interact with them.

2.6.3 Gestalt Principles

The Gestalt Principles [14] are a set of principles which explain how human perception works, based on the earlier work of Gestalt psychologists [39]. Part of their observation concerned how humans perceive objects in a grouped, organised way and in a pattern, even if they are not deliberately placed like that. These observations can be exploited when designing a UI, as following the Gestalt Principles allows properties of interface elements to be conveyed in-explicitly. For example, one of the principles is that of proximity: that objects close to each other seem to create groups. UI designers can use this principle by placing interface elements, such as buttons belonging to the same theme, close together, while leaving space between unrelated groups. Figure 2.5 shows a diagram of the different Gestalt principles of perception.

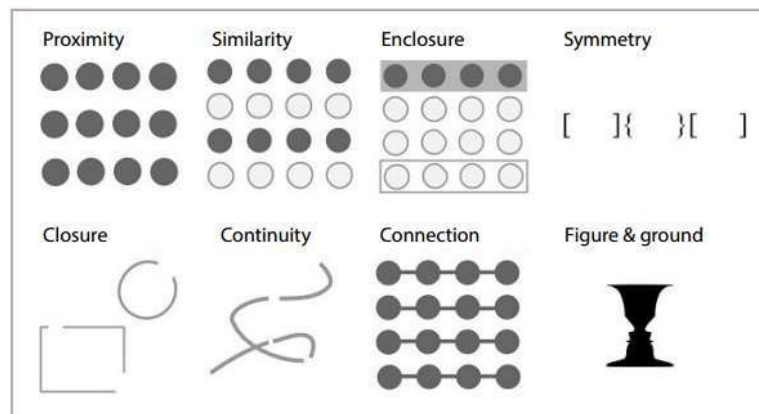


Figure 2.5: An illustration depicting the Gestalt Principles [14] [39] taken from a blog entry by Taylor [40]. The eight illustrations depict different ways the brain can be influenced to believe that objects belong in the same group.

2.6.4 Gutenberg Diagram

The Gutenberg Diagram demonstrates the direction Western users look at information which is presented in a fairly distributed and even manner [13]. Users typically immediately focus on the top left quarter of a screen, known as the *primary optical area*, before moving down and across to the bottom right of the screen, known as *terminal area*. The top right and bottom left of the information, known as the *strong fallow area* and *weak fallow area* respectively, receive little attention unless they are visually emphasised. This principles generally does not apply when the elements are not homogeneous and evenly distributed — in that case, the weight and make up of the elements guide the eye. This principle can serve as inspiration when creating UIs, as following it is said to improve comprehension of the information presented. Figure 2.6 shows an illustration of the Gutenberg diagram.

2.6.5 Contextual Help Information

Extensive, comprehensive support documentation is useful and undoubtedly one of the key tools to make a UI easier to use. However, there are times when it would be

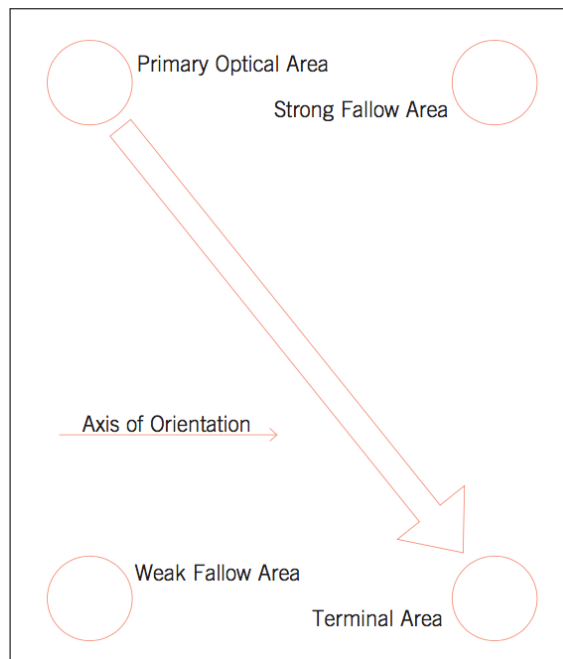


Figure 2.6: An illustration depicting the direction of a user's eyes when perceiving homogeneous material which is equally distributed [13].

valuable to display short excerpts of this documentation directly in the location of the UI element being used. This is known as contextual information [12] with one of the most popular forms existing as tool-tips, which appear when hovering a mouse pointer over a GUI element like in Figure 2.7, such as when a user is hesitating to interact with the GUI element. Contextual information offers a quick and effortless means of conveying additional information to a user when it seems to be needed, without requiring them to go through support documentation.



Figure 2.7: An example of contextual help information in a video game [41]. The information is presented within the UI itself.

2.6.6 Usability Inspections

There are different ways of performing usability inspections, but they are generally all cost-effective means of discovering problems with a UI [42]. They usually involve evaluators inspecting a UI, usually against a defined criteria, and attempting to discover issues without involving an end user.

2.6.6.1 Cognitive Walk-through

A cognitive walk-through is a task-specific usability inspection method which is especially useful when evaluating systems for which users do not receive prior training, and when users are expected to simply use the system without having consulted support documentation first [5]. Evaluators check whether each step of the interaction process bring a user closer or further away from their goal when using the system, by following a defined, systematic, analytic approach. They first create a set of tasks that they believe a user would want to carry out using the system they are testing. For example, if they are testing a parking ticket payment machine, a sample action would be "Pay the parking ticket". Afterwards, they decompose the action into steps which a user must go through to perform the action, such as "Insert the ticket into the ticket machine".

Usually, evaluators ask four standard, learning theory-based questions at each of the steps that they are analysing:

1. Do users actually want to carry out this specific action?
2. Is the control for the action, such as the button, visible?
3. Is it clear that the control for the action produces the effect the users want?
4. Does the system respond appropriately once a user triggers the control, allowing them to confidently proceed to the next step?

The answers to the questions are then used to determine whether the system should be redesigned or not.

2.6.6.2 Heuristic Evaluation

A Heuristic Evaluation is a cost-effective means of checking whether a UI adheres to an agreed-upon list of usability best practices [6]. Evaluators usually iterate through the list and then suggest improvements if they discover that the UI does not implement the best practice. This method is often used before implementing or involving users in usability testing. It usually results in a holistic approach of issue finding.

An often used list is known as Nielsen's Heuristics [43], which is cited below:

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use

8. Aesthetic and minimalist design
9. Help users recognise, diagnose, and recover from errors
10. Help and documentation

2.6.7 Interview

An interview is a research method which provides a rich understanding of a subject's opinion of a matter, providing the possibility of obtaining firsthand opinions, beliefs and attitudes [8]. They are often conducted in person, as body language can be included as part of the participant's opinion. Structured interviews follow a strict set of scripted questions, and unstructured interviews are more akin to a conversation and usually more comfortable for participants. In unstructured interviews, researchers usually have a number of subject areas they would like to investigate but not a defined number of questions. Interviews can also be semi-structured, borrowing elements from structured and semi-structured.

The appropriate selection of the target audience is an important element when conducting interviews — it is important that researchers involve users who qualify as stakeholders in their research. For example, if a researcher is interested in the behaviour of new users of a particular software application, it is important that the interviewees are indeed new users, and not seasoned users.

2.6.8 Think-Aloud Protocol

The Think-Aloud protocol is a simple, but effective, evaluative method which involves direct participation of a user [7]. When using the think-aloud protocol, the researcher sits by a user, and asks them to perform a number of tasks and to verbalise what is going through their mind as they are doing so. Two major forms exist: *concurrent* think-aloud, where a user speaks as they perform the task, or *retrospective* think-aloud, when a user describes their experience after they complete the task. Participants are usually recorded in the retrospective setup, to help them recall their thoughts afterwards.

Users are usually requested to completely speak their mind, including intermediate thoughts or at points of confusion, to ensure the researcher understands their intuition and identifies the sources of any issues. Researchers are not only interested in the participant's thoughts, but also their feelings and how they approach the assigned task.

2.6.9 Personas

Personas humanise the focus of the design effort and remind designers who the target audience is [9]. Personas usually consist of half page to a page-long descriptions of a fictitious person's life and goals, history, skills and sometimes also a photo, and there is usually one persona for each group of users targeted. For example, if an application is to be made usable by novice, intermediate and expert users, one would usually create a persona for each of those levels of expertise. Personas are usually created to cover a larger group of users with similar needs, rather than an outlier.

Personas are then typically used throughout all the project phases, making it easier for system developers to sympathise with users' needs.

Chapter 3

Planning and Methodology

This chapter briefly describes System Constraints which impact the design and development of this project. We will also be describing the work plan we followed in this project.

3.1 System Constraints

As part of the initial stages of the project, we listed the system constraints to ensure they are followed. They are organised by interface type. The CLI constraints impact users who are either directly running the functions in the files (bypassing the existing CLI), or else those using the existing CLI. The GUI constraints naturally only impact users accessing the functionality using the GUI.

This is by no means a complete, explicit list of requirements; rather, it is a collection of informal constraints from different stakeholders. Henceforth, *stakeholders* will refer to fellow CPiME Team Members and the Project Supervisor.

3.1.1 Command Line Interface Constraints

1. The CLI should retain the same functionality as it was published by Rhodes [3]. Any changes in the code should not be visible to the CLI user, but only to anyone inspecting the source code.
2. Upgrade the CLI functionality, if required.
3. Inline documentation should be extended in the files containing functions which can be directly called, and the extension's functionality should be made modular.

3.1.2 Graphical User Interface Constraints

1. The desktop application GUI should recreate the existing CLI functionality appropriately by adapting it to the new interface format. The design should avoid including a series of dialog boxes to enter parameters, or a wizard-style approach [28].
2. The GUI should adopt popular design cues and should be as simple as possible.

3. The GUI design should be consistent [44] across the different screens.
4. The .cpi file selected on one screen should remain selected even if a user selects another screen.
5. The GUI should offer appropriate documentation, in the form of contextual information [12] and standalone support documentation.

3.2 Work Plan

After reviewing the literature covering HCI and UCD practices in Bioinformatics and Systems Biology, we synthesised it to form a work plan. Our goal was to combine the novel techniques and approaches in the literature and while still going through the typical phases of the development cycle, such as the requirements collection, design, development and test phases [45]. The plan is similar to Pavelin *et al.*'s [34] in Figure 2.3, involving stakeholders and prospective end-users in every step of the design and implementation process. It is as follows:

1. Review alternative Systems Biology and Bioinformatics applications, perform a Heuristic Evaluation [6] on them and analyse whether they observe the Gestalt principles [14].
2. Review the existing CPiME [3] code and implement changes to fulfil the constraints defined in Section 3.1.1.
3. Design the mock-up designs of the GUI next, in line with Section 3.1.2. This part was planned to involve several iterations, as we designed different versions.
 - (a) Finish designing a version of the mock-up designs.
 - (b) Perform a Cognitive Walk-through [5] which should tackle issues affecting users who do not consult support documentation before using the application. Perform a Heuristic Evaluation [6].
 - (c) Involve different stakeholders in evaluating the mock-up designs by interviewing them [8]. Incorporate their feedback in the mock-up designs, and repeat the process if necessary.
4. Select the final set of mock-up designs and start iterating through the implementation process.
 - (a) Implement the first set of GUI screens.
 - (b) Perform Usability Testing using an Interview [8] and the Think-Aloud protocol [7]. Implement any changes which are discovered to be needed.
 - (c) Repeat the iterative process with the next set of screens.
5. Once the screens are implemented, implement other functionality in the GUI, such as support documentation and improving the look and feel. Possibly, experiment with other UCD techniques. This part of the plan was known to be time permitting.

Chapter 4

Analysis of Existing Tools

After consulting relevant literature and extracting what we feel were the best practices from it, we turned to existing Bioinformatics or Systems Biology tools for inspiration on what to do, and what to avoid. We analysed seven different tools which are likely to be used by the same user-base as CPiME to understand how they make complex functionality easy to use.

We first recorded the general observations when using the tools, and then performed a Usability Inspection by informally performing a Heuristic Evaluation using Nielsen's Heuristics [46], and also checked whether they generally observed the Gestalt principles of perception [39]. These two techniques were described in Chapter 2. We summarised our analysis of which of Nielsen's Heuristics [43] each application observed by creating Table 4.1, which can be found at the end of this chapter.

Since a full and thorough analysis of these software tools using these two techniques would have been beyond the scope of this project, and also unrealistic due to the time constraints, we used a quicker, less formal method and then summarised our findings below each application. We generally used the list of Nielsen's Heuristics and Gestalt principles of perception as guidelines and commented if we noticed whether they were significantly followed, or not. Understanding the technical functionality of these applications was also deemed outside the scope of this exercise — we merely sought to understand the approach taken when creating a UI for these tools, and to establish whether best practices seem to have been followed or not.

4.1 Intrinsic Noise Analyser

The Intrinsic Noise Analyser (henceforth INA) [47] is a software application which uses system size expansion to explore stochastic biochemical kinetics.

4.1.1 General Observations

INA provides a similar feature set to CPiME, allowing a user to create models and then simulate them. It is also possible to plot these simulations. Users can use a GUI to interact with the functionality. Unfortunately, we could not compile or run the software when we evaluated this application, and this issue was confirmed by one of the authors of this paper. Therefore, we evaluated the screen shots found in the paper itself, as

seen in Figure 4.1; there exists a possibility that the functionality which we presume is provided is not fully accurate.

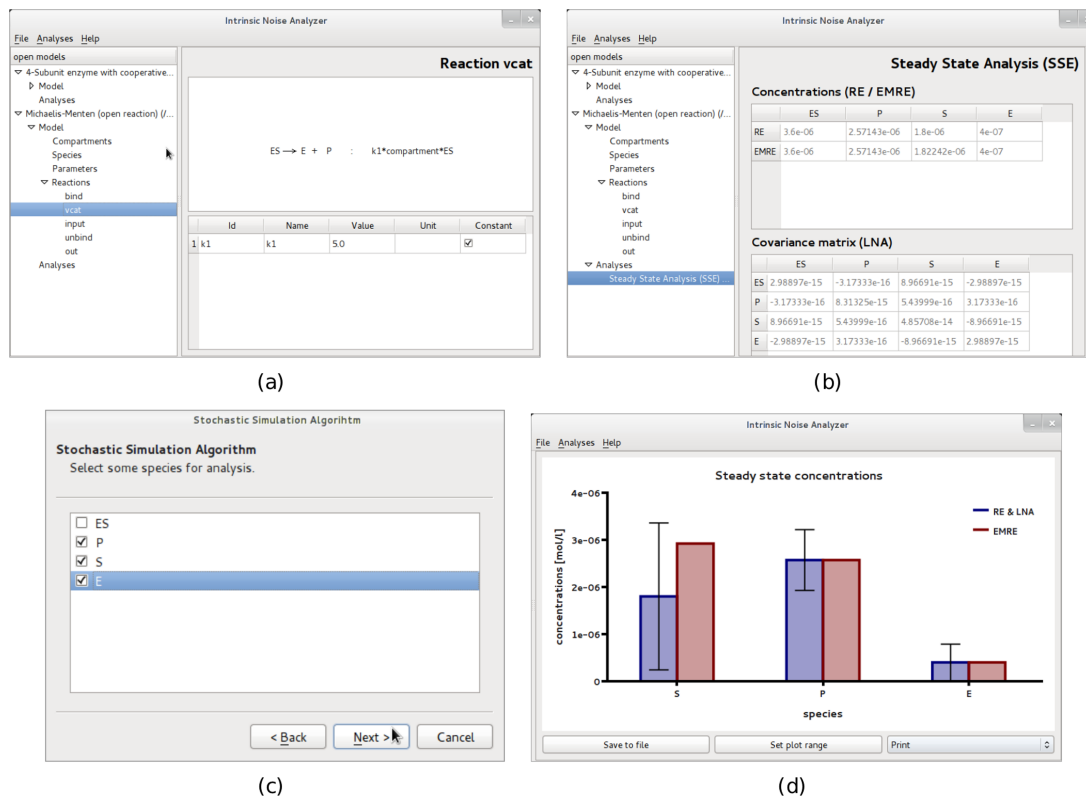


Figure 4.1: Four sample screens of the Intrinsic Noise Analyser application [47]. Figure A seems to depict the screen used to modify a reaction model. The screen on Figure B can seemingly be used to analyse the reaction and its properties. Figure C is a screen requesting feedback from a user, in the form of a wizard. Figure D depicts the plot of concentrations of the different species.

4.1.2 Design Observations

INA appears to adhere to most of Nielsen's heuristics, with a clean, consistent design that supports the user's actions. Support documentation appears to be available, and the terminology used seems to be familiar to a Systems Biology or Bioinformatics user. We could not verify whether it prevents errors effectively, and we were also unclear with how the system status is communicated to the user. We were also unsure whether the feedback given when an error arose was helpful or not. The GUI seems to respect the Gestalt principles, using distinct areas to organise the screen of elements into groups. Proximity is used to show that screen elements are related to each other; for example, the buttons in Figure 4.1c are close together as a user is expected to select only one button of the three at a time.

4.2 Kappa Calculus

Kappa Calculus [48] is a visual language which is used to define first-order functions. A popular application of the language is in modelling molecular biology, for example in modelling protein-protein interactions.

We analysed two different tools which use Kappa Calculus. The first was KaDE [49], which is a desktop application which compiles Kappa Calculus models into ODEs. The second was a more recent application called KaSim [50], a web-based application which allows models to be simulated and plotted.

4.2.1 KaDE

KaDE offers a GUI to compile Kappa Calculus models into ODEs with many customisable parameters, as seen in Figure 4.2.

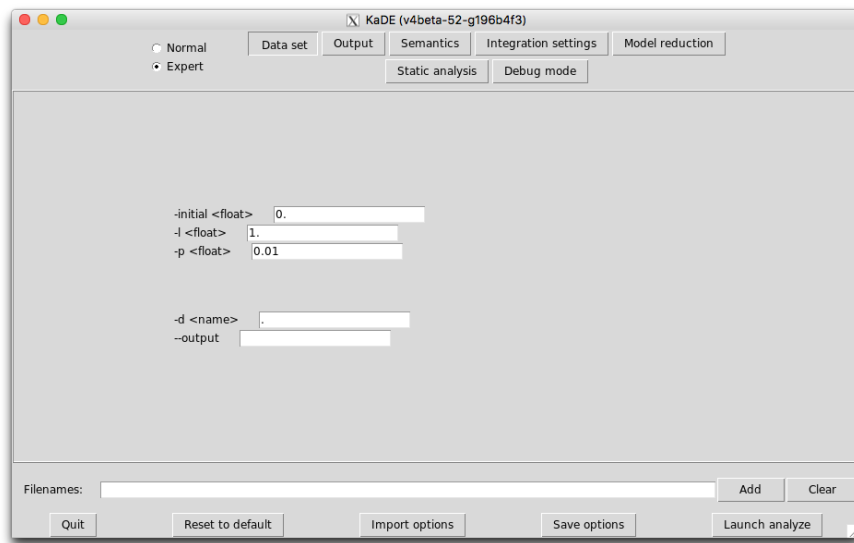


Figure 4.2: The landing page of KaDE [49] showing the data selection page.

4.2.1.1 General Observations

KaDE offers a fairly rudimentary GUI. We feel that parts of it are a little haphazard, possibly due to a lack of interest or resources in investing in a better design. This seems to be in line with the limitations of scientific software design mentioned in some of the literature surveyed in Chapter 2, such as Javafery *et al.*[28]. It offers "Normal" and "Expert" modes, offering more parameters in the latter setting. Unfortunately, it is not clear how or why some options are considered for "Expert" use only, and this could be a course of confusion for some users. There does not seem to be any extensive support documentation built-in, but contextual information is available when hovering over the buttons or the text boxes. Users can save the options inserted into the GUI onto a file on disk, or import them from it, which would be appreciated by the scientists who lament needing to remember all the parameters they need to insert [24].

4.2.1.2 Design Observations

The GUI is organised in different pages which are accessible using tab buttons. Radio buttons are also used for the "Normal" or "Expert" mode switching, another interface metaphor allowing only one of the two options to be selected at a time. Onscreen buttons pop out of the screen with their 3D design, which is a Affordance [11], helping users identify which buttons they can press.

The developers do not seem to have followed many of Nielsen's Heuristics; for example, it is not easy to know what the current system status is, nor does it seem like sufficient guidance is provided to help users prevent errors. Some heuristics seem adhered to; contextual information allows users to avoid errors, and "Reset to default" and "Clear" buttons provide a high degree of user control and freedom. The design is also fairly clean.

Furthermore, the design does not seem to abide by the Gestalt principles: parameter text fields are not in line with each other or close enough to suggest that they belong to the same group. As stated by Lidwell *et al.* [51], alignment can lead a user through a design and contribute to the overall aesthetic, so this could prove problematic.

4.2.2 KaSim

KaSim [50] allows users to create and edit models defined in the Kappa Language, and to simulate and plot graphs of them. Apart from a text editor with line numbers, the program also represents the process using a visual editor, which can then be exported as an image, as seen in Figure 4.3.

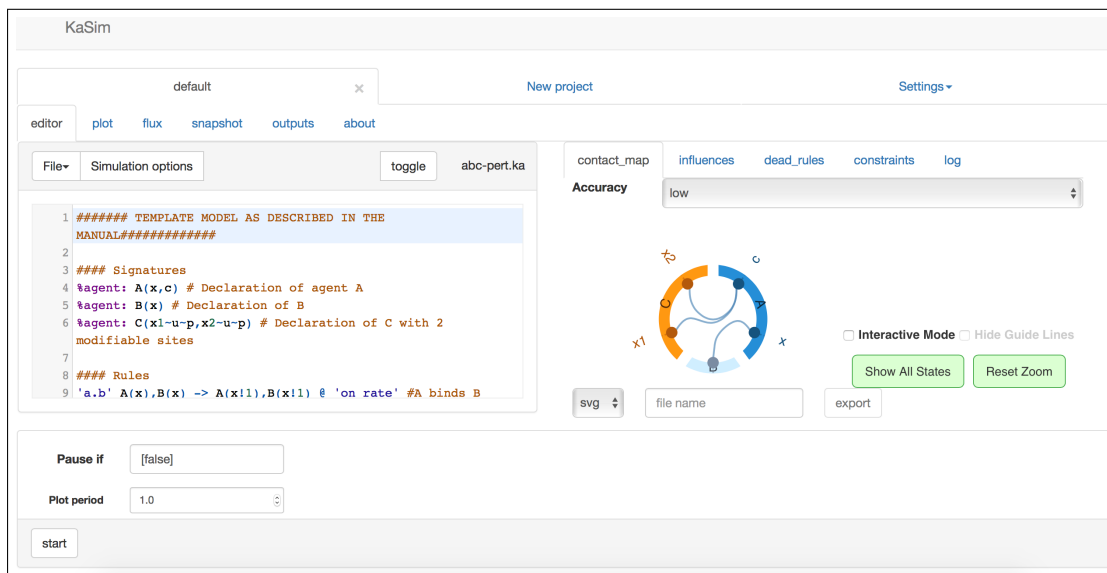


Figure 4.3: The landing page of KaSim [50] showing the Kappa Calculus model text editor on the left and the visual editor on the right.

4.2.2.1 General Observations

This web-based application seems to have been developed using fairly modern standards such as data visualisation library d3 [52].

The interface feels responsive, using JavaScript [53] to prevent reloading the page when new data needs to be presented on screen. JavaScript also animates elements on screen. When a simulation is taking place, no end time is required and the simulation continues indefinitely, updating the plotted graph and the numbers displayed on screen until the simulation is halted.

4.2.2.2 Design Observations

We believe that the developers of this web-application designed it with usability in mind, as many of Nielsen's Heuristics seem adhered to. As the process is simulated, a live log, which is in a dedicated tab, allows a user to know the status of the system. A user can stop and clear a partially simulated model, easily undoing any changes they might have made. The design is clean and fairly straightforward to use, and implements popular interface metaphors such as tabs.

We could not trigger any error messages while using the application, which could either mean that the application is robust enough to support the variety of parameters we inserted, or that the application actually doesn't help a user prevent and recover from errors. The application also wholly excludes support documentation in the application itself, with contextual help and embedded, longer guides not being found.

The observation of Gestalt principles makes it naturally easy to navigate the interface. Different colours of shading and areas enclosed by lines group onscreen elements together. These grouped elements have the same dimensions and shapes. Although no border line is present around the right panel on the landing page, the continuity principle affords a user to naturally associate the interface elements within it, together. Drop down menus use shadows to indicate that they are in focus and in the foreground, with the rest of the elements being in the background when they are open.

4.3 Visual SPiM

Stochastic Pi is a programming language used to design and then simulate computational models of biological processes. Visual Stochastic Pi Machine (henceforth SPiM) [54] is a web-based application created for use with the Stochastic Pi Language [55].

4.3.1 General Observations

Visual SPiM offers a GUI which is rich in functionality, responsive and fast to use. It was built using Microsoft's proprietary web browser plug-in called Silverlight [56]. Users can load or save Visual SPiM models defined in text files using the interface and also edit them using the text editor, as seen in Figure 4.4. It also allows users to zoom in and out of the text in the text editor. It has a number of examples built in, so users can quickly try out the application and the functionality. Models are first built, and then

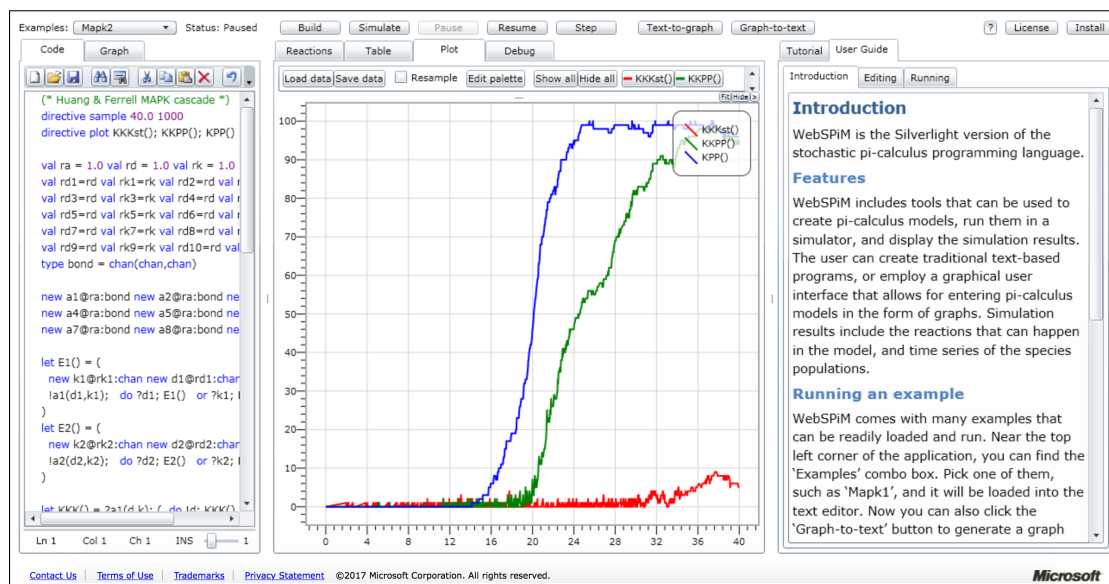


Figure 4.4: Visual SPiM [54] showing the model text editor on the left, a plot of the simulated model in the centre and the support documentation on the right.

simulated using respective buttons on the UI. Users can plot graphs of the chemical ODEs, and numerical solutions are presented in tabular format.

However, the richness in functionality comes at a cost, as defined in Hick's Law [57] which states that the time it takes to perform a task increases as the number of alternatives increases. We found that the interface looked busy, and while the tabbed interface metaphor is useful in structuring the onscreen elements, we discovered a deeply nested tabs structure which left the application feeling cumbersome to use. All buttons and tabs are active at all times, even when the functionality behind that element is not ready to be used. This meant that we were not always clear with what we should click next, and this negatively impacted the application's ease of use. We also found repeated use of the same terms across the UI, without an explanation of how one instance differs from another. For example, we found three "Graph" tabs across the entire interface, and there was no information regarding the distinction between them.

4.3.2 Design Observations

Visual SPiM adheres to the majority of Nielsen's Heuristics. A clear system status field can be found in the top left corner, and the UI uses design cues which are visually similar to typical Microsoft software [58], matching the system with that of the real world of the user. It supports undo and redo operations in the text editor, and the design language used is mostly consistent. User support documentation is available and highly detailed, and contextual help is also present in the form of tool tips on many buttons. Since the UI is generally static, users do not need to recall anything from memory, but rather can simply recognise that they are using the application appropriately. Plotted graphs can be panned by simply clicking and dragging them, a UI affordance which feels natural to use. Additionally, users can immediately know which buttons to click in the text editor as they have popular interface metaphors, such as a floppy disk icon

on the "Save File" button.

We observed an element of inconsistency in the UI, for example, some "Save File" buttons have a floppy disk icon on them, while others have a string description instead. There also seems to be another system status indicator in the lower right corner, which replicates that in the upper left corner.

Gestalt principles appear to be followed since related UI elements are in close proximity and use varying degrees of shading, for example in the text editor buttons layout. Shadows are used to show which elements are in the foreground, such as active tabs.

4.4 IQM Tools

IQM Tools [19] is a software package which plugs into MATLAB to expand its functionality. It is used mainly in systems pharmacology, pharmacometrics and systems biology, and is made up of a number of different, distinct libraries which all have independent and unrelated GUIs.

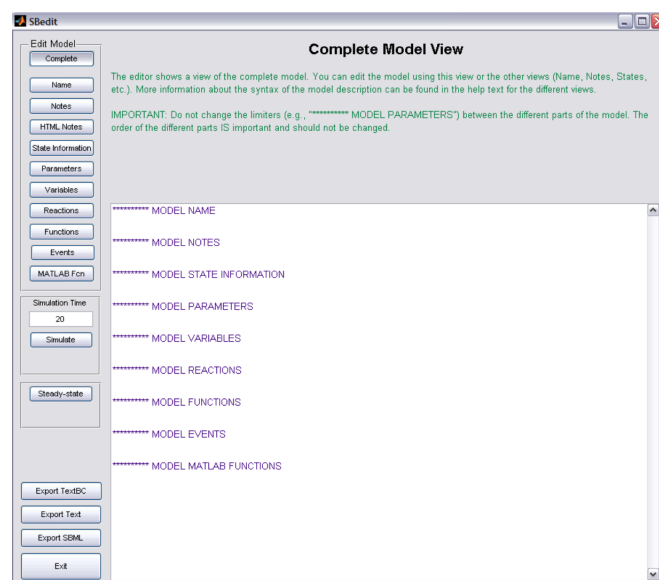


Figure 4.5: IQM Tools [19] has numerous different GUIs. This one is used to edit and simulate models.

4.4.1 General Observations

The installation process of IQM Tools involved a significant amount of configuration and preparation, which might discourage inexperienced users from installing it. Once installed, the different GUIs of the software package are loaded by inserting a command into the CLI in MATLAB. The GUI designs vary considerably, and depend on the use of the specific GUI. For this exercise, we are going to focus on the "SBedit" screen, which can be used to create and edit models, simulate them and export the simulations to a variety of formats. This screen can be seen in Figure 4.5.

4.4.2 Design Observations

Since there is no unified GUI to launch all the different components of IQM Tools, users need to recall the CLI command every time they choose to use it. This directly challenges the Nielsen's Heuristic that recognition is better than recall. In "SBedit", the system status is not clearly visible when a model is being simulated, and it is also not possible to stop a simulation once it starts, or to undo or redo any changes. Since the different GUIs are significantly different, we feel that this package is not consistent and non-homogeneous.

IQM Tools has support documentation in the form of text files within the tool's directory structure, but not inbuilt to the system's interface. Contextual information, in the form of tool tips, is also not widely available.

Some buttons belonging to the same group are enclosed in a border line which organises the interface, but this does not apply to all button groups, such as the export buttons in the lower left corner. The spacing principle between groups of buttons is also not applied consistently, so some groups appear more distant than others without a definitive reason.

4.5 MATLAB Simbio

MATLAB Simbio [18] is offered as part of MATLAB [1], and offers system biology and pharmacokinetic/pharmacodynamic model building and analysis functionality.

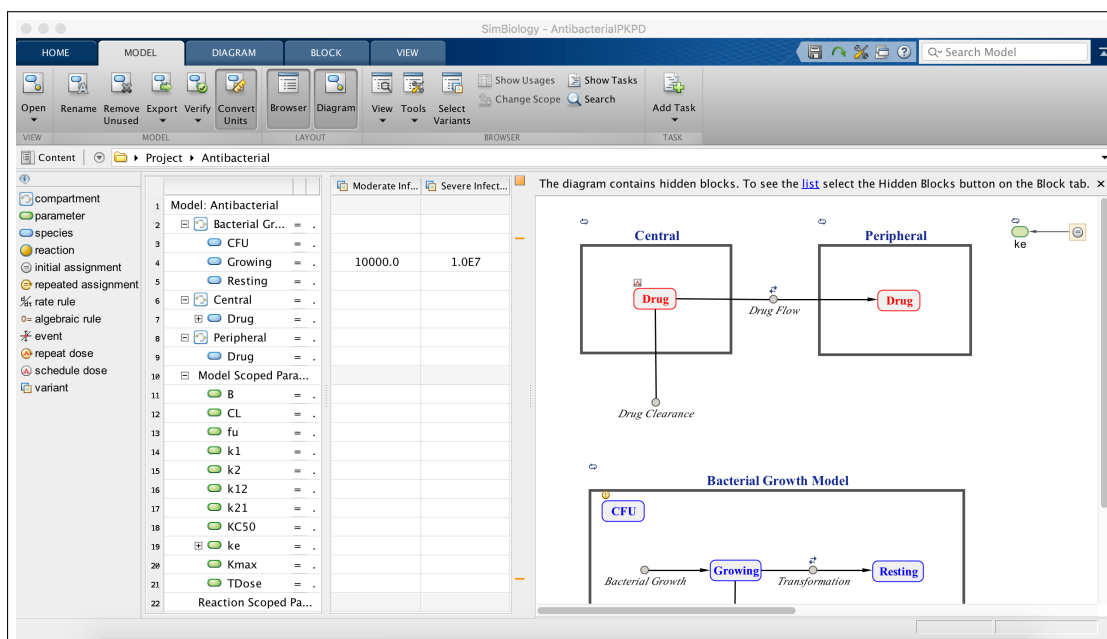


Figure 4.6: The MATLAB Simbiology [18] interface, displaying a model.

4.5.1 General Observations

This application appeared to be the most mature in terms of functionality and design, following MATLAB's design style and pattern [1], as seen in Figure 4.6. It felt well

designed because we felt it incorporated good usability practices, such as tabs and iconography on GUI buttons using popular interface metaphors [10], which are likely to help users understand what the buttons do quicker. Breadcrumb navigation represents files and folders in the file selection bar, which offers quick, inline navigation to the desired file or folder [59]. The task-bar design mimics that of the Microsoft-developed Ribbon interface [60], which is used another popular application like Microsoft Office [61]. We suspect that this allows new users to immediately feel familiar with the application, even if it would be their first time using it.

We agreed that this application likely offers the best model for us to design CPiME's GUI design off, since we suspected that the target audience is likely similar.

4.5.2 Design Observations

We found that MATLAB Simbio follows Nielsen's Heuristics significantly. There is a system status bar along the bottom of the GUI, and the application's GUI resembles another popular applications, Microsoft Office [61]. Undo and redo are both supported, design is consistent and error prevention was developed by providing suitable user feedback when needed. The application also offers keyboard shortcuts for more experienced users, and displays critical information at all times, so users do not need to remember it as they use the interface. Contextual help information and technical support documentation are both available.

Drop down menus also keep extra information out of the way until it is needed, following the 80/20 rule [62]. The Gestalt principles [14] also seem observed, for example, the proximity and styling of UI elements seem to be used to group them together, or distinguish them. Users see different buttons in the menu bar together as a whole, as the continuity principle affords this. Different shades are used to indicate which elements are in the figure and ground of the interface.

4.6 CPi-IDE

CPi-IDE was one of the previous attempts at creating a GUI for use with CPi [63].

4.6.1 General Observations

CPi-IDE offers a simple interface which displays the contents of the text file containing the CPi model definition on the left. It offered similar functionality to that of CPiME, including CPi model definition editing, viewing ODEs, model simulation, plotting of the simulation and phase plots, and also the possibility to logically verify models.

Although the application feels basic and unfinished at times due to elements not being positioned appropriately, the functionality offered seems robust. The plotting functionality allows users to remove species and re-plot the graph easily. It also provides a graphical formula builder.

Unfortunately, it reportedly does not scale well and becomes unresponsive if it is made to perform complex simulations [3]. Regardless, CPi-IDE offered an appropriate starting point to the design and functionality which should be included in the CPiME GUI, primarily because it was tailored to CPi [2].

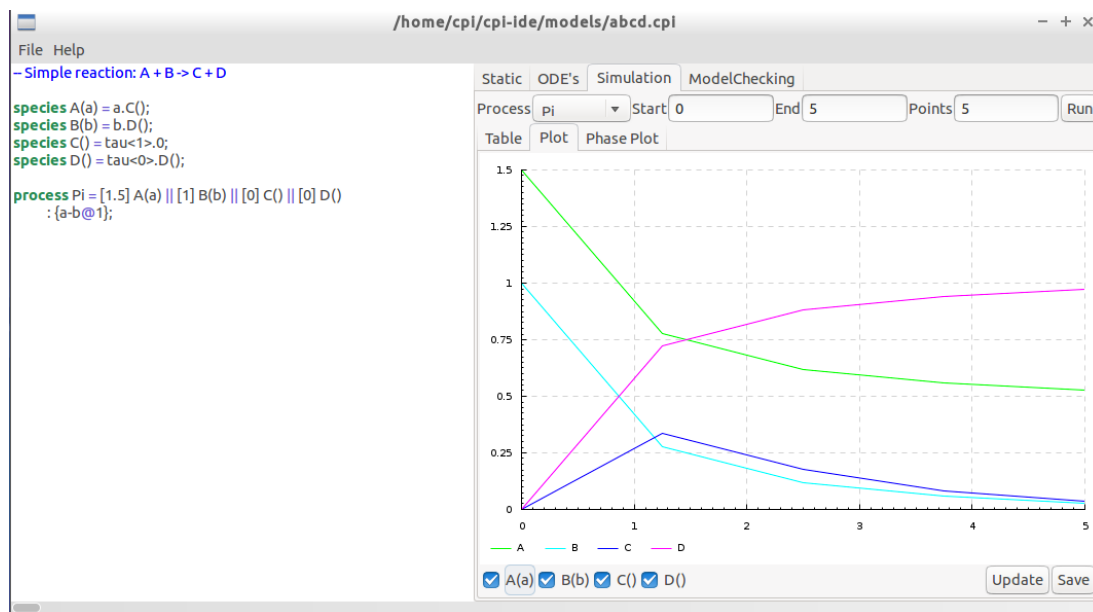


Figure 4.7: The CPi-IDE application [63], displaying a model definition and a plot of its simulation.

4.6.2 Design Observations

CPi-IDE's GUI is organised using nested tabs. The design is clean and simple, and some Gestalt principles are observed, such as grouping elements by proximity and similarity which aids users in navigating the interface.

The "file" menu allows users to open a file selection dialog box, and to save the model displayed, and we feel that it might have been more appropriate to display this functionality permanently on the GUI. Even though there is a "help" menu option, this menu only generates a dialog about the program. There is no contextual help information available and we could not find evidence of any other Nielsen Heuristic being observed.

4.7 Summary of Design Observations

Tool	Heuristics Observed	Heuristics Not Observed
INA	6	4
KaDE	1	9
KaSim	7	3
Visual SPiM	9	1
IQM Tools	3	7
MATLAB Simbio	10	0
CPi-IDE	2	8

Table 4.1: The total number of Nielsen's Heuristics [43] observed for each application.

Chapter 5

Code Re-factoring and Command Line Interface Redesign

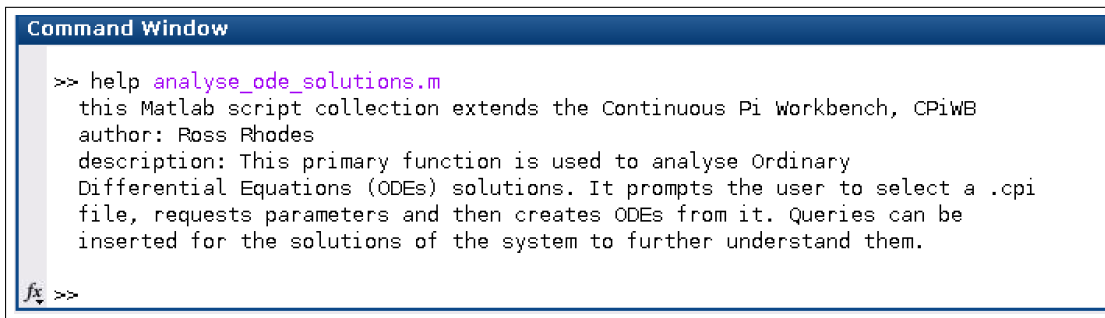
One of the planned goals of this project was to re-factor the existing source code of CPiME [3] as we wanted to allow users to call the different functions of the CPiME CLI independently. This additional flexibility would make the functions more suitable for plugging into data processing pipelines, and also allows users direct access to the functionality without needing to load the interactive CPiME CLI. Another planned goal was to redesign the existing CLI to make it more interactive, allowing a user more flexibility when following a function rather than necessitating a strict, logical flow. The work described in this chapter is targeted to the Persona described by Ross McIntosh, found in Section 6.3.2. We believe it could make CPiME more efficient to use for more technical users.

5.1 Inline documentation and the Private Folder

While following the introductory MATLAB [1] courses mentioned in Chapter 2 [20], we discovered that MATLAB supports direct, external calling of functions defined in specific files without any additional setup. As a result, we did not feel the need to re-factor the existing code to expose suitable entry points, as we felt that this was already available. Instead, we focused on analysing the existing code to identify how the functions rely on each other and how the chain of function calls take place. We generated the built in "Code Dependency Report" to assist our effort — a copy of it can be found in Appendix A. We identified nine function files from the previous total of twenty-eight which were suitable entry points into the code logic, being reasonably independent and accepting parameters which a user can easily provide. We used another MATLAB trick to organise the folder contents, by placing the unsuitable entry point files in a folder called `private`. Function files defined inside a `private` folder can only be called by function files immediately one level above them, and not from the MATLAB command line [64]. This successfully limited the scope of the functions and will show future users of CPiME that only nine functions files should be called directly.

We improved the existing inline code documentation of the nine function files to help users quickly understand the implemented functionality, and also described the expected parameters. CLI users can now use a standard MATLAB CLI support

documentation function, `help` [65], to return information about a CPiME function. An example is shown in Figure 5.1.



```
>> help analyse_ode_solutions.m
this Matlab script collection extends the Continuous Pi Workbench, CPiWB
author: Ross Rhodes
description: This primary function is used to analyse Ordinary
Differential Equations (ODEs) solutions. It prompts the user to select a .cpi
file, requests parameters and then creates ODEs from it. Queries can be
inserted for the solutions of the system to further understand them.

fx >>
```

Figure 5.1: Using the inbuilt MATLAB `help` [65] functionality to return information about the `analyse_ode_solutions` function.

We also considered merging separate function files together, and using nested or local functions to define more than one function in one function file. We abandoned this effort, however, when we realised that the length of two merge candidate files (`answer_query.m` and `validate_query.m`) were over three-hundred lines long, so it was more appropriate to leave each function in its own file.

We hypothesise that these changes will improve the experience of new users of CPiME, allowing them to understand how to make use of the extension with greater ease than the previous file organisation structure.

5.2 CLI Modification

After spending more time investigating CPiME, we realised that a significant degree of flexibility and interactivity is already implemented. Users can pick the function they want to use from the initial menu, and they can also easily quit most processes they are currently in, and return to the main menu.

We consulted relevant literature [66] [67] and believe that the CLI for CPiME already implements those recommendations. Therefore, we chose not to change the existing CLI implementation and allocated the time we had planned for this part of the project to subsequent tasks.

Chapter 6

Graphical User Interface Design and User Evaluation

Four different versions of the mock-up designs were created as we involved different stakeholders, and evaluated their feedback. The final set of mock-up designs, found in Appendix B, incorporate the best-practices from literature, other tools analysis and user feedback sessions.

6.1 Design

Our motivation behind designing the GUI was making it as straightforward to use as possible, allowing users to navigate through the functionality using little more than their intuition. We decided to use Balsamiq Mockups 3 [68] to generate the mock-up designs, as we found it fast, easy to use, and it offered the UI elements we needed.

The screen which allows users to generate ODEs can be seen in Figure 6.1. The full set of final mock-up designs can be found in Appendix B, with full descriptions of how we anticipated a user should use the screen.

6.1.1 Design Choices

We planned one screen for each existing function in CPiME. While designing the screens, we were aware of Ockham's Razor [69], which recommends always selecting the simplest design given a choice. We simplified designs to include only elements which had to be present on that specific screen. We also exploited the Gestalt Principles [14] to depict element grouping naturally, such as with the buttons near the file selection bar in Figure 6.1. The buttons are close together, aligned and also have the same dimension. For example, one can immediately tell that the buttons near the file selection bar fall into a different group to the menu buttons along the top of the GUI.

We designed top-left heavy interfaces since this is the *primary optical area* given the Gutenberg Diagram hypothesis [13], and most screens have buttons to perform the execution in the bottom right since that is the *terminal area* of the screen, such as the "Analyse Solutions" screen found in Figure B.6.

We designed the menu structure to resemble the Microsoft Ribbon interface [60] which we observed in MATLAB Simbio [18] as we believe that this would make it

easier for our users to start using the GUI, especially if they are regular MATLAB [1] users too. We followed Fitts' Law [70] when establishing the size of the buttons — it states that the time it takes to move to a target depends on the target size and the distance to the target. We therefore designed the screen to have a small size with comparatively large menu bar buttons, and organised what we believe are the most frequently used elements to be in the top-left and bottom-right areas.

We gave preference to a symmetric design as this is said to be aesthetically pleasing [71]. We used a tabbed interface metaphor given its prevalence in the alternative applications we surveyed in Chapter 4, such as in the Compare Processes screen, seen in Figure B.4. We also designed the mock-up designs using iconography rather than text on the buttons or labels, as we believed that this would allow users to understand what the buttons do at a glance. Finally, we aimed to fit all screen elements within the screen, letting users observe all parameter values even after they selected them. We believe that this is a significant improvement over the CLI, where new commands showing up on the interface would automatically scroll the onscreen text upwards, taking them out of sight.

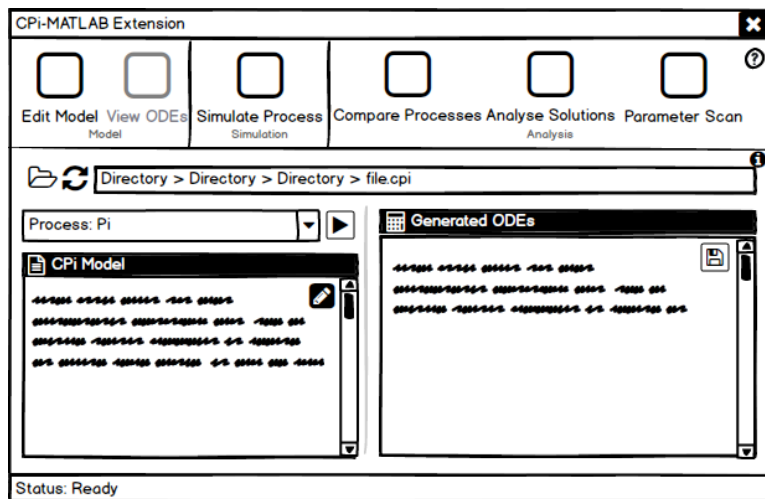


Figure 6.1: A mock-up design of the "View ODEs" screen, allowing users to generate ODEs for a selected CPI model definition. The full set of final mock-up designs can be found in Appendix B.

6.1.2 Usability Inspections

We performed a Cognitive Walk-through [5] and Heuristic Evaluation [6] immediately after designing the first set of mock-up designs, before we showed them to any stakeholders or prospective users. We thought through the four questions as listed in Section 2.6.6.1, and then reorganised the menu bar buttons to better reflect the groups of terms we believe a user would be looking for. Before the Cognitive Walk-through, the buttons were organised as in Figure 6.2.

The initial designs allowed users to check the generated ODEs from each screen. We moved away from this, as we realised through the Cognitive Walk-through that

it is unlikely that a user would need to check the ODEs every time they use other functionality in CPiME.

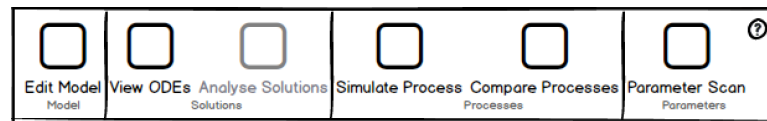


Figure 6.2: The first version of the mock-up design of the menu bar of the CPiME GUI, which was reorganised following the Cognitive Walk-through.

We also performed a Heuristic Evaluation [6] using Nielsen's Heuristics [43]. The results influenced the initial design and prompted us to include additional features, as seen:

1. **Visibility of system status:** We added a status bar along the bottom, and planned on using dialog boxes to present feedback. Initially, we planned on using three dots to indicate the page position when using screens with multiple pages.
2. **Match between system and the real world:** We matched the language used on the GUI to what users without a computational background might be more familiar with: "Settings" instead of "Parameters" and "Model" instead of "Definition". We planned on using GUI elements which offered perceived affordances [11].
3. **User control and freedom:** We included a "Reset" button on screens with a settings panel. We also planned on investigating inserting "Undo" and "Redo" buttons, although this was anticipated to be complex. The "Back" button, such as in Figure B.8, offers user control.
4. **Consistency and standards:** We used the same design for file selector bar and menu bar along on top in all screens. We also used the same icons and labels throughout, and a design which is influenced by other popular applications like MATLAB Simbio [18].
5. **Error prevention:** We planned on inserting confirmation [72] boxes before significant processing or clearing of data. We would test this further during the post-implementation testing session.
6. **Recognition rather than recall:** We display important information like CPi file selection or CPi definitions in all screens and throughout other parameter selection process.
7. **Flexibility and efficiency of use:** We would look into enabling keyboard short-cuts to accelerate GUI use for more experienced users.
8. **Aesthetic and minimalist design:** We kept the design clean and organised using the Gestalt Principles [14].
9. **Help users recognise, diagnose, and recover from errors:** We would notify users if a parameter value is outside the acceptable range or in an incorrect format, after using contextual help information to guide them.

10. **Help and documentation:** We would create Support Documentation, accessible from a ? button, and use contextual help information, accessible from a i button [12].

6.1.3 Stakeholder Influence

We discussed the first set of mock-up designs with other members of the CPi team who recommended additional changes. For example, the initial designs allowed users to check the ODEs generated from each model from each different screen. A stakeholder felt that this resulted in a lot of repeated functionality, so this functionality was retained only in the "View ODEs" screen. We also changed the design so models could only be edited from the "Edit Model" screen, rather than every screen as originally designed. Instead, we placed an edit button, which was originally designed with a pencil icon on it, overlapping the "CPi Model" text area to let users quickly switch to the "Edit Model" screen from anywhere. This is in line with the Flexibility-Usability Trade-off, which states that increased flexibility in a system results in less usability [73]. We were also asked to ensure that common GUI elements, such as the file selection bar, "CPi Model" text areas and process drop-down list selector were consistent [44] across all screens.

This feedback process led to two iterations of the mock-up designs, with this third version being presented to end users as part of the "Pre-Implementation User Testing" phase.

6.2 Pre-Implementation User Testing

Three different male participants with a Systems Biology or Bioinformatics background evaluated the mock-up designs individually. One had prior CPi experience, and all had at least six to ten years of experience in the field and considered themselves as having advanced or better computer skills. Their highest level of education achieved was a doctoral level degree.

We picked another three participants for the Post-Implementation User Testing session, bringing the total number of participants to six. This is above the recommended minimum total of five participants [74], needed to effectively find usability issues. In fact, involving more than five participants is said to have a small impact on the total number of usability issues found.

6.2.1 Test Procedure

A central part of UCD is performing user testing when designing an application [34]. We provided participants with consent forms prior to commencing the tests, which can be found in Appendix C, informing them that the session will be audio recorded and that notes will be taken. We carried out the same pre-implementation test with each participant — the full plan can be found in Appendix E.

The first section of both the pre- and post-implementation tests involved a brief interview [8] to understand the participant's attitude to Systems Biology or Bioinformatics software, and a description of CPi. This is described in Section 7.2.

The next part involved the mock-up design evaluation. Our main objective was to understand whether the designs were in line with the participants' expectations for what such a GUI would look like. We started by priming the participants by describing what the mock-up design would be of, to encourage them to create a mental image. Then, we showed the participants the design and asked them whether it fell within their expectations; we also asked them to describe how they would perform a sample task by using the screen. We tried to understand whether the design principles or practices we used, such as interface metaphors or pre-populating fields with default parameter values, were indeed known and favoured by our participants. This process was repeated for each mock-up design for the entire GUI.

6.2.2 Results

We discuss the interview questions and analyse the responses from both test sessions together in Section 7.2. Apart from some exceptions, the participants' expectations of the screen were generally aligned with the mock-up designs, and their intuition correctly guided them in anticipating how to use them.

6.2.2.1 Consensus

Participants seemed to find the screens mostly consistent and commented on their similarity. Their reservations were also mostly similar. All participants found the initial page number interface metaphor selected unsuitable for this type of interface, believing it was better suited for mobile devices, as seen in Figure 6.3. They all called for a clear description of what the "refresh" button, such as that in the "View ODEs" screen (Figure 6.1), actually does — either by inserted text labels or by using contextual help information [12].

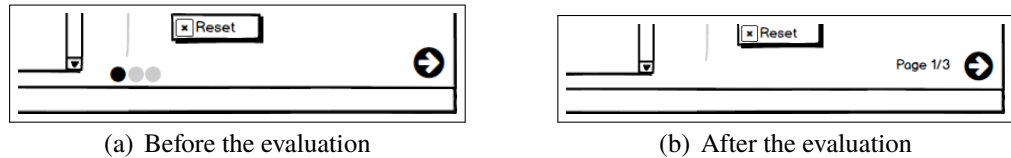
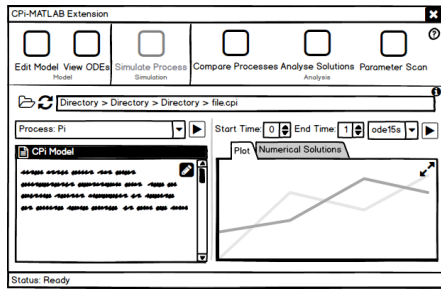
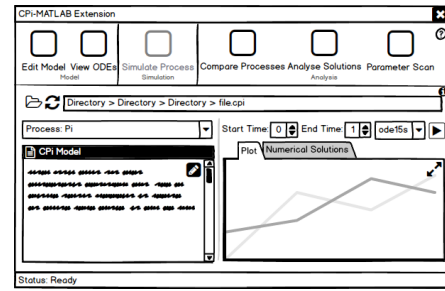


Figure 6.3: Our participants found the first page interface metaphor we used more suitable for mobile devices, not desktop applications (a). We redesigned the page interface metaphor to a more traditional one (b).

Before the evaluation, the screens all had a "Play" button on the left side of the screen which displayed the CPi Model definition, as in Figure 6.4a. This allowed users to generate the ODEs prior to proceeding with the processing, just like the CLI. All participants found this intermediate step unnecessary, so we removed it and retained only one "Play" button per screen, anticipating the ODE generation to happen silently in the background instead, like in Figure 6.4b. Users also found it challenging to identify which elements were editable on the second "Parameter Scanning" screen. We added iconography which we believe makes it more evident, as seen in Figure 6.5a and 6.5b respectively.

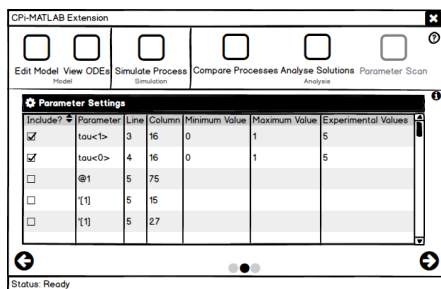


(a) Before the evaluation

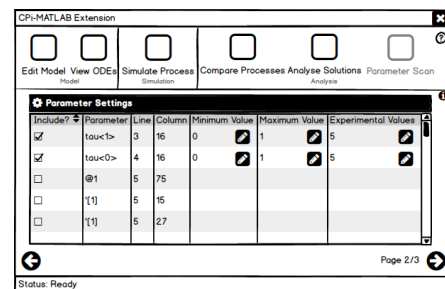


(b) After the evaluation

Figure 6.4: Our participants found the multiple "Play" buttons on many screens confusing (a). We retained only one "Play" button per screen, making the ODE generation invisible (b).



(a) Before the evaluation



(b) After the evaluation

Figure 6.5: Our participants found it difficult to identify which elements were editable (a). We added "Pencil" icons to make it more evident (b).

6.2.2.2 Differing Views

One user did not immediately realise that the buttons along the top bar were intended to change the screen displayed, and thought that they were actually buttons to perform a specific action instead. Another noted that the parameters requested for each screen were not consistent, and that this could be a source of confusion. This participant also incorrectly interpreted how the "Analyse Solutions" screen (Figure B.6) should be used, as the test results positioning seemed to have misguided them. We believe that these difficulties could be due to the fact that a wire-frame mock-up design was being used and parts of the design could be ambiguous, and this could be mitigated by using appropriate contextual help information and appropriate support documentation in the implemented GUI. Two participants felt that displaying the CPi Model definitions on each and every screen was not necessary, and it would have been more suitable to display them only on the "Edit Model" screen (Figure B.1). We appreciated this concern, but in order to remain consistent with CPi-IDE [63] and to provide users the possibility of using each screen independently of the other, we retained this setup.

6.2.3 Test Improvements

We suspect we could have been clearer in wording the question "Is this mock-up design what you expected to see?", as while we wanted participants to reply with a yes or a no, most felt compelled to describe the screen instead and describe how one would use it.

6.3 Personas

Personas [9] would have been ideally been created at the very beginning of the UCD process and used during all parts of it. Personas could have helped guide the selection of participants in the Pre-Implementation User study and also our design choices when designing the mock-ups.

We chose to create Personas at this stage just the same to help us with the remainder of the project and to be available for future CPiME team members. Although many different Personas could have been created, we chose to focus on two which we believe represent a significant proportion of CPiME's user base: experienced System Biologists/Bioinformaticians from either a Computer Science or a Life Sciences background. Some of the interview responses, which are discussed in Section 7.2, served as inspiration. These Personas are completely fictitious and any similarities to actual people, living or dead, are purely coincidental. We avoided references to gender, age and marital status to avoid unconscious bias.

6.3.1 Emily MacDonald

City: Glasgow, U.K.

Job Description: Emily MacDonald, seen in Figure 6.6, has over ten years of experience leading a team of five System Biologists investigating the microbiology processes leading to malignant skin cancer. Her typical day involves planning and conducting experiments in the morning, and then analysing the results using her computer in the afternoon. She enjoys experimenting with novel tools, but sometimes finds it hard to get them up and running and to be sure of the impact of the parameter values being used. She usually keeps notes of dry lab experimental procedures, so she can easily replicate the data processing tasks. Emily usually gets the subway to work.



Figure 6.6: System Biologist Emily MacDonald conducting an experiment [75].

Education: She graduated with a first class undergraduate degree in Biological Sciences from the University of Glasgow. She started reading for a PhD shortly thereafter and successfully defended her thesis in 1992, in which she developed a novel experimental method for monitoring malignant tumour growth in epidermal tissue. Apart from a single class in her undergraduate degree which introduced her to Scientific Computing practices and programs, Emily's computer skills are self-taught.

Aspirations: Emily's life goal is to create an extensible model of epidermal malignant cancer, a disease which took the life of her first Biology teacher. She hopes that this model would be used to create a cure for it. She also wants to become a qualified Yoga

instructor.

Favourite Colour: Green **Favourite Food:** Roast Chicken

6.3.2 Ross McIntosh

City: Edinburgh, U.K.

Job Description: Ross McIntosh, seen in Figure 6.7, has been coding for over 20 years, and is currently part of a team of four software engineers creating data processing pipelines for a synthetic biology company. His typical day involves a scrum meeting in the morning to report his progress and get set his day's work. Ross typically codes all day, stopping only for lunch or to ask a question to his peers or manager. Ross enjoys experimenting with new tools and has a knack for remembering minute details with little effort. His obsession is writing ever leaner code, and believes that this is how his competitive streak manifests itself. Ross usually gets the bus to work.

Education: Ross read for a B.Sc. (Hons.) in Computer Science at the University of Barcelona, Spain, graduating in 2001. He later read for an M.Sc. in Informatics at the University of Edinburgh. Ross is a great fan of online courses and complements his formal education with these. He also participates in coding competitions regularly, and believes this has granted him invaluable coding experience.

Aspirations: Ross' life goal is to create a software package which would vastly improve Biology researchers' performance. He would also like to become a certified fitness instructor.

Favourite Colour: Blue **Favourite Food:** Vegan Mushroom Burger

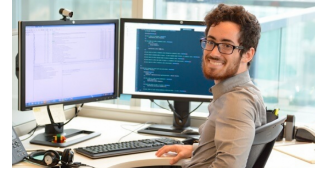


Figure 6.7: Software Engineer Ross creating a new data processing pipeline [76].

Chapter 7

Graphical User Interface Implementation and User Evaluation

We split the implementation phase into two iterations, each handling half the GUI screens as seen in Appendix B. We successfully completed the first iteration, implementing the first three screens, and then testing them with three participants. We used the feedback to influence the development of the next three screens. In the next iteration, we managed to implement two of the three screens, but then had to stop due to time constraints. We managed to develop a fully-functioning GUI for five of the six CPiME CLI functions. We have enclosed screen-shots of the implemented GUI in Appendix D. The work described in this chapter is targeted to the Persona described by Emily MacDonald, found in Section 6.3.1. We believe it could make CPiME more straightforward to use for less technical users. We have not detailed the GUI functionality in this section; a full, detailed description can be found in the support documentation of the GUI in Appendix F. This documentation was written following the recommendations from Barlett *et al.* [37], as noted in Section 2.5.4.

7.1 Implementation

We investigated two different ways of creating a GUI in MATLAB [1], GUIDE [77] and App Designer [78]. GUIDE was the first attempt at creating a GUI builder and has been available, and includes a limited set of just thirteen UI elements and a classic look-and-feel. App Designer is a very recent addition to MATLAB, released only in 2016 and includes a greater range of UI elements and a modern look-and-feel. We originally planned on using App Designer in this project, but discovered very limited graphics support [79] in the 2016 version which was only improved in the 2017 version. Since the computers of the School of Informatics at the University of Edinburgh ran MATLAB version 2015, and with version 2017 being so cutting edge, we chose to go for the more established GUI builder instead. This ensured that our GUI could run on most computers running MATLAB, rather than only those running version 2017.

We managed our software changes and versions using Git [80]. While implementing the GUI, we kept the Personas presented in Section 6.3 in mind to guide any decisions we had to take.

7.1.1 Implementation Choices

We used one MATLAB Figure, named `gui.fig` and its accompanying `gui.m` for the entire GUI, so that the entire GUI is defined in one file, which makes it could make it easier for developers to analyse and led to fewer files in the file structure. Launching the GUI simply involves typing `gui` in the MATLAB command window.

We have not detailed the GUI functionality in this section; a full, detailed description can be found in the support documentation of the GUI in Appendix F.

7.1.1.1 Different Screen Implementation

GUIDE allowed us to layout the GUI visually, and then add functionality to the elements in each element's respective `Callback` function which is executed when the element is clicked. We also inserted some functionality in the `gui_OpeningFcn` in the accompanying `gui.m` which is executed just before the GUI is made visible, for example to display the default screen panel. We created six overlaid Panels, one for each screen, and another Panel along the top which contained the menu bar and its six Buttons (one for each screen). We set all screen Panels to hidden by default, and we display the Panel upon detecting a click on its respective Button by calling a custom `change_panel` function, passing the frame name identifier as a parameter. The "View ODEs" screen can be seen in Figure 7.1. Screen-shots of the entire GUI can be found in Appendix D.

7.1.1.2 Main Button Implementation

Some screens have a Button which executes the primary calculations of the screen, for example, the "Generate" Button in the "View ODEs" screen (Figure 7.1) or the "Simulate" Button in the "Simulate Process" screen (Figure D.3). Pressing these buttons is analogous to calling the respective function file directly in the CPiME CLI. To adapt these functions to the GUI, first we fully understood them and established what parameters needed to be passed for them to work. Then, we implemented each Button's `Callback` function such that it parses the information which has been typed into that screen's GUI, validating it if necessary. Validation feedback is returned via dialog boxes, which also display an appropriate, informative message, helping users recover from the error. For example, start and end times must be digits and not negative numbers, otherwise a dialog box is displayed asking the users to correct the error.

In all cases, we first tried to directly call the functions which were already implemented in CPiME and pass the input as parameters, allowing us to perform these calculations while reusing as much code as possible. Since the previous functions were primarily written to be used with the CLI, any user feedback was always returned to the CLI. We discovered that the previously implemented functions did return feedback from a uniform level in the code — rather, feedback was printed to the CLI whenever an exception arose, sometimes even deep within nested called functions. This meant that it was not straightforward to convert the existing CLI functions to work with the GUI. Even if we had adapted the top level function, a lower level function might inappropriately return feedback to the CLI even if it was called from the GUI.

We tried to implement an overloaded version of the function which would have allowed us to extend the function definition and adapt it for the GUI cleanly. However, we discovered that MATLAB does not support function overloading and has limited object-oriented-paradigm support. Due to time restraints, we could not significantly extend the pre-existing code and also test it. Therefore, we chose to replicate some function files and add a `_gui` suffix instead; we converted eight pre-existing functions into these function files with a `_gui` suffix. Each return user feedback using GUI elements instead of the CLI, for example using dialog boxes.

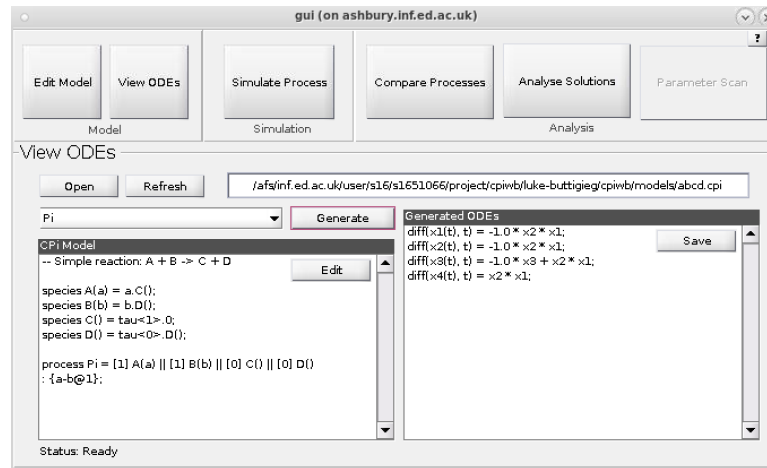


Figure 7.1: The implemented "View ODEs" screen, displaying generated ODEs.

7.1.1.3 Retaining the File Selection Between Screens

Once a user selected a file, we wanted the file and process selection to remain so even if the user moves to another screen. Since the GUI needs to store the details of up to four models which are being compared in the "Compare Processes" screen, we needed an efficient way of storing and retrieving the models' information. We implemented this using two different hash maps; one for the file name and path (`file_map`) and another for the selected process within the file (`process_map`). The index number value of the currently selected process from the process drop-down list (`curr_pname`) and the model number (1–4) (`curr_fname`) are stored in global variables. The model number is actually which of the four model processes is currently selected in the "Process" radio button group in the "Compare Processes" screen (Figure D.4). We use this number directly to see which data to retrieve from the hash maps. We also defined functions which open a file for each of the implemented screens, all starting with an `openfile_` prefix. These functions are invoked whenever a user goes from one screen to another, populating the new screens UI elements with the currently selected model's properties. The file path and name and the file contents are populated on each screen using this function.

7.1.1.4 Reset Functions

We implemented reset functions for each screen, all starting with a `reset_` prefix. They are invoked if the user closes the currently selected file from the "Edit Model"

screen (Figure 7.2), or else clicks the "Reset" Button which is available on some screens, like the "Compare Processes" screen (Figure D.4). These functions clear the selected file name, path and process in the hash maps mentioned in Section 7.1.1.3, and then change the onscreen UI elements back to their empty state. Some UI elements revert to disabled, as they were when the GUI was first launched. For example, the "Generated ODEs" text area in the "View ODEs" screen (Figure 7.1) is disabled when the "Reset" button is pressed.

7.1.1.5 Scalable GUI

We attempted to make the GUI scalable by changing the "Resize behaviour" in the GUI options menu to "Proportional", but we discovered that this led to unpredictable behaviour with parts of the UI elements disappearing sporadically. We discontinued our effort and the GUI is now of a fixed size.

7.1.1.6 Heuristics Implementation

Preventing errors from occurring and correcting them was an important heuristic we followed [43], so we used different Dialog boxes to our advantage, as in Figure 7.2. Moreover, all UI elements have tool tips assigned to them which appear when hovering over them, like in Figure D.5. We also disable parts of the screen which are not ready to be used, such as the right panel of the "View ODEs" screen as seen in Figure D.2, unlike applications like Visual SPiM [54]. We implemented the status string as a simple text label on each of the string panels, like MATLAB Simbio [18]. We programmatically change the contents of this status label depending on what the user is doing, and then reset it if the user resets the screen, as described in Section 7.1.1.4. A "Help" button was implemented in the upper right corner as designed. This button displays a dialog box advising users to check the `help.pdf` file for Support Documentation. The full support documentation can be found in Appendix F of this document.

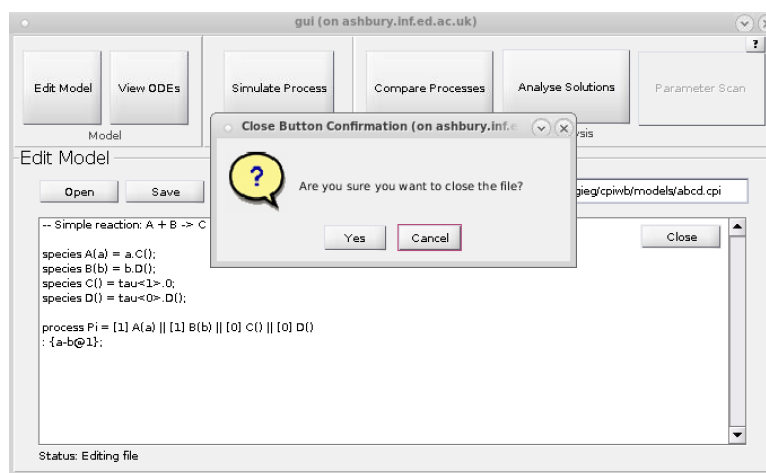


Figure 7.2: The implemented "Edit Model" screen, displaying a warning dialog box before closing a file.

7.1.2 Differences between Mock-Up Designs and Implementation

We discovered that MATLAB does not fully support the design we had created, so we had to adapt it accordingly. For example, although we originally wanted to provide a way of displaying all contextual help information by clicking on an "i" button, we could not identify a way of doing this and had to abandon our efforts.

7.1.2.1 Interface Metaphors

Interface metaphors [10], such as a tabbed design and iconography over buttons, such as a "floppy disk" for the "save" button, could not be implemented as GUIDE does not support this natively. Instead, we used radio buttons for the "Compare Processes" screen (Figure D.4) and opened separate windows for the graph plot and numerical solutions in the "Simulate Process" screen (Figure D.3). We also did not end up displaying any icons over any UI elements as planned, as GUIDE provided no means of doing this. We resorted to using text labels only instead, which was actually recommended by a participant in the pre-implementation user testing. We also discovered that MATLAB did not support displaying the file selection bar format using breadcrumb format [59] as designed, and had to settle for a simple string layout instead.

7.1.2.2 The "Compare Processes" Screen

The "Compare Processes" (Figure B.4-5) was designed with a two screen layout, but the implementation (Figure D.4) only contains one screen because the graph plotting do not appear within the UI but rather in standalone windows. There were multiple reasons for this: the lack of tabbed interface support and the difficulties experienced when creating a scalable GUI (Section 7.1.1.5) made it difficult to fit all elements on a small screen.

7.1.2.3 The "Analyse Solutions" Screen

In the designed "Analyse Solutions" screen as seen in Figure B.6, the "Test" Button is on the left and the "Clear" Button is on the right. The implemented version (Figure D.5) has a "Test" Button on the right and a "Reset" Button on the right. We changed it to maintain consistency [44] among all the screens.

7.1.2.4 The "Simulate Process" Screen

The "Simulate Process" screen (Figure B.3) was designed with an inline graph plot and numerical solutions. Due to the lack of tabbed interface support and difficulties creating a scalable GUI (Section 7.1.1.5), the screen has a "Settings" panel on the right instead (Figure D.3). The graph plot and numerical solutions open in separate windows.

7.1.2.5 Impact of Time Constraints

The second iteration of the implementation process had to be halted before we finalised it due to time constraints. We chose to organise our work in two iterations instead of one as soon as we discovered that there was not going to be enough time to implement

the entire GUI and then perform the post-implementation user testing afterwards. We believed it was better to conclude at least one implementation iteration in its entirety and then fit in as much of the second iteration as we could. Consequently, we did not finish developing the complex "Parameter Scan" screen, which explains why its button is disabled on all screen-shots, and we also did not perform Post-Implementation User Testing of the final three screens.

7.2 Interviews

We describe the demographics and background of the Post-Implementation User Testing in Section 7.3. We asked participants three open ended questions and their opinion on six different statements in the interview. We also checked what their experience using the alternative tools which we analysed in Chapter 4 was like. The entire test plan including the interview questions can be found in Appendix E.

7.2.1 Question Structure

The aim of this project was to improve and expand the interface of CPiME, primarily by creating a GUI which could make it easier to use. Hence, our first interview question tackled the participant's attitude to interface types: whether a CLI or a GUI was actually preferred and why. We then asked them to recall their most frustrating and rewarding experiences using Bioinformatics or Systems Biology software. Next, we asked participants to give their opinion on six statements which describe their attitude towards when and how they use user manuals, whether they expect themselves to use applications out of the box without consulting any help documentation, and their involvement of peers in understanding how to use such applications. We offered standardised Likert-scale [81] responses to the participants to reduce any biases. Finally, we mentioned the names of the applications we analysed in Chapter 4, and asked them to describe their past experiences using them, if any at all.

7.2.2 Answer Analysis

Since only six participants were involved in the interview, we did not analyse the data quantitatively but rather qualitatively.

7.2.2.1 CLI versus GUI

Participants preferred a CLI when using Bioinformatics or Systems Biology Software, citing efficiency, the possibility for computation parallelism and a higher degree of control. A participant also stated that CLIs usually make it easier to keep a natural log of execute by saving commands as scripts. Some participants claimed that GUIs are useful when they are new to a software package and need to experiment with the functionality to understand it better, but echoed complaints found in literature [34] that GUIs in scientific software often fall short of expectations.

7.2.2.2 Sources of Frustration

Most of the frustration using such tools arose when users did not achieve the aims they set out to achieve. Participants cited various issues, such as difficulty installing, setting up and using applications. For example, knowing which parameters should be used for a specific result to be obtained is said to be tricky, as noted in literature [28]. Two participants mentioned limited access to the source code as a concern, also recommended in literature [67].

7.2.2.3 Practices Worth Praising

On the contrary, participants were pleased when the tools worked as they needed to and provided them with insights which helped them advance their work. Four participants praised "easy-to-use" applications which feel natural to use (even by a user without a computational background), keep a log and also allow backtracking — characteristics which are aligned to Nielsen's Heuristics [43].

7.2.2.4 Attitudes Towards Support Documentation

Support documentation was generally popular sources of guidance for participants, but there was no clear usage pattern. Approximately the same number of participants said they used support documentation before using an application for the first time, and did not expect themselves to know how to use applications out-of-the-box. The reverse was also true, with participants expecting themselves to know how to use the applications immediately. These participants only said they used support documentation when they ran into a difficulty and also relied on contextual help to find their way around an application. Most participants claimed not to request assistance from peers when using such software, while younger participants were more likely to involve them.

These answers reveal an interesting distinction between our participants: some learnt by tinkering and experimenting, and others relied on support documentation first to ensure they are using an application appropriately.

7.2.2.5 Attitudes Towards Alternative Tools

Only two participants had used any of the alternative tools we analysed in Chapter 4 in the past, and they did not clearly remember how the tools looked or worked. Some had used other similar tools [82][82][83], but we did not get any valuable information about such experiences.

7.3 Post-Implementation User Testing

Three different male participants with a Systems Biology or Bioinformatics background evaluated the first three implemented GUI screens individually. None had used CPi before, and two were relatively younger (25–34 years old) and naturally had few years of experience in the field (1–5 years). The other participant was older and had over ten years of experience.

7.3.1 Test Procedure

Users were provided a consent form prior to starting the study — a copy of this can be found in Appendix C. We audio recorded the entire session and also took notes during the exercise. After the interview (Section 7.2), we used the concurrent Think-Aloud protocol [7] to discover usability issues in the first three screens of the implemented GUI by asking participants to speak out loud while performing two specific compound tasks. After briefly describing CPi, we conditioned their mind-set into speaking out loud by asking them to perform a simple, unrelated exercise — this is known as Think-Aloud training [84].

Our usability goal was for the participants to successfully carry out the tasks we assigned to them. The tasks were purposely designed to use secondary screens first, and not require participants to interact with the screens in a serial way which we hoped would mimic how a real user would use CPiME. The study is known as *within subjects*, as we showed all the UIs to all the participants. Our *independent variables* were the demographics and years of experience of the participants, and their prior experience using CPi or alternative tools. Our *dependent variable* was simply the success rate of performing the tasks assigned, since we were interested in uncovering usability issues.

7.3.2 Results

We found that the results obtained were generally coherent across all participants. We did not analyse how the independent variables caused the dependent variable to change, as we were only interested in implementing improvements to the GUI based on any critical incidents which happened.

Technical errors during the testing process caused the Think-Aloud session to be interrupted. Although we believed that the GUI screens was working as intended before performing the user testing, we experienced fatal errors in the second task which meant that users could not complete it. Usability issues also led to the first task not being completed. This means that the usability goal was not achieved for both tasks and all three participants. We introduced changes in the implementation after these results which we believe would change the success rate of achieving the usability goal should a future Think-Aloud session be held.

7.3.2.1 Common Tasks Outcomes

We will now describe the results which were common across all participants, and the impact they had on the implemented CPiME GUI.

All participants worked through the first part of the first task successfully, but became confused when trying to confirm whether the file was successfully saved to disk. Most of the confusion arose from the status string in the lower left corner which was originally not being updated when a user clicked "Save" — the status remained *Editing file*, as in Figure D.1. We changed this behaviour so the status string reverted back to *Ready* after a user clicked "Save". Another source of confusion was the original wording in the dialog box shown when the "Close" button was clicked, as in Figure 7.2. During the test, users were warned that closing the file could result in the loss of unsaved data, even though they had just saved the data. This led them to

doubt whether they had actually saved the file or not, especially because such warning messages are usually shown only in those circumstances when using popular software, such as Microsoft Office [61]. We simplified the dialog box warning message to the one in Figure 7.2, with no mention of unsaved changes. These errors halted participant's progress and we had to intervene, leading to failing to achieve the usability goals.

The second task uncovered technical issues with the implementation. The CPiME GUI should display the numerical solutions on a first Figure and a graph plot on a second figure when simulating a model. Instead, two participants were experiencing non-deterministic behaviour where the graph plot would end up within the numerical solutions figure and the figure intended for the graph plot would end up blank. After restarting the CPiME GUI, participants succeeded in completing the task.

We determined that this could be due to a thread racing issue, with MATLAB [1] using the same figure object instead of different ones because of the way we are calling separate secondary functions from the main GUI function. We considered implementing a thread waiting condition, but we did not find evidence that MATLAB supports this. Instead, we introduced small execution pauses of 2 milliseconds between the calling of the two functions which resolved the matter.

Moreover, the GUI did not always feel fully responsive which led some users to doubt their success at carrying out their assigned task. For example, dialog boxes sometimes took longer than expected to appear, and users reacted by repeatedly clicking the button. A thorough analysis of the GUI performance could be performed to understand the cause, but this was not attempted due to time constraints.

7.3.2.2 Other Outcomes

A participant recommended the use of progress bars to communicate the system status more explicitly. Another participant was also uncomfortable using the type of computer which was used for the test; this could have introduced additional unspecified bias which impacted the participant's test performance. In fact, this user was the only one who found it difficult to tell which screen one was actually on. We chose to increase the font size of the screen name to mitigate this. This participant also described the additional "Edit" Button over the CPi definition text box in most screens as redundant and said that it was not clear what the "Close" Button in the "Edit Model" screen was for.

7.3.3 Test Improvements

We successfully identified a number of usability issues with the first three GUI screens which we implemented, and took measures to mitigate the problems. The test results could have been verified by running the same test with the improved interface through different participants, which is known as a *between subjects* study. The results of this second round of testing could indicate whether the measures we took were effective. We would have avoided involving the same people for the second round of testing due to priming.

Ideally an additional two rounds of Think-Aloud testing would have been performed with the second set of three GUI screens, but these could not be performed due to time constraints.

Chapter 8

Project Evaluation

Our goal in this project was to improve and expand the existing interface of CPiME [3], chiefly by making it more flexible and straightforward to use. We believe that through our efforts, we have produced a set of deliverables which altogether could make CPiME more suitable for new users and easier to use for existing ones. We have also created a number of complimentary items, such as the Personas descriptions, which have laid the groundwork for future analysis and could guide the next CPiME team members into creating more targeted software. All in all, we believe that we have successfully experimented with applying a UCD approach to improve the interface of CPiME, and delivered on all the original objectives. Further experimentation is needed to prove whether these deliverables have indeed improved the interface of CPiME.

The original project plan was somewhat ambitious in the time it should take to design and implement the GUI, resulting in the project running two weeks behind schedule. We had foreseen that the project timing could be tight, so we had organised the original list of objectives using the MoSCoW method, which prioritises the components in must have, should have, could have and would not have groups. No would not have components had been identified. We reproduced this below and reported our results. This led to the second GUI implementation iteration having to be terminated prematurely, leaving out one of the six different screens.

Must Have: Background research for a list of best practices and design decisions for systems biology or other Bioinformatics software, re-factored source code, restore the CLI, design and implementation of GUI, Post-Implementation User Testing, project write up

Should Have: Pre-Implementation User Testing, system documentation detailing the functionality of the CLI and GUI

Could Have: Upgraded (revamped) CLI, a system walk through

8.1 Background Research

In Section 2.5, we presented a thorough analysis of HCI studies applied to the Bioinformatics or Systems Biology domain, and described two different approaches to UCD found in literature. We also described nine HCI principles and methods which we used in this project in Section 2.6.

We investigated and evaluated seven biology modelling applications which are likely used by a similar user-base as CPiME in Chapter 4, using Nielsen's Heuristics [43] and the Gestalt Principles [14]. We took note of what we believed were best practices which could have enhanced the usability of these applications. We also kept track of what seemed to be standard GUI design practices in this class of applications.

8.2 Re-factored Source Code and CLI Changes

Chapter 5 describes our efforts reducing the number of visible files for CLI users from twenty-eight to nine. We feel that this satisfies the main goal of improving the existing interface of CPiME, while keeping the changes invisible for CLI users. We also added more detailed in-line code documentation which is accessible using the MATLAB [1] `help` [65] command. We then asserted that after consulting relevant literature, the CLI does not require any further modification as previously envisaged.

8.3 Design and Implementation of a GUI

We condensed all the information we reviewed in Chapter 2 and 4 while also applying the knowledge we learnt in the MATLAB online-courses [20] to design the CPiME GUI. This effort is described in Chapter 6. We combined the best practices we found in literature with those found implemented in the other existing biology modelling applications. We performed two usability inspections on the first version of these mock-up designs. We presented the mock-up designs to other members in the CPiME team, which led to the creation of an additional three versions. Afterwards, we organised feedback gathering sessions with three experienced Bioinformaticians or System Biologists, and incorporated their feedback in the GUI design by redesigning elements of it. We also created two Personas which were used from that point on-wards in the project, and which could be used by future CPiME team members when designing or developing software.

In Chapter 7, we justified the important implementation choices we made while developing the fully-functional CPiME GUI, including how we organised our work-plan into two iterations. The full support documentation of the GUI can be found in Appendix F, which describes the functionality of the implemented GUI. We noted that almost all the functionality which was previously available through a CLI is now accessible using a GUI, which we believe has enhanced the flexibility of the interface of CPiME as we had planned. We explained that only the "Parameter Scan" screen was not implemented due to time constraints, and that there are some outstanding performance issues. We organised three testing sessions with another three experienced Bioinformaticians or System Biologists, and used the Think-Aloud protocol [7] to identify usability issues in the first three implemented screens of the CPiME GUI.

Since our goal was to improve and expand the CPiME interface, we did not compare the usability of the pre-existing CLI with the new GUI. We also did not perform repeated usability testing on different versions of the same interface, so we did not comment on how the GUI improved following the Post-Implementation User Testing described in Section 7.3.

8.4 Pre- and Post-Implementation User Testing

Although we initially graded the Pre-Implementation User Testing as a 'Should Have' not a 'Must Have', the literature survey in Section 2.5 made it clear to us that it is an essential part of the entire UCD process. Therefore, we organised three sessions with Bioinformaticians or System Biologists, as described in Section 6.2. After interviewing them to understand their attitudes and behaviour better, we presented the mock-up designs and gathered their opinions. We modified the mock-up designs on the basis of their opinions.

After implementing the first three GUI screens, we organised another three sessions with different Bioinformaticians or System Biologists. We interviewed them as we did in the earlier testing sessions, and then used the Think-Aloud protocol to understand their experience using the implemented GUI. We uncovered usability issues using this testing process, and modified the GUI based on these results. Although the usability goal of performing the assigned tasks was not achieved while performing the testing, we believe we were successful in discovering usability issues and trying a UCD technique first-hand.

We believe that this user testing process could have been made more effective had we known exactly what functionality MATLAB offers. This would have avoided discovering that some designed UI elements are not supported by MATLAB after performing evaluating them with users.

8.5 System Documentation and Walk-through

We also added more detailed in-line code documentation which is accessible using the MATLAB [1] `help` [65] command, as described in Section 5.1. We also created Support Documentation for the GUI, which can be found in Appendix F. We did not create a walk-through as it was classified as a 'Could Have' deliverable, and due to time constraints we prioritised other parts of the project instead.

Chapter 9

Conclusion

We have described how we explored and tried out different approaches of improving and expanding the user interface of an existing MATLAB [1] extension which facilitates the use of CPi [2].

CPiME could previously only be used through a CLI, and the aim of this project was to improve this and expand it, primarily by providing a GUI but also by enhancing the existing CLI.

9.1 Achievements

We have conducted a thorough literature survey and analysed other tools to familiarise ourselves with this class of scientific software. Then, we tried different UCD techniques which allowed stakeholders to be well involved in the design and implementation process, including Cognitive Walk-through [5], Heuristic Evaluation [6], Think-Aloud Protocol [7], Interview [8] and Personas [9]. We believe that we have managed to create a GUI which embodies many of the best practices, principles and methods we found, such as Interface Metaphors [10] and Affordances [11], Contextual Help Information [12], Gutenberg Diagram[13] and the Gestalt Principles [14]. We improved on it through pre- and post-implementation user testing by iterating through four different designs and two different implementations. We believe that by creating a GUI, we have provided inexperienced CPiME users with a possibly more straightforward way to experiment with the functionality available.

We have created two Personas which can guide future CPiME team members when developing future versions. We provided features which appeal more to Emily MacDonald (Section 6.3.1), such as a GUI, Support Documentation and Contextual Help Information. We also targeted Ross McIntosh (Section 6.3.2) by providing richer in-line comments, and reducing the number of visible function files from twenty-eight to nine.

9.2 Known Issues

The CPiME GUI cannot be maximised due to the issues described in Section 7.1.1.5. The CPiME GUI also occasionally responds slower than expected, as noted in Section

7.3.2.2. These issues could possibly have been remedied in the newer MATLAB App Designer [78] GUI builder. The "Parameter Scan" screen is not fully implemented and inaccessible from the CPiME GUI.

9.3 Future Work

While extensive effort was made to create a fully-functional GUI for CPiME in this project, numerous areas can be enhanced. We now present a suggested list of some enhancements.

9.3.1 Additional Functionality

This section explores some ideas for expanding the functionality that CPiME offers.

9.3.1.1 Logging

Logging functionality can be implemented to keep track of the user's actions. This is especially useful when repeating experiments or analysing results, and useful since scientists reportedly do not keep track of parameters used [24].

9.3.1.2 Time-stamping Exported Data

Saved ODEs and other exported data can be time-stamped to help users recall their context at a later date.

9.3.1.3 Parameter Scanning Screen

The screen allowing users to perform parameter scanning using the GUI has already been designed; it can now be implemented and then tested with a number of users.

9.3.1.4 Accelerators

As recommended by Nielsen *et al.*, users prefer flexible ways of interacting with interfaces [43]. Keyboard shortcuts allowing users to quickly navigate to a screen, navigate within the screen or start processing data would enable this.

9.3.1.5 Undo/Redo Buttons

Nielsen *et al.* state that users value "emergency exits" and being able to undo changes [43]. Implementing Undo/Redo functionality throughout the application would provide this reassurance.

9.3.2 Interface Improvements

While we are satisfied with our CPiME GUI implementation, we have compiled a list of improvements which could be made.

9.3.2.1 Caching of Simulations

In "Analyse Solutions" screen (Figure D.5), CPiME recreates the ODEs and the system simulation every time a statement is tested. If we had to cache the simulated model instead, we could provide quicker evaluations of the inserted LBC queries.

9.3.2.2 System Status

Apart from the Status String we implemented in this project, progress bars with percentages of progress could be used to convey the system status to the user, as suggested by a participant in the Post-Implementation User Testing.

9.3.2.3 Rich Model Definition Editor

The "Edit Model" screen text area can be enhanced to provide syntax highlighting and also additional text editor buttons such as cut, copy and paste buttons.

9.3.2.4 Graph Plots

One could explore how the graph plots could be displayed inline as part of the interface, as originally designed. Ideally, the design is responsive and a fresh render is displayed live, as soon as the user changes a parameter.

9.3.2.5 Interface Metaphors

Although tabs and iconography were originally part of the GUI design, MATLAB GUIDE [77] did not support them natively and we had to use workarounds instead. One could explore other ways of implementing these metaphors in the CPiME GUI, possibly programmatically.

9.3.2.6 Enhanced File Selection Bar

The file selection bar could be enhanced to use a breadcrumb navigation layout, similar to MATLAB Simbio [18] rather than its current string format.

9.3.2.7 Expandable GUI

While we tried create a scalable GUI, the inbuilt functionality proved unreliable at best. Providing such functionality would be worthwhile, especially when users have large screen resolutions.

9.3.2.8 Merge CLI and GUI Functions

Since adapting existing code to the GUI was not straightforward, some code duplication exists between pre-existing CLI code and new GUI code. Merging the two version of a function would reduce the number of files in the source code.

Appendix A

Code Dependency Report

The code dependency report showed us how the existing MATLAB [1] extension functions call each other, allowing us to find the files which we should place in the private folder. It is automatically generated by MATLAB.

Dependency Report

The Dependency Report shows dependencies among MATLAB files in a folder ([Learn More](#)).

[Rerun This Report](#)

[Run Report on Current Folder](#)

☒ Show child functions ☒ Show parent functions (current folder only)

☒ Show subfunctions

Built-in functions and files in toolbox/matlab are not shown

Report for Folder /afs/inf.ed.ac.uk/user/s16/s1651066/project/cpiwb/luke-buttigieg/cpiwb/matlab_extension

MATLAB File List	Children (called functions)	Parents (calling functions, current dir. only)
analyse_ode_solutions	current dir : answer_query current dir : create_cpi_odes current dir : prepare_legend current dir : select_single_process current dir : solve_cpi_odes current dir : validate_query	cpime
answer_query	toolbox : /symbolic/symbolic/@sym/sym.m toolbox : ? Multiple class methods match subs.m	analyse_ode_solutions
command_docs		cpime
compare_cpi_processes	current dir : construct_new_system_for_comparison current dir : determine_num_simulations_in_comparison current dir : retrieve_simulation_times current dir : separate_plot_comparison current dir : simulate_single_process current dir : single_plot_comparison	cpime
construct_new_system_for_comparison	current dir : create_cpi_odes current dir : display_definitions current dir : select_multiple_processes current dir : solve_cpi_odes	compare_cpi_processes
cpime	current dir : analyse_ode_solutions current dir : command_docs current dir : compare_cpi_processes current dir : parameter_scan current dir : simulate_single_process current dir : view_odes	
create_cpi_odes		analyse_ode_solutions construct_new_system_for_comparison parameter_scan simulate_single_process view_odes
create_process_simulation	current dir : plotCallback	simulate_single_process
determine_num_simulations_in_comparison		compare_cpi_processes
display_definitions		construct_new_system_for_comparison parameter_scan simulate_single_process view_odes
experiment_plots	current dir : plotCallback current dir : prepare_legend	parameter_scan
fillCallback		
find_common_species	current dir : prepare_legend	separate_plot_comparison single_plot_comparison
parameter_scan	current dir : create_cpi_odes current dir : display_definitions current dir : experiment_plots current dir : retrieve_experiment_info current dir : retrieve_simulation_times current dir : select_parameters current dir : select_single_process current dir : solve_cpi_odes	cpime
plotCallback		create_process_simulation experiment_plots separate_plot_comparison single_plot_comparison
prepare_legend		analyse_ode_solutions experiment_plots find_common_species simulate_single_process
retrieve_experiment_info		parameter_scan
retrieve_process_definitions		select_multiple_processes select_single_process

retrieve simulation times		compare_cpi_processes parameter_scan simulate_single_process
select multiple processes	current dir : retrieve process definitions	construct_new_system_for_comparison
select parameters		parameter_scan
select single process	current dir : retrieve process definitions	analyse_ode_solutions parameter_scan simulate_single_process view_odes
separate plot comparison	current dir : find common species current dir : plotCallback	compare_cpi_processes
simulate single process	current dir : create cpi odes current dir : create process simulation current dir : display definitions current dir : prepare legend current dir : retrieve simulation times current dir : select single process current dir : solve cpi odes	compare_cpi_processes cpime
single plot comparison	current dir : find common species current dir : plotCallback	compare_cpi_processes
solve cpi odes	toolbox : ? Multiple class methods match simplify.m toolbox : /symbolic/symbolic /@sym/massMatrixForm.m toolbox : /symbolic/symbolic/@sym/odeFunction.m toolbox : /symbolic/symbolic/@sym/sym.m	analyse_ode_solutions construct_new_system_for_comparison parameter_scan simulate_single_process
validate query		analyse_ode_solutions
view odes	current dir : create cpi odes current dir : display definitions current dir : select single process	cpime

Appendix B

Final Version of the Design Mock-Ups

We designed four versions of the design mock ups following usability inspections and incorporating the feedback from the stakeholders involved in the pre-implementation testing.

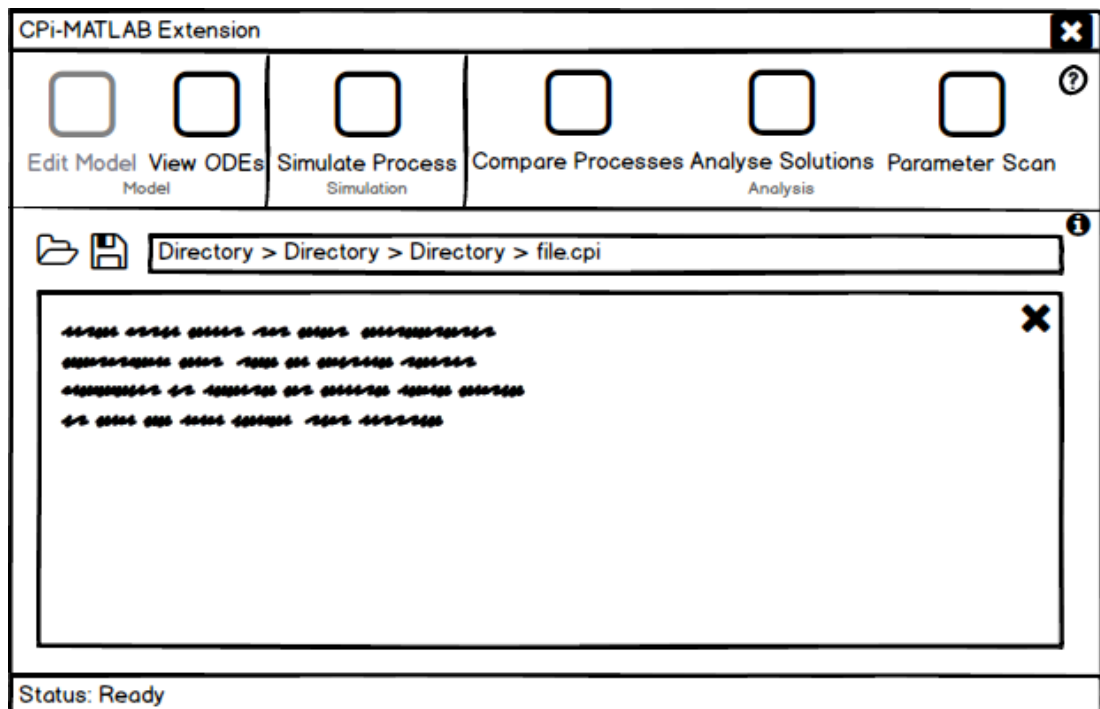


Figure B.1: The 'Edit Model' screen. A user is presented with the 'Edit Model' functionality activated by default. A user needs to load a file using the 'folder' icon, and perform any modifications in the text area below. The file can be saved using the 'floppy disk' icon, or else closed using the 'cross' icon.

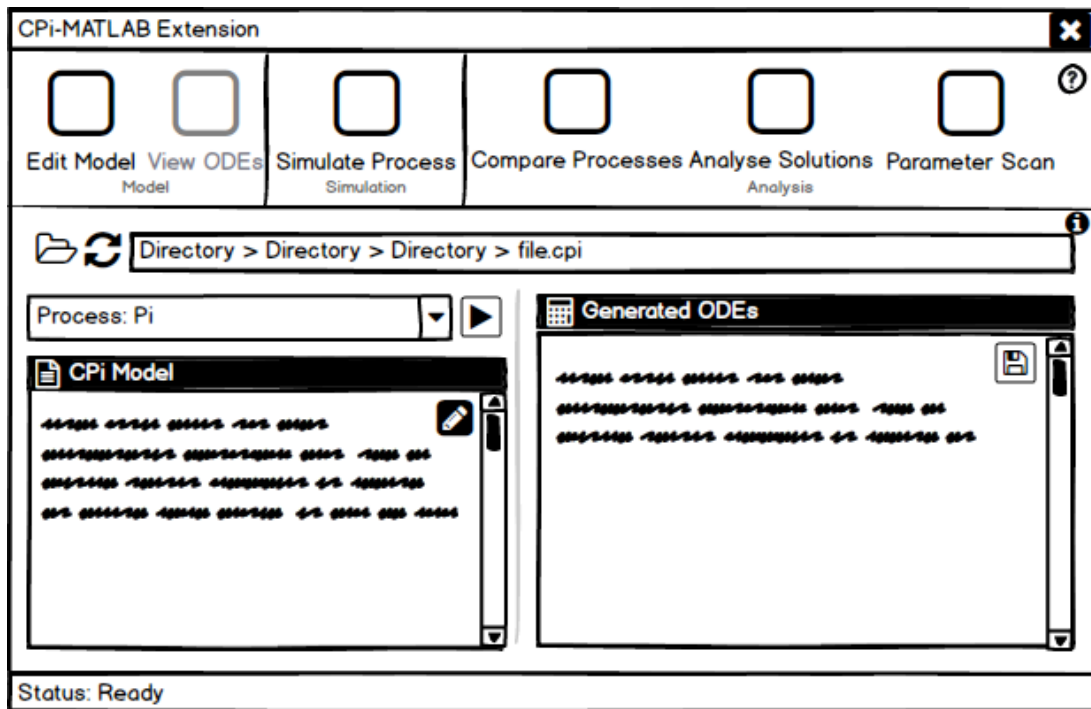


Figure B.2: The 'View ODEs' screen. A user needs to load the file like in the 'Edit Model' screen, select the process from the drop down list and then click on the 'play' icon to calculate the ODEs. The ODEs are presented in the 'Generated ODEs' text area. The save button saves the ODEs to a text file.

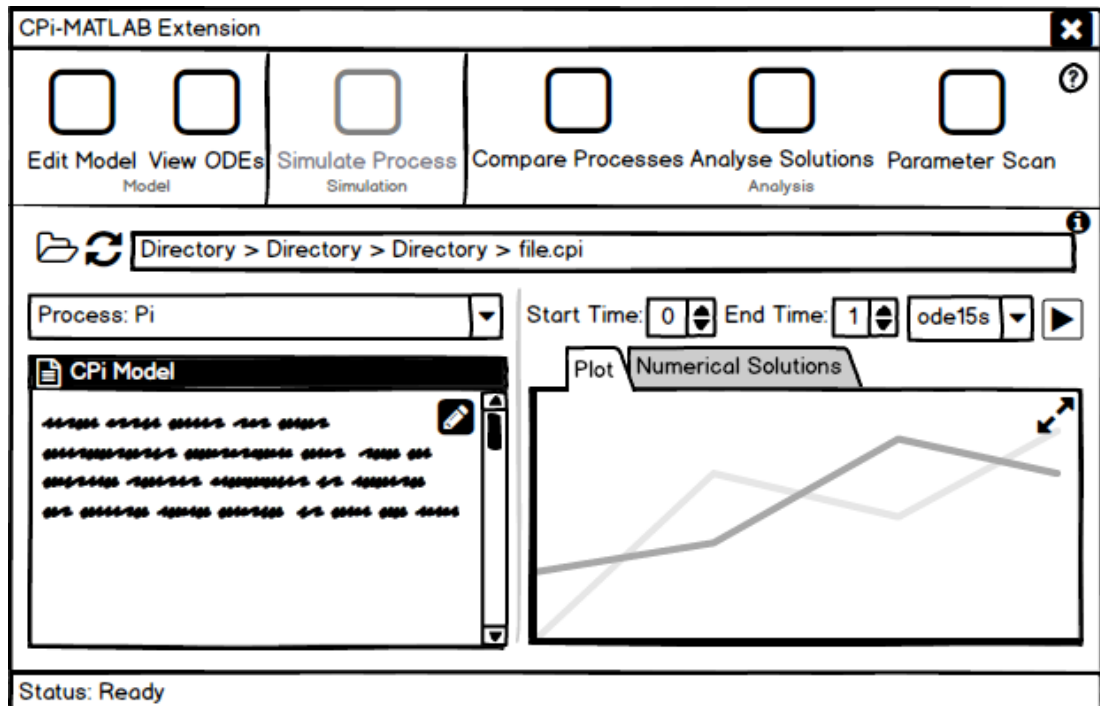


Figure B.3: The 'Simulate Process' screen. A user needs to open the CPi Model file as in the previous screens, unless the user has already selected one while using a previous screen. The definition is shown in the CPi Defn. section. The user then selects the process to calculate from the drop down, and the right hand side of the screen becomes activated. The user can modify the start and end time (which are set to 0 and 1 respectively by default). The user can also select a different solver from the ode15s default solver by using the drop down list. The user then needs to click the second play button to generate the plot and the numerical solutions in the secondary tab. The arrows icon on the plot can be used to view the plot in a maximised, separate window - the default MATLAB [1] plot viewer, which is currently used in the CLI of CPiMATLAB [3]

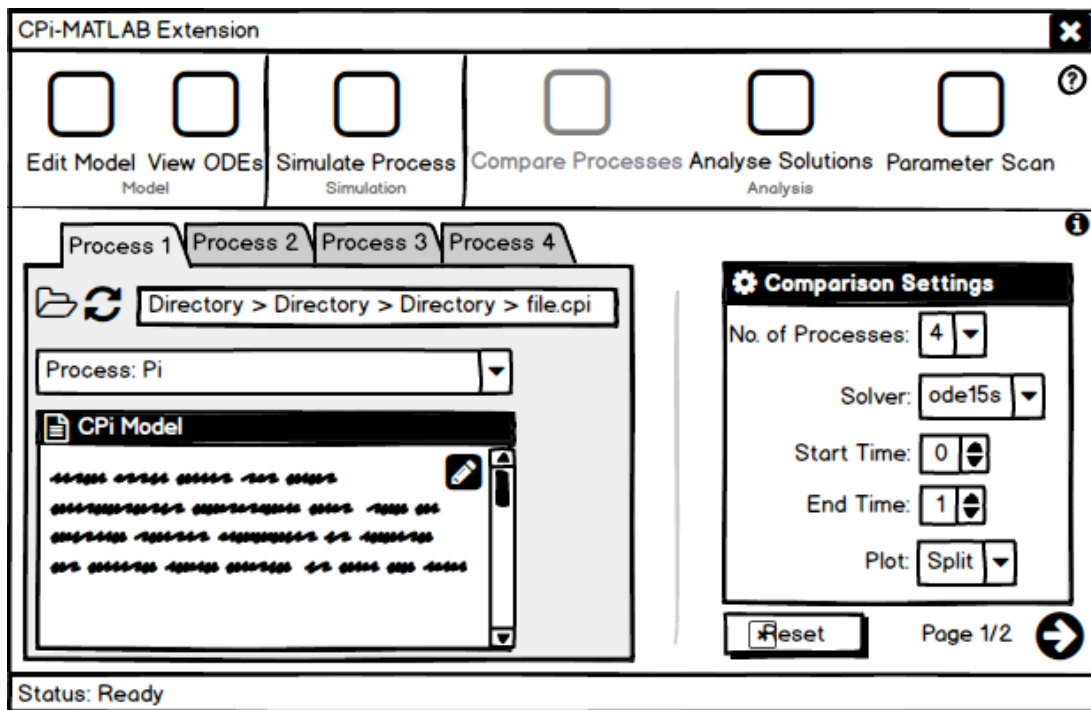


Figure B.4: The first 'Compare Processes' screen. The user needs to select the number of processes from the Comparison Settings combo box, which is set to 2 by default. This changes the number of tabs visible. The user needs to select the CPi file, and then modify the other settings if required. The arrow in the lower right corner takes the user to the plots screen.

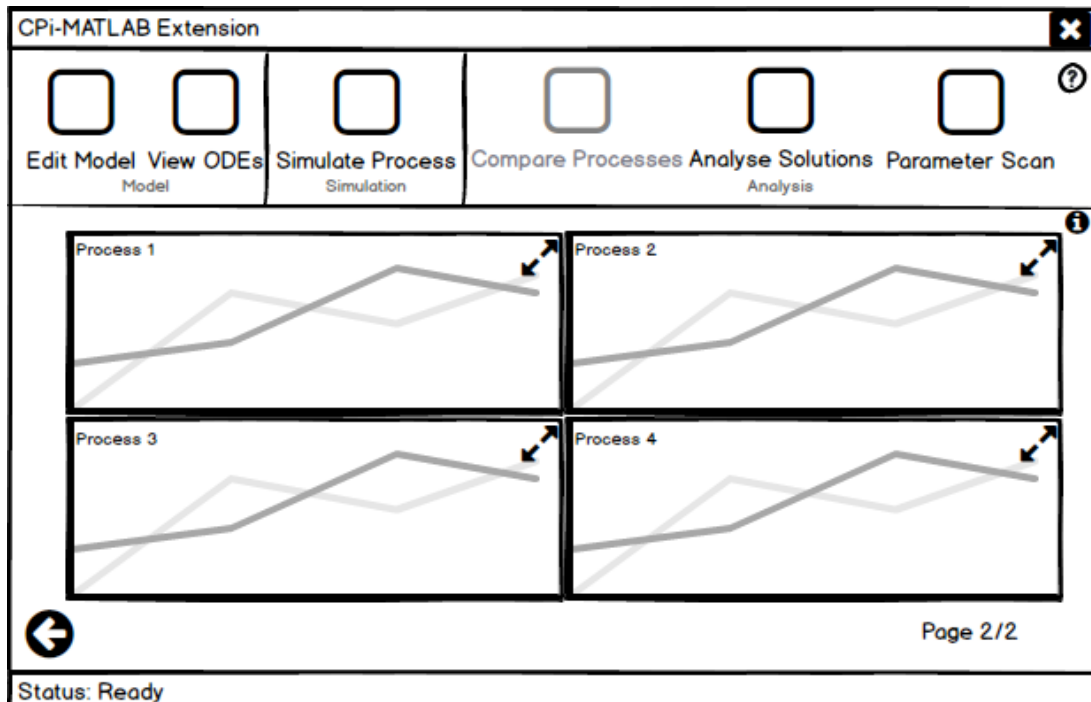


Figure B.5: The second 'Compare Processes' screen. It shows the plots generated from the previous screen.

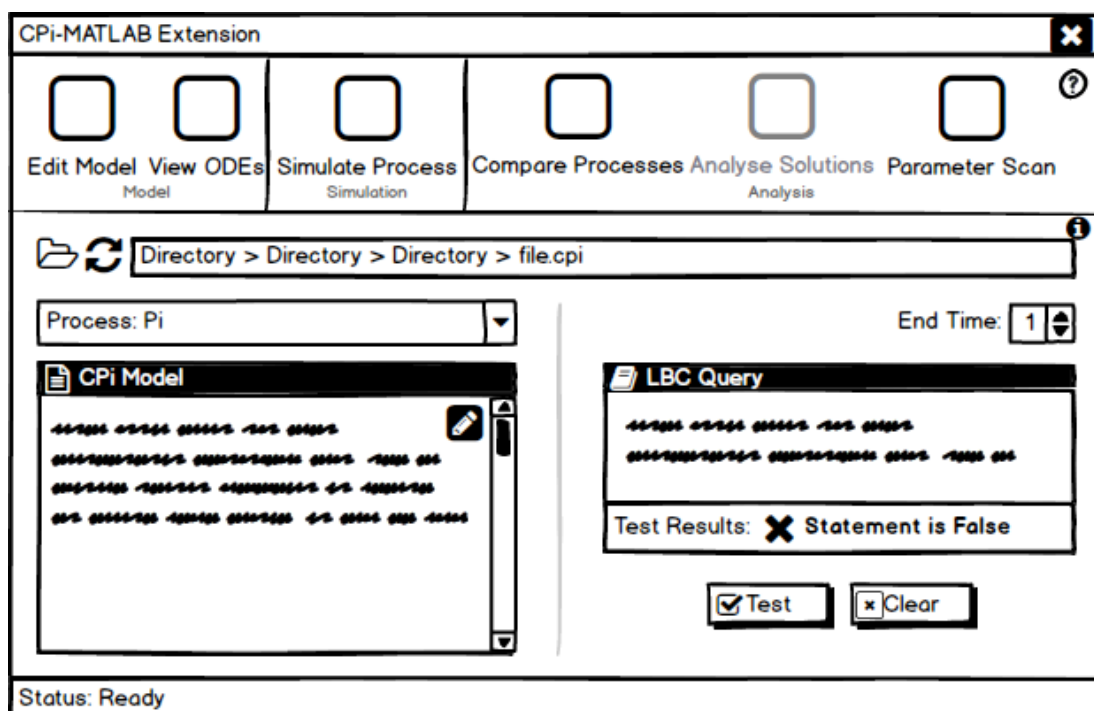


Figure B.6: The 'Analyse Solutions' screen. A user first needs to select the model to use from the top file bar. The end time is set to 1 by default, but the user can change this to any value as allowed by CPI-MATLAB. LBC [85] logic statements can then be inserted in the LBC Query text box. The user can check whether they are true or false by clicking on the 'Test' button.

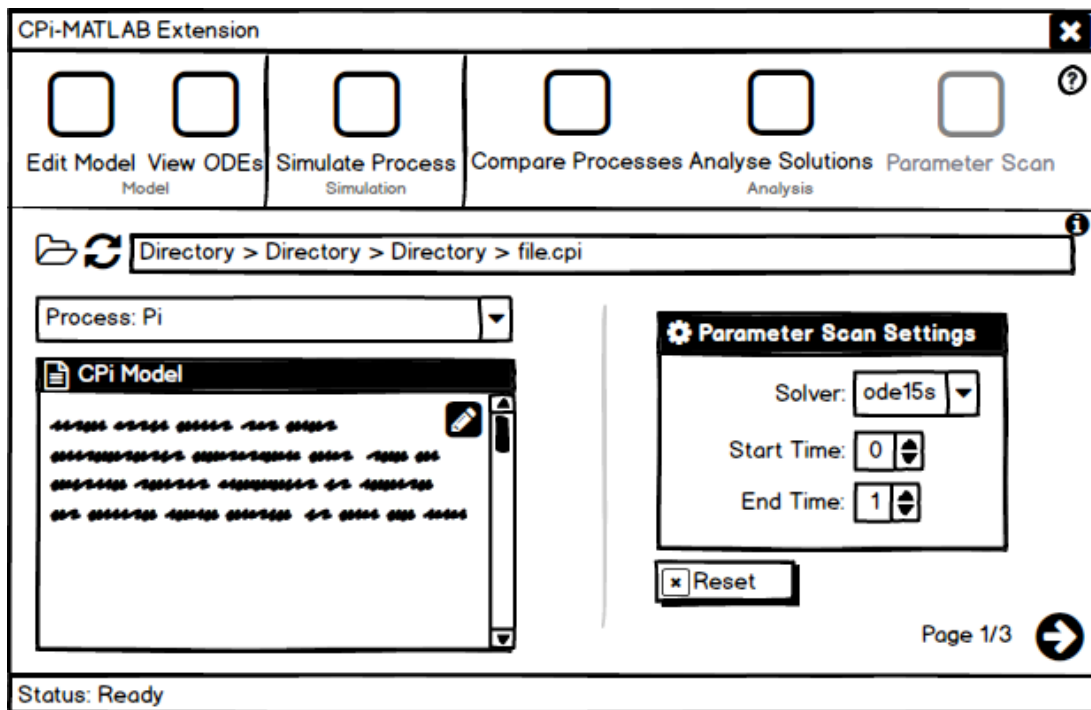


Figure B.7: The first 'Parameter Scan' screen. A user needs to select the CPi File as in previous screens, and modify parameter scan settings if this is required. The user needs to click the right arrow in the lower right corner to proceed to the next screen.

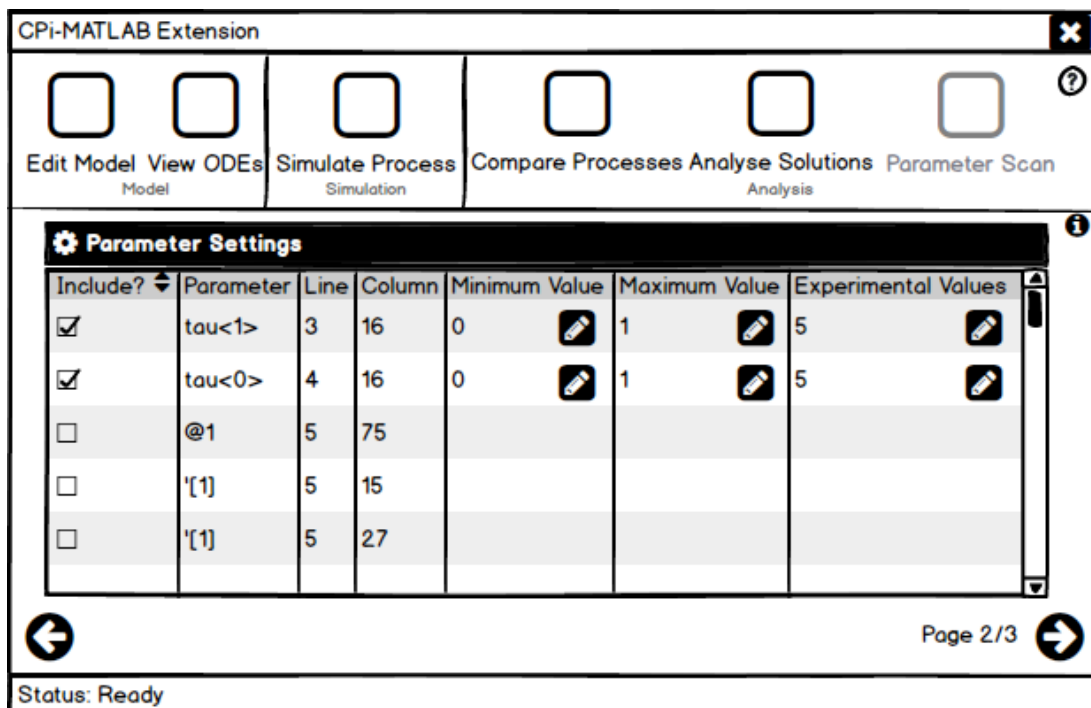


Figure B.8: The second 'Parameter Scan' screen. The user needs to select the parameters to include, and then insert the minimum, maximum and number of experimental values.

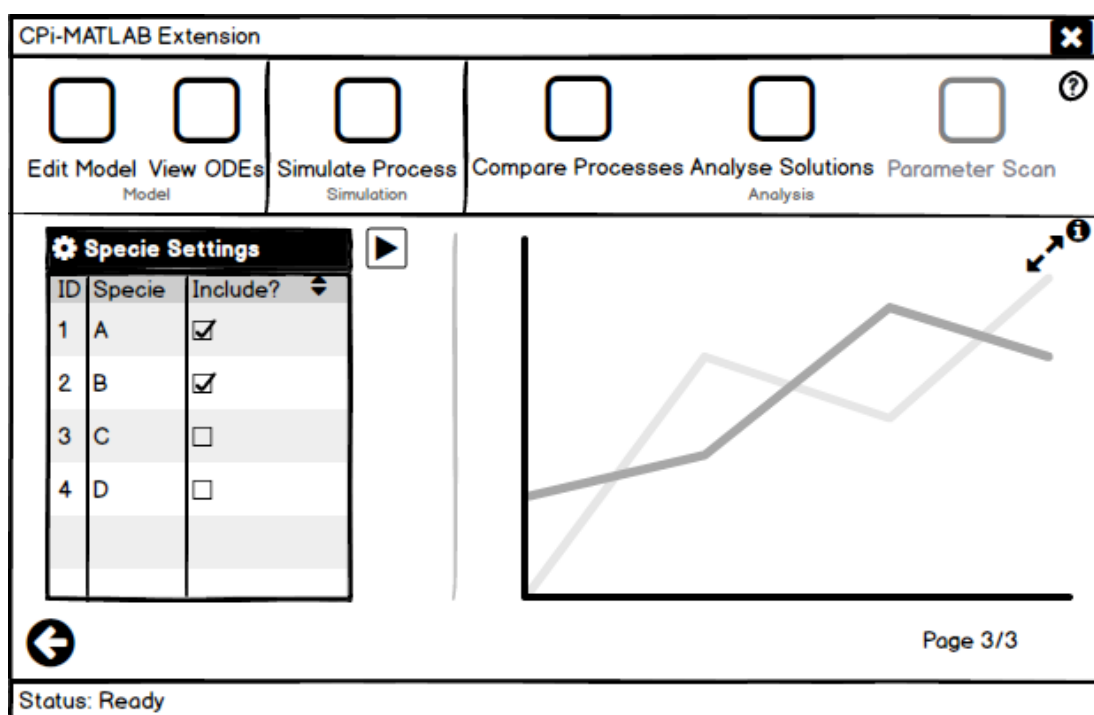


Figure B.9: The third 'Parameter Scan' screen. The user needs to select which species are to be included on the scan, and then one needs to select the play button to perform the parameter scan.

Appendix C

Sample Consent Forms

These consent forms were inspired from an existing one [86]. Figure C.1 shows the consent form used in the Pre-Implementation User Testing. Figure C.2 is the one used in Post-Implementation User Testing.

CPI-MATLAB GUI Design Test CONSENT FORM

EXPERIMENT PURPOSE & PROCEDURE

The purpose of this experiment is understand the attitudes of System Biologists or Bioinformaticians when using Computer Software, and to verify how usable these users find the Graphical User Interface (GUI) designed for CPI-MATLAB. This is part of a Masters degree project at the University of Edinburgh.

The experiment consists of two parts as follows, during which you will be asked to answer a number of questions and then to provide your opinion about a number of designs of the GUI for CPI-MATLAB.

Please note that none of the tasks is a test of your personal intelligence or ability. The objective is to test the usability of this Graphical User Interface.

CONFIDENTIALITY

The following data will be recorded:

- Audio Recording of the entire testing session
- Replies to interview questions
- Description of behaviour and reactions when analysing the mock up designs

All data will be coded so that your anonymity will be protected in any research papers and presentations that result from this work.

RECORD OF CONSENT

Your signature below indicates that you have understood the information about the CPI-MATLAB GUI Design Test experiment and consent to your participation. The participation is voluntary and you may refuse to answer certain questions on the interview and withdraw from the study at any time with no penalty. This does not waive your legal rights. You should have received a copy of the consent form for your own record. If you have further questions related to this research, please contact the researcher.

Participant

Researcher

Date

Date

Figure C.1: The Pre-Implementation Consent Form.

CPI-MATLAB GUI Test CONSENT FORM

EXPERIMENT PURPOSE & PROCEDURE

The purpose of this experiment is understand the attitudes of System Biologists or Bioinformaticians when using Computer Software, and to verify how usable these users find the Graphical User Interface (GUI) designed for CPI-MATLAB. This is part of a Masters degree project at the University of Edinburgh.

The experiment consists of two parts as follows, during which you will be asked to answer a number of questions and then to use the newly designed GUI for CPI-MATLAB.

Please note that none of the tasks is a test of your personal intelligence or ability. The objective is to test the usability of this Graphical User Interface.

CONFIDENTIALITY

The following data will be recorded:

- Audio Recording of the entire testing session
- Replies to interview questions
- Description of behaviour and reactions when using the GUI to perform the tasks

All data will be coded so that your anonymity will be protected in any research papers and presentations that result from this work.

RECORD OF CONSENT

Your signature below indicates that you have understood the information about the CPI-MATLAB GUI Test experiment and consent to your participation. The participation is voluntary and you may refuse to answer certain questions on the interview and withdraw from the study at any time with no penalty. This does not waive your legal rights. You should have received a copy of the consent form for your own record. If you have further questions related to this research, please contact the researcher.

Participant

Researcher

Date

Date

Figure C.2: The Post-Implementation Consent Form.

Appendix D

Graphical User Interface Screen-shots

We implemented five out of the planned six screens, due to time restraints. Screen-shots of each screen, with sample model `abcd.cpi` loaded, are shown below. The "Parameter Scanning" button is disabled as this functionality was not fully implemented; further details can be found in Chapter 7.

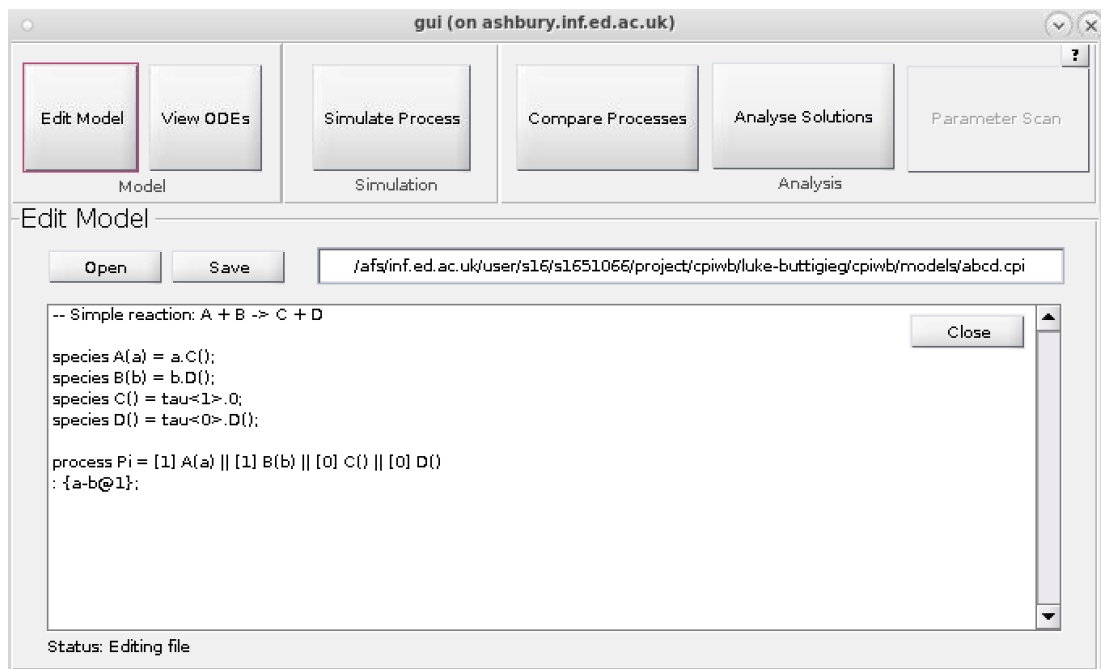


Figure D.1: The "Edit Model" screen, which allows users to edit the CPi Model Definition file.

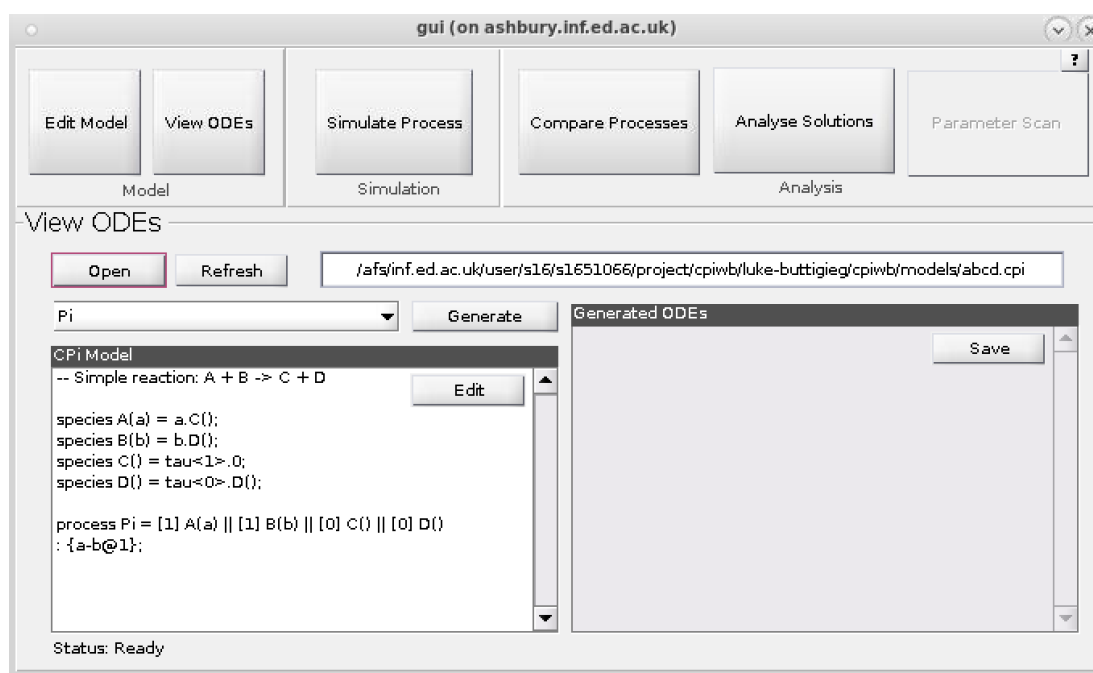
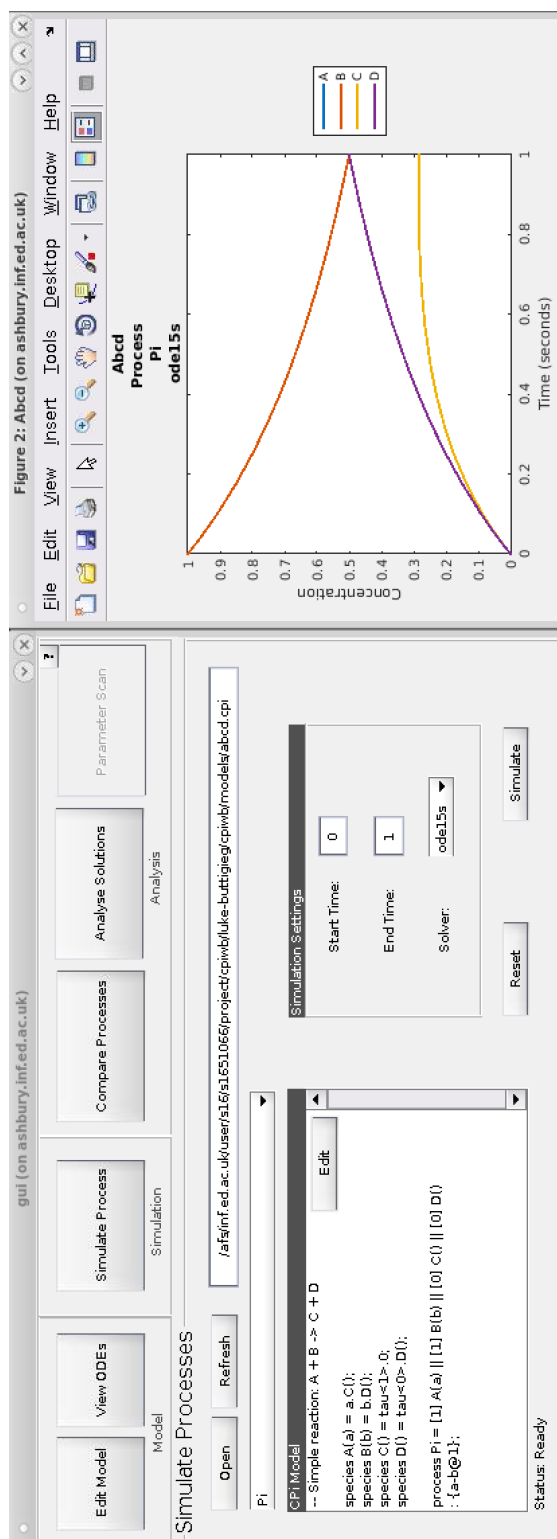


Figure D.2: The "View ODEs" screen, through which users can view the ODEs generated from the CPi Model Definition file.



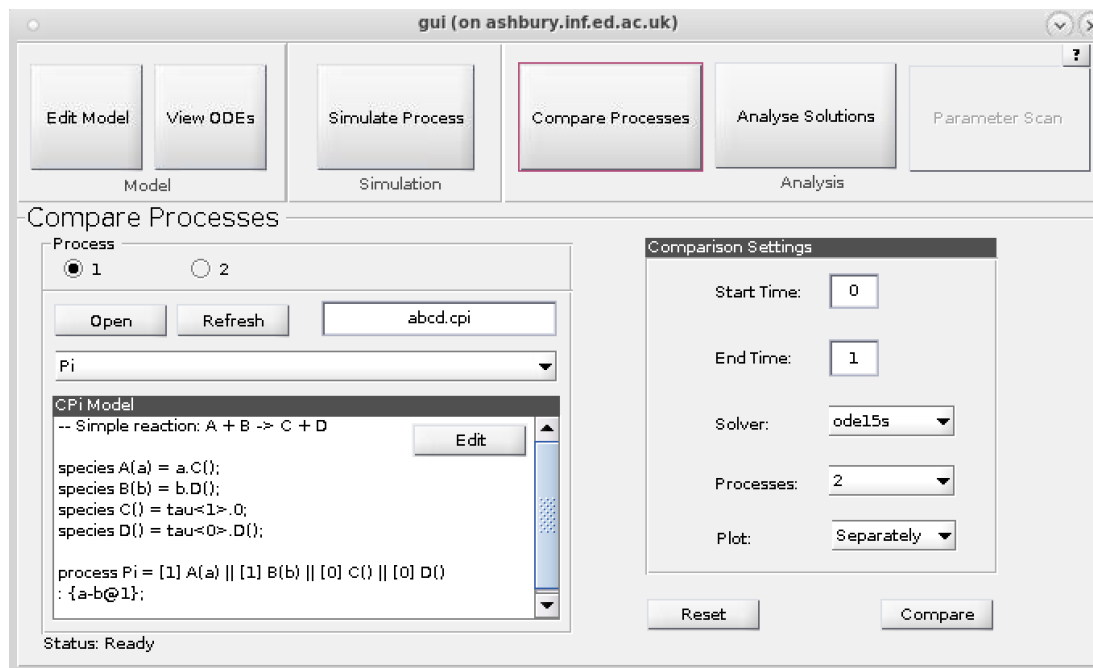


Figure D.4: The "Compare Processes" screen, which allows users to compare up to four processes together. Users can plot graphs of the processes; either on separate figures, or altogether on one figure.

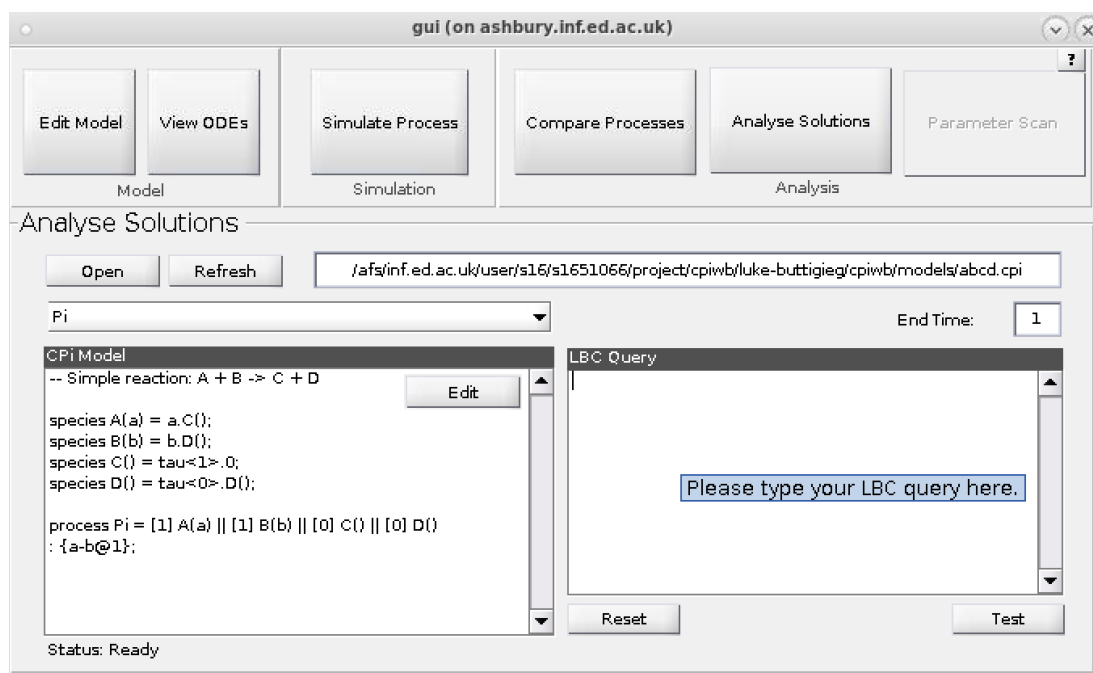


Figure D.5: The "Analyse Solutions" screen, displaying contextual help information. This screen allows users to test whether an LBC [85] query is true or false for the CPi Model Definition.

Appendix E

User Testing Plans

These two documents were used to guide the pre- and post-implementation user testing. The "Interview" section is the same in both plans. Further information can be found in Section 6.2 and 7.2–3.

Pre-Implementation User Test

- Thank you for meeting me. Can stop at any time. Anonymised. Is it OK if I record the session?
- Consent form and signature

Personal Questions

- Age group (12-17, 18-24, 25-34, 35-44, 45-54, 55-64, 65+)
- Level of current study _____
- Gender (M/F)
- Area of study (Open) -

Computer Skills

- What in your opinion, is your computer skills expertise from 0 to 5, where 0 is a person who is unable to use a computer, 1 is someone with basic knowledge, 2 is a novice with limited experience, 3 is intermediate, 4 is advanced, and 5 is someone who is an expert?
- How many years of experience do you have in Systems Biology/Bioinformatics?

Less than 1 year 1-5 Years 6-10 Years More than 10 Years

Interview

- When using bioinformatics software, do you prefer GUI or CLI? Why?
- Most frustrating experience using bioinformatics software, and why?
- Most enjoyable experience using bioinformatics software, and why?

Expectations of ease of use of systems biology/bioinformatics applications

State (VT, T, N, U, VU) for each of these statements.

- I often consult user manuals when using bioinformatics applications.
- I normally consult user manuals before starting to use an application.
- I expect myself to know how to use Systems Biology/Bioinformatics software out of the box.
- I only consult a user manual when I run into a problem.
- I rely on contextual help, such as tooltips or overlays to understand how to use Systems Biology/Bioinformatics software.
- I frequently need to ask peers for assistance using Systems Biology/Bioinformatics software, whether in person or online.

Other Biology Process modelling applications

- Are you familiar with other Biology Process Modelling languages or tools, such as Kappa Calculus (KaSim), Microsoft Stochastic Pi Machine (Visual SPiM), Intrinsic Noise Analyser, MATLAB SimBio?
 - Have you ever used them?
 - Have you ever used any tools built for them?
 - What was your experience like? (Open)

This testing is taking place prior to implementation, to the intuition obtained using your existing knowledge of systems biology/bioinformatics/computer software.

Mock-Up Design Opinion

CPI-MATLAB GUI is being developed to make CPI-WB easier to use. It allows a user to edit a CPI Model definition, view ODEs, simulate processes, compare processes, analyse solutions, perform a parameter scan. All are organised in separate screens. I'm going to present the mock ups and repeat the process for each one to understand your reaction.

I need you to be completely honest when you answer the following questions.

1) Edit Model

Used to edit the text file defining the CPI model. Here is the mockup.

- Initial reaction to mockup? Is this what you expected to see?

Scenarios

- How would you do load a file? Describe it.
- How would you edit it?
- How would you save it?
- What do you think that 'i' icon is?
- What do you think that '?' icon is?
- What do you think the (file) bar represents?
- How would you navigate to the next screen?

General Questions

- Is there anything you would change? Anything seems out of place?
- Anything you really like?

2) View ODEs

Used to check which ODEs are created from each model. Here is the mockup.

- Initial reaction to mockup? Is this what you expected to see?

Scenarios

- How would you do load a file? Describe it.
- Can you think of how you would obtain the ODEs?

General Questions

- Is there anything you would change? Anything seems out of place?
- Anything you really like?

3) Simulate Process

Used to capture settings from a user and generate a plot of the model's ODEs. Here is the mockup.

- Initial reaction to mockup? Is this what you expected to see?

Scenarios

- Which part of the screen do you think you need to use first?
- How would you load a file?
- What do you think this pencil icon does?
- How would you generate a plot?

General Questions

- Would you prefer having default values inserted, or none at all?

- Is there anything you would change? Anything seems out of place?
- Anything you really like?

4) Compare Processes

Used to compare different CPi Models to each other, generating plots. Here is the mockup.

- Initial reaction to mockup? Is this what you expected to see?

Scenarios

- How would you select the CPi file for the four different files?
- How would you proceed to the next screen?

General Questions

- Is there anything you would change? Anything seems out of place?
- Anything you really like?

5) Analyse Solutions

Used to run queries on the model. Here is the mockup.

- Initial reaction to mockup? Is this what you expected to see?

Scenarios

- Can you easily see the test result?
- What do you think the refresh button does?

General Questions

- Is there anything you would change? Anything seems out of place?
- Anything you really like?

6) Parameter Scan

Used to check how parameter values impact the model. Here is the mockup.

- Initial reaction to mockup? Is this what you expected to see?

Scenarios

- How many internal pages do you think this section has?
- How would you use the second page?
- How would you use the third page?

General Questions

- Is there anything you would change? Anything seems out of place?
- Anything you really like?

Thank you for your time!

Is it OK if you participate in a second session, in which you would use the implemented version?

Post-Implementation User Test

- Thank you for meeting me. Can stop at any time. Anonymised. Is it OK if I record the session?
- Consent form and signature

Personal Questions

- Age group (12-17, 18-24, 25-34, 35-44, 45-54, 55-64, 65+)
- Level of current study _____
- Gender (M/F)
- Area of study (Open) -

Computer Skills

- What in your opinion, is your computer skills expertise from 0 to 5, where 0 is a person who is unable to use a computer, 1 is someone with basic knowledge, 2 is a novice with limited experience, 3 is intermediate, 4 is advanced, and 5 is someone who is an expert?
- How many years of experience do you have in Systems Biology/Bioinformatics?

Less than 1 year 1-5 Years 6-10 Years More than 10 Years

Interview

- When using bioinformatics software, do you prefer GUI or CLI? Why?
- Most frustrating experience using bioinformatics software, and why?
- Most enjoyable experience using bioinformatics software, and why?

Expectations of ease of use of systems biology/bioinformatics applications

State (VT, T, N, U, VU) for each of these statements.

- I often consult user manuals when using bioinformatics applications.
- I normally consult user manuals before starting to use an application.
- I expect myself to know how to use Systems Biology/Bioinformatics software out of the box.
- I only consult a user manual when I run into a problem.
- I rely on contextual help, such as tooltips or overlays to understand how to use Systems Biology/Bioinformatics software.
- I frequently need to ask peers for assistance using Systems Biology/Bioinformatics software, whether in person or online.

Other Biology Process modelling applications

- Are you familiar with other Biology Process Modelling languages or tools, such as Kappa Calculus (KaSim), Microsoft Stochastic Pi Machine (Visual SPiM), Intrinsic Noise Analyser, MATLAB SimBio?
 - Have you ever used them?
 - Have you ever used any tools built for them?
 - What was your experience like? (Open)

This testing is taking place after implementation, to gauge your intuition using your existing knowledge of systems biology/bioinformatics/computer software.

Think-Aloud Protocol

The only background knowledge which would be useful to know is the following: Continuous Pi (CPi) is a biochemical process algebra which provides a framework for expressing reactions in terms of algebraic definitions. An algebraic definition has two components - 'species' which define the individual molecules, and 'processes' which define the overall system. A CPi model is usually defined in a text file, which is then converted to first-order ordinary differential equations (ODEs). These ODEs can be solved to study the continuous temporal fluctuations of species concentration and therefore could lead to better understanding of the system's behaviour.

CPi-MATLAB GUI is being developed to make CPi easier to use. It allows a user to edit a CPi Model definition, view ODEs, simulate processes, compare processes, analyse solutions, perform a parameter scan. All are organised in separate screens. The first three screens will be tested using this test.

Think-Aloud Training

I'm going to ask you to perform a task, and I need you to speak your mind as you think your way through it.

I need you to speak your mind out loud when answering my questions of how you would use the screen. As an example, I'll describe how many windows are in my mother's house.

Now you do so.

If at any point I hear you go quiet, I'll just tell you 'keep talking' to remind you to talk. I can't answer any questions during the session, but I'll happily answer them at the end.

Task 1

Open the kai+jpto-eq.cpi definitions file and generate the ODEs for the Kai, and save the file. Now generate them for the KaiPTO process. Finally, edit the CPI definition file by inserting 'test' on the very last line, and verifying that the changes have been written to disk.

Task 2

Open the abcd.cpi definitions file and simulate the process using the ode15s solver, and with an end time of 2. What is the value of B at time 1? Save the graph to the desktop. Now simulate the process using the ode23t solver. What is the value of B at time 1? Save the graph to the desktop.

Appendix F

GUI Support Documentation

CPiME Support Documentation

Luke Paul Buttigieg

August 2017

Contents

1	Introduction	2
2	Background Reading	2
3	Downloading CPiME	2
4	Command Line Interface	2
5	Graphical User Interface	3
5.1	Opening the Application	3
5.2	GUI Layout	4
5.3	Editing CPi Model Definition Files	4
5.4	Viewing ODEs	7
5.5	Simulating a Process	8
5.6	Comparing Processes	10
5.7	Analysing Solutions	12
6	Bibliography	15

1 Introduction

Welcome to the Continuous Pi-Calculus [1] MATLAB [2] Extension [3] (henceforth CPiME) Support Documentation. This document primarily describes how to use the functionality found in the Command Line Interface (CLI) and Graphical User Interface (henceforth GUI).

2 Background Reading

We have included a list of Continuous Pi-Calculus papers which you can consult if you need to understand the syntax and semantics of Continuous Pi-Calculus, or need further information about CPiME [1][4][5][6][3].

3 Downloading CPiME

You can download the CPiME Source Code directly from GitHub [7] from <https://github.com/continuouspi/cpiwb>. The extension source code is found in the `matlab_extension` directory. You need to be in this directory in MATLAB to use any of the CPiME functionality.

4 Command Line Interface

The CLI offers quick, direct access to the CPiME functionality. You can load an interactive interface by typing `cpime` in the MATLAB Command Window, as seen in Figure 1. Additional CLI-specific help can be accessed by typing `help`, including a list of available functions.

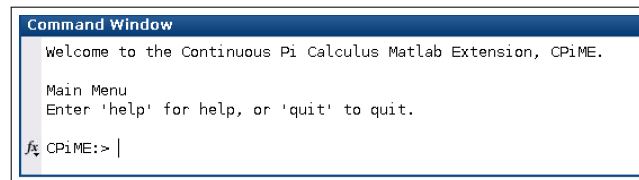
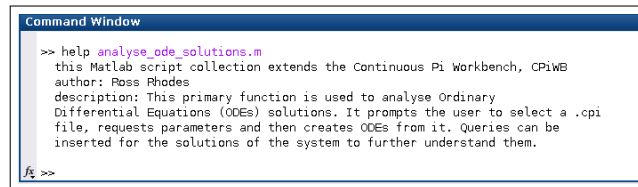


Figure 1: The CPiME [3] interactive CLI.

You can also call nine function files directly, bypassing the interactive CLI and integrating them in data processing pipelines. For example, to call the `analyse_ode_solutions` function, type `analyse_ode_solutions` in the MATLAB Command Window. All the callable files are found in the `matlab_extension` directory and end with a `.m` file extension. There is one exception: the `gui.m` file should **not** be used for direct access, as this file is used by the GUI. Files inside the `private` directory are **not** intended for direct calling.

You can access specific CPiME function file support documentation using the standard MATLAB CLI support documentation function, `help` [8]. An example is shown in Figure 2.



```

Command Window

>> help analyse_ode_solutions.m
this Matlab script collection extends the Continuous Pi Workbench, CPiWB
author: Ross Rhodes
description: This primary function is used to analyse Ordinary
Differential Equations (ODEs) solutions. It prompts the user to select a .cpi
file, requests parameters and then creates ODEs from it. Queries can be
inserted for the solutions of the system to further understand them.

f8 >>
  
```

Figure 2: Using the inbuilt MATLAB `help` [8] functionality to return information about the `analyse_ode_solutions` function.

5 Graphical User Interface

The GUI offers a visual way of using the CPiME functionality and could help newer or less technical users. Nearly all the CLI functionality is available through the GUI.

Contextual Help Information [9] is available for all GUI elements. Simply hover over a GUI element to see the associated tool-tip, as in Figure 6.

5.1 Opening the Application

You can launch the GUI by typing `gui` in the MATLAB Command Window. The default landing page is the "Edit Model" screen, as seen in Figure 2.

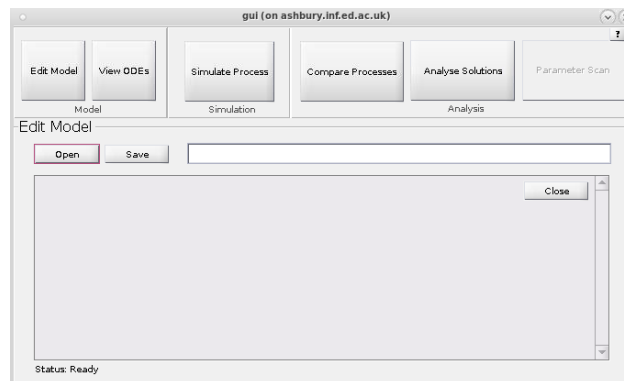


Figure 3: The default landing page is the "Edit Model" screen.

5.2 GUI Layout

You can access different screens by clicking on the different menu buttons along the top, such as "Edit Model" and "View ODEs". The current screen title can be seen in the upper left corner, such as "Edit Model" in Figure 3. Some screens have an "Open" button which is used to open files and a "Refresh" button which is used to refresh the currently loaded file. The "Edit Model" screen has a "Close" button, which is used to close the currently open file.

There is a status string in the bottom left which updates depending on how CPiME is being used.

5.3 Editing CPi Model Definition Files

CPiME offers a text editor which can be used to edit CPi Model Definition files. Currently, new model definition files cannot be created using CPiME.

1. Ensure you are on the "Edit Model" screen by clicking on the "Edit Model" menu button. The GUI should look like Figure 3.
2. Click on the "Open" button to display the "File Selector" Dialog Box, as seen in Figure 4. The dialog only shows files with the `.cpi` file extension by default.
3. Click on the file you want to open to select it, and then click "Open".
4. The file contents will be displayed in the "Edit Model" window, as seen in Figure 5. The file path will be displayed in the file path bar.
5. You can write the changes to disk by clicking the "Save" button. The status string will update to "Ready" when the files contents have been written to disk, as in Figure 6.
6. Click on the "Close" button to close the open file. You will see a dialog box to confirm whether you want to close the file, as in Figure 7. The screen will look like Figure 3 again if you click "Yes" on the dialog box.

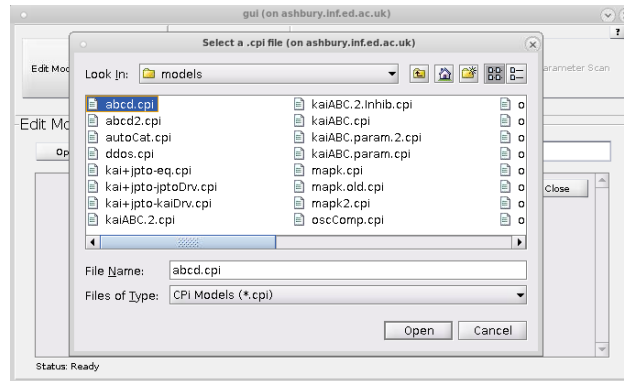


Figure 4: After clicking "Open", select a file to open in "Edit Model" screen.

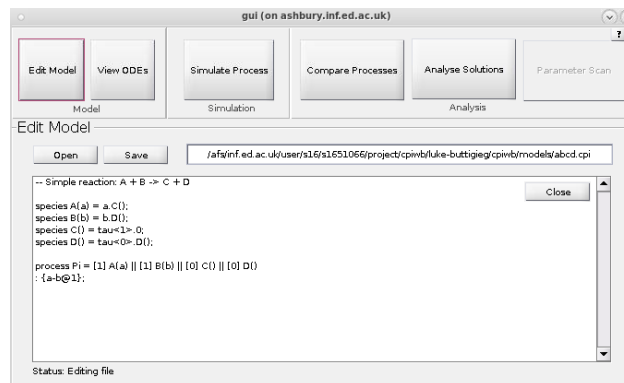


Figure 5: Editing a model definition file's contents in the "Edit Model" screen.

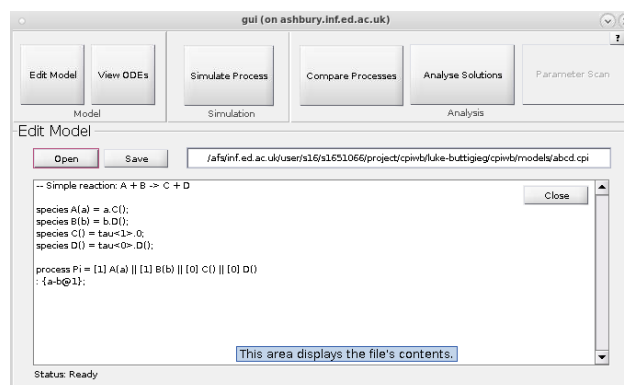


Figure 6: Click on the "Save" button to write any changes to disk. The status string will revert back to "Ready" on the "Edit Model" screen.

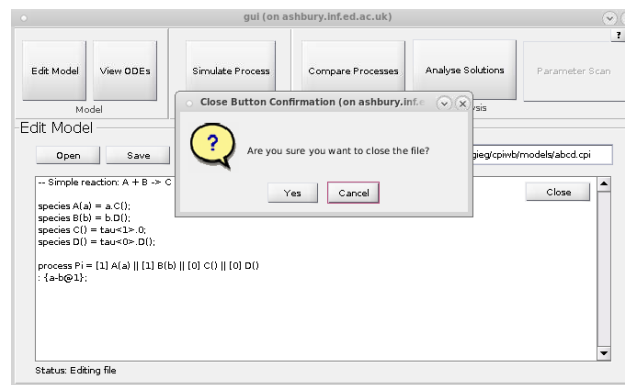


Figure 7: Close a file by clicking on the "Close" button. A confirmation dialog box will be displayed.

5.4 Viewing ODEs

You can view the first-order ordinary differential equations (henceforth ODE(s)) which CPiME is using in the "View ODEs" screen.

1. Navigate to the "View ODEs" screen by clicking on the "View ODEs" button in the menu bar, as seen in Figure 8. Any file selection made in a previous window will carry over to this screen.
2. Click on the "Open" button to display the "File Selector" Dialog Box, similar to that seen in Figure 4. The dialog only shows files with the `.cpi` file extension by default.
3. Click on the file you want to open to select it, and then click "Open".
4. The file contents will be displayed in the "View ODEs" window, as seen in Figure 9. The file path will be displayed in the file path bar.
5. You can click on the "Edit" Button if you need to modify the CPi Model Definition file. This will take you to the "Edit Model" screen.
6. Select the process you are interested in using the drop-down list. In Figure 9, this drop-down list contains a process called "Pi".
7. Click on the "Generate" button to convert the CPi Model definition in ODEs. The status string will change to "Generating ODEs". This process could take a while, depending on the complexity of the model.
8. The ODEs are displayed in the "Generated ODEs" text box, as in Figure 10.
9. You can save the generated ODEs by clicking on the "Save" button. This will display the standard MATLAB "Save File" dialog box, which will allow you to insert a file name and choose a disk location for the file.

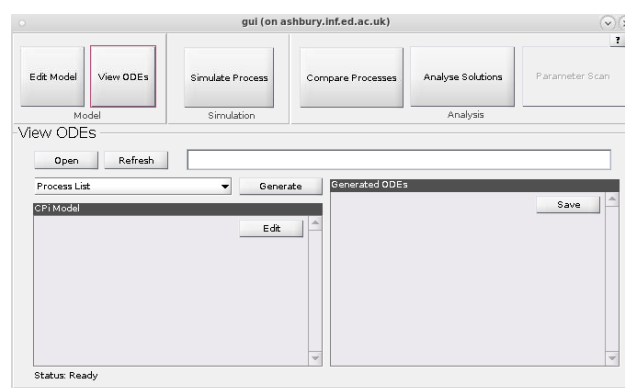


Figure 8: This is the "View ODEs" screen before a file is loaded in it.

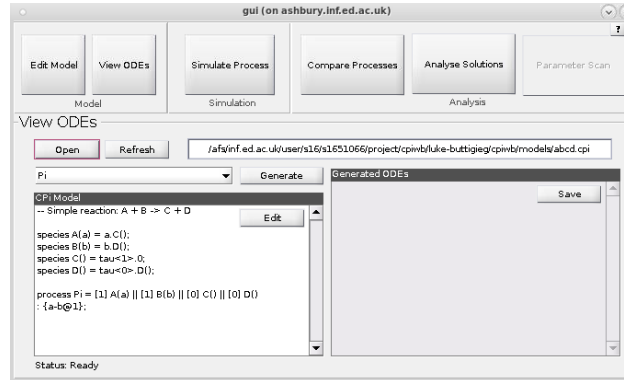


Figure 9: This is the "View ODEs" screen displaying a model definition file's contents.

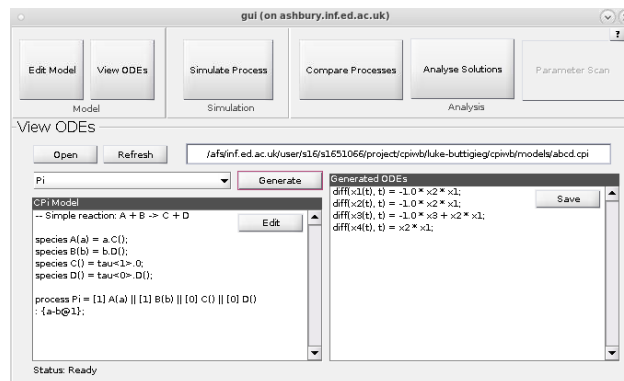


Figure 10: This is the "View ODEs" screen displaying the generated ODEs. You can save the ODEs by clicking on the "Save" button.

5.5 Simulating a Process

You can use CPiME to plot a graph of the ODEs generated from the model definition files. Numerical solutions are also displayed.

1. Navigate to the "Simulate Process" screen by clicking on the "Simulate Process" button in the menu bar, as seen in Figure 11. Any file selection made in a previous window will carry over to this screen.
2. Click on the "Open" button to display the "File Selector" Dialog Box, similar to that seen in Figure 4. The dialog only shows files with the .cpi file extension by default.
3. Click on the file you want to open to select it, and then click "Open".

4. The file contents will be displayed in the "CPi Model" text area. The file path will be displayed in the file path bar.
5. You can click on the "Edit" Button if you need to modify the CPi Model Definition file. This will take you to the "Edit Model" screen.
6. Select the process you are interested in using the drop-down list. This is similar to what is seen in Figure 9: this drop-down list contains a process called "Pi".
7. You can change the settings found in the "Simulation Settings" as required. You can select a different MATLAB ODE solver [10] and the start and end time.
8. Click on the "Simulate" button to convert the CPi Model definition in ODEs, plot a graph of them and display numerical solutions. The status string will change to "Simulating Process". This process could take a while, depending on the complexity of the model.
9. The graph plot and numerical solutions are displayed in separate windows, as in Figure 12.
10. You can reset the "Simulation Settings" to their default values by clicking on the "Reset" Button.

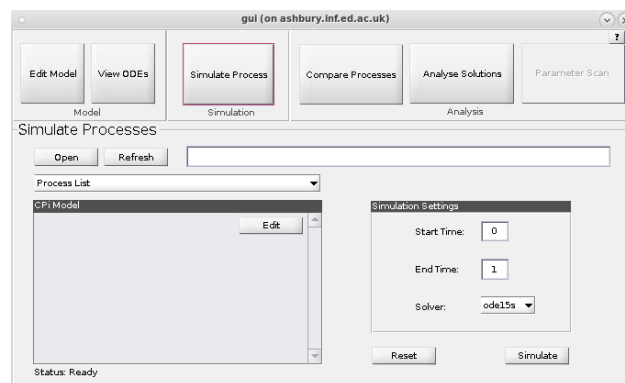


Figure 11: This is the "Simulate Process" screen before a file is loaded in it.

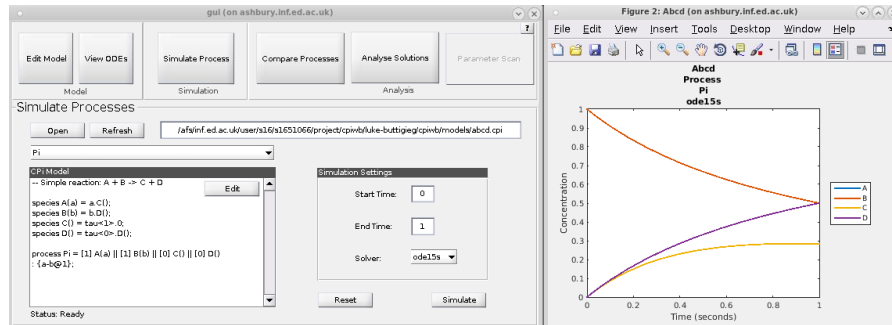


Figure 12: This is the "Simulate Process" screen displaying a model definition file's contents and a graph plot of its ODEs. The numerical solutions will be shown on a separate window.

5.6 Comparing Processes

You can use CPiME to compare up to four processes with each other by plotting graphs of the ODEs generated from the model definition files. Graphs can be plotted on the same graph figure, or on separate ones. Numerical solutions are not displayed in this case.

1. Navigate to the "Compare Processes" screen by clicking on the "Compare Processes" button in the menu bar, as seen in Figure 13. Any file selection made in a previous window will carry over to this screen and populate the first process settings.
2. Click on the "Open" button to display the "File Selector" Dialog Box, similar to that seen in Figure 4. The dialog only shows files with the .cpi file extension by default.
3. Click on the file you want to open to select it, and then click "Open".
4. The file contents will be displayed in the "CPi Model" text area. The file path will be displayed in the miniaturised file path bar.
5. You can click on the "Edit" Button if you need to modify the CPi Model Definition file. This will take you to the "Edit Model" screen.
6. Select the process you are interested in using the drop-down list. This is similar to what is seen in Figure 9: this drop-down list contains a process called "Pi".
7. Click on the "2" "Process" radio button to select the file and process for the second process you would like to compare.
8. If you want to compare more than 2 files, change the "Processes" drop-down list value in the "Comparison Settings" area to a higher number.

9. Continue selecting files until all processes are selected.
10. You can now modify the "Comparison Settings" as required. You can select a different MATLAB ODE solver [10] and the start and end time. You can also change whether to display the plotted graphs on the same Figure by selecting the "Together" option in the "Plot" drop-down list. Otherwise, display them separately by selecting the "Separately" option.
11. Click on the "Compare" button to convert the CPi Model definitions in ODEs, and plot graphs of them. The status string will change to "Comparing Processes". This process could take a while, depending on the complexity of the models.
12. The graph plots will be displayed once the process has finished executing.
13. You can clear the file selection and reset the "Comparison Settings" to their default values by clicking on the "Reset" Button.

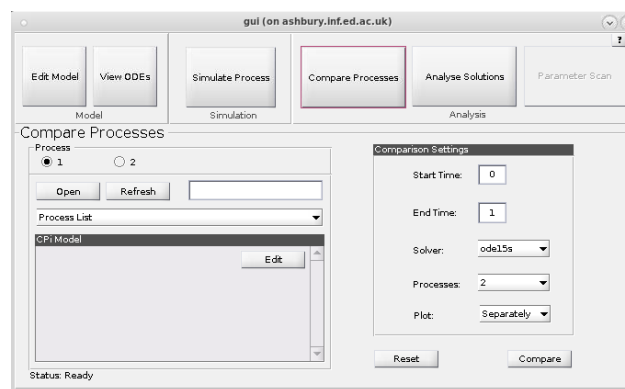


Figure 13: This is the "Compare Processes" screen before a file is loaded in it.

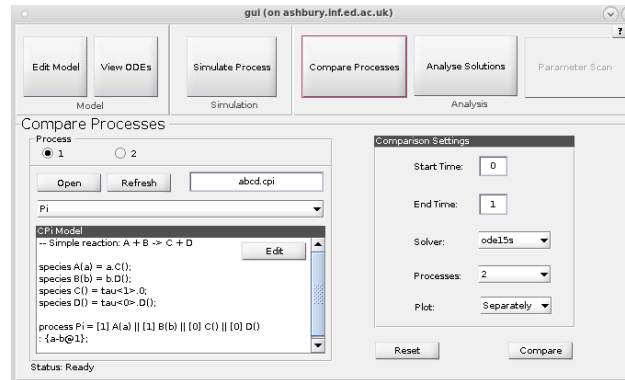


Figure 14: This is the "Compare Process" screen displaying the model definition file's contents for the first selected file. You can select a another file by using the "Process" radio buttons.

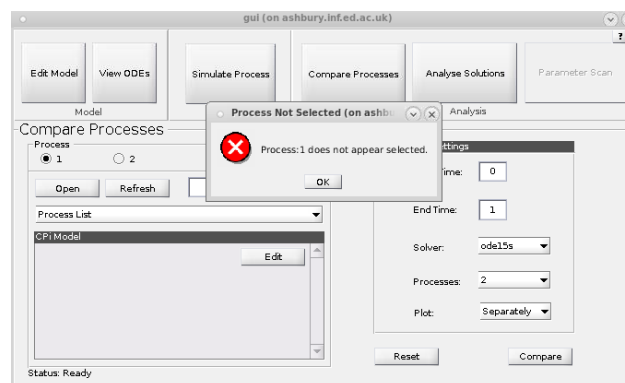


Figure 15: You will see this error if you do not select a file for all the processes you want to compare.

5.7 Analysing Solutions

You can use CPiME to check whether LBC [4] queries are true or false for the ODEs generated from the model definition files.

1. Navigate to the "Analyse Solutions" screen by clicking on the "Analyse Solutions" button in the menu bar, as seen in Figure 16. Any file selection made in a previous window will carry over to this screen and populate the first process settings.
2. Click on the "Open" button to display the "File Selector" Dialog Box, similar to that seen in Figure 4. The dialog only shows files with the .cpi file extension by default.
3. Click on the file you want to open to select it, and then click "Open".

4. The file contents will be displayed in the "CPi Model" text area. The file path will be displayed in the file path bar.
5. You can click on the "Edit" Button if you need to modify the CPi Model Definition file. This will take you to the "Edit Model" screen.
6. Select the process you are interested in using the drop-down list. This is similar to what is seen in Figure 9: this drop-down list contains a process called "Pi".
7. You can change the "End Time" to a value other than the default value of 1.
8. Type in the LBC query you would like to test, or type in `example` for a list of examples, as seen in Figure 17.
9. Click "Test" to check whether the query is true or false for the selected CPi Model Process, as seen in Figure 18.
10. You can clear the file selection and reset the "Comparison Settings" to their default values by clicking on the "Reset" Button.

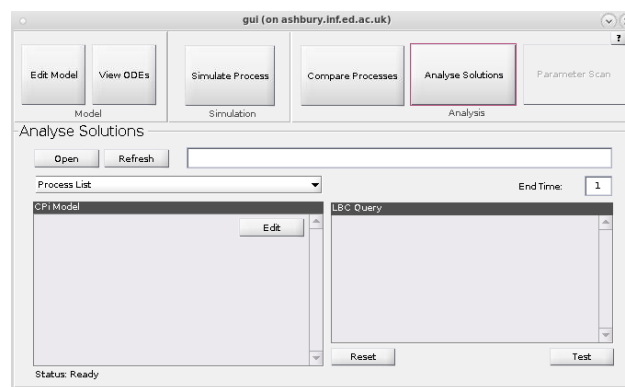


Figure 16: This is the "Analyse Solutions" screen before a file is loaded in it.

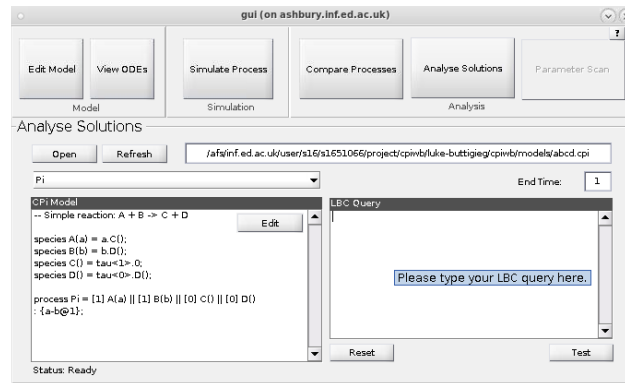


Figure 17: This is the "Analyse Solutions" screen displaying the model definition file's contents for the first selected file.

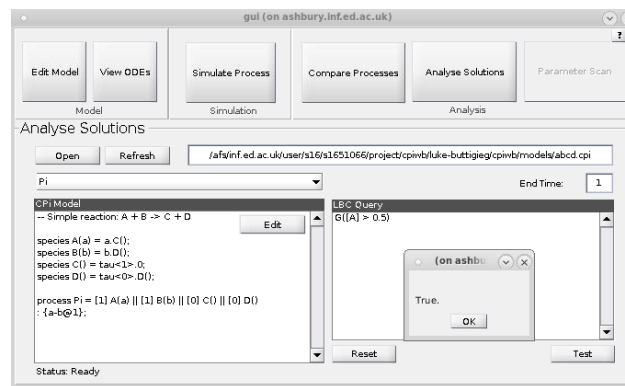


Figure 18: After typing in the LBC query, click the "Test" button to check whether the query is true or false for the selected CPI Model Process.

6 Bibliography

- [1] M. Kwiatkowski and I. Stark, “The continuous π -calculus: A process algebra for biochemical modelling,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5307 LNBI, pp. 103–122, 2008.
- [2] Mathworks Inc., *MATLAB and Statistics Toolbox Release R2016b*. Natick, Massachusetts, United States: The MathWorks, Inc., 2016.
- [3] T. R. Rhodes, “MATLAB Extension for the Continuous π -Calculus,” Ph.D. dissertation, The University of Edinburgh, 2017.
- [4] C. J. Banks and I. Stark, “A logic of behaviour in context,” *Information and Computation*, vol. 236, no. 2014, pp. 3–18, 2014. [Online]. Available: <http://dx.doi.org/10.1016/j.ic.2014.01.009>
- [5] A. McCrae, “CPi-IDE: A GUI for Continuous Pi Calculus,” Ph.D. dissertation, The University of Edinburgh, 2014.
- [6] C. Banks, “CPi Calculus Workbench GitHub Repository,” 2016. [Online]. Available: <https://github.com/chrisbanks/cpiwb>
- [7] S. F. Conservancy, “Git,” 2017. [Online]. Available: <https://git-scm.com/>
- [8] Mathworks Inc., “Add Help for Your Program - MATLAB & Simulink - MathWorks United Kingdom,” 2017. [Online]. Available: https://uk.mathworks.com/help/matlab/matlab_prog/add-help-for-your-program.html
- [9] C. Walley and B. Vedachalam, “Contextual help information,” 2002. [Online]. Available: <https://www.google.com/patents/US20020091993>
- [10] Mathworks Inc., “Choose an ODE Solver,” 2017. [Online]. Available: <https://uk.mathworks.com/help/matlab/math/choose-an-ode-solver.html>

Bibliography

- [1] Mathworks Inc., *MATLAB and Statistics Toolbox Release R2016b*. Natick, Massachusetts, United States: The MathWorks, Inc., 2016.
- [2] M. Kwiatkowski and I. Stark, “The continuous π -calculus: A process algebra for biochemical modelling,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5307 LNBI, pp. 103–122, 2008.
- [3] T. R. Rhodes, “MATLAB Extension for the Continuous π -Calculus,” Ph.D. dissertation, The University of Edinburgh, 2017.
- [4] C. Banks, “CPi Calculus Workbench GitHub Repository,” 2016. [Online]. Available: <https://github.com/chrisbanks/cpiwb>
- [5] B. Martin and B. Hanington, “Cognitive Walkthrough,” in *Universal Methods of Design*. Rockport Publishers, 2012, pp. 32–33.
- [6] —, “Heuristic Evaluation,” in *Universal Methods of Design*. Rockport Publishers, 2012, pp. 98–99.
- [7] —, “Think Aloud,” in *Universal Methods of Design*. Rockport Publishers, 2012, pp. 180–181.
- [8] —, “Interviews,” in *Universal Methods of Design*. Rockport Publishers, 2012, pp. 102–103.
- [9] —, “Personas,” in *Universal Methods of Design*. Rockport Publishers, 2012, pp. 132–133.
- [10] A. Marcus, “Metaphor design for user interfaces,” in *CHI 98 conference summary on Human factors in computing systems*. ACM, 1998, pp. 129–130.
- [11] W. Lidwell, K. Holden, and J. Butler, “Affordance,” in *Universal Principles of Design*. Rockport Publishers, 2010, pp. 22–23.
- [12] C. Walley and B. Vedachalam, “Contextual help information,” 2002. [Online]. Available: <https://www.google.com/patents/US20020091993>
- [13] W. Lidwell, K. Holden, and J. Butler, “Gutenberg Diagram,” in *Universal Principles of Design*. Rockport Publishers, 2010, pp. 118–119.

- [14] S. Coren and J. S. Girgus, "Principles of perceptual organization and spatial distortion: The gestalt illusions," *Journal of Experimental Psychology: Human Perception and Performance*, vol. 6, no. 3, pp. 404–412, 8 1980.
- [15] D. Spinellis, "An Implementation of the {H}askell Language," Imperial College, London, UK, Tech. Rep. June, 1990.
- [16] Mathworks Inc., "Choose an ODE Solver," 2017. [Online]. Available: <https://uk.mathworks.com/help/matlab/math/choose-an-ode-solver.html>
- [17] —, "Create a figure window," 2017. [Online]. Available: <https://uk.mathworks.com/help/matlab/ref/figure.html>
- [18] —, "MATLAB SimBiology," 2017. [Online]. Available: <https://uk.mathworks.com/products/simbiology.html>
- [19] IntiQuan GmbH, "IQM Tools," 2017. [Online]. Available: <http://www.intiquan.com/iqm-tools/>
- [20] Mathworks Inc., "Self-Paced Training - MATLAB and Simulink," 2017. [Online]. Available: <https://uk.mathworks.com/services/training/online.html>
- [21] D. M. Ritchie, "The Development of the C Language," *The Second ACM SIGPLAN Conference on History of Programming Languages*, vol. 28, no. 3, pp. 201–208, 1993. [Online]. Available: <http://doi.acm.org/10.1145/154766.155580%5Cnhttp://dl.acm.org/citation.cfm?doid=155360.155580>
- [22] D. Bolchini, A. Finkestein, and P. Paolini, "Designing usable bio-information architectures," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5613 LNCS, no. PART 4, pp. 653–662, 2009.
- [23] C. Macaulay, D. Sloan, X. Jiang, P. Forbes, S. Loynton, J. R. Swedlow, and P. Gregor, "Usability and user-centered design in scientific software development," *IEEE Software*, vol. 26, no. 1, pp. 96–102, 2009.
- [24] S. Morrison-Smith, C. Boucher, A. Bunt, and J. Ruiz, "Elucidating the role and use of bioinformatics software in life science research," *Proceedings of the 2015 British HCI Conference on - British HCI '15*, pp. 230–238, 2015. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2783446.2783581>
- [25] J. Barker and J. Thornton, "Software engineering challenges in bioinformatics," *Proceedings. 26th International Conference on Software Engineering*, pp. 12–15, 2004. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1317409>
- [26] D. Tran, C. Dubay, P. Gorman, and W. Hersh, "Applying task analysis to describe and facilitate bioinformatics tasks," *Studies in Health Technology and Informatics*, vol. 107, pp. 818–822, 2004.

- [27] L. Preeyanon, A. B. Pyrkosz, and C. T. Brown, "Reproducible bioinformatics research for biologists," *Implementing Reproducible Research*, p. 185, 2014.
- [28] R. T. Javafery H., Seffah A., "Making Bioinformatics Tools User-Centered," *COMMUNICATIONS OF THE ACM*, vol. 47, no. 11, pp. 59–63, 2004.
- [29] N. Al-Ageel, A. Al-Wabil, G. Badr, and N. AlOmar, "Human Factors in the Design and Evaluation of Bioinformatics Tools," *Procedia Manufacturing*, vol. 3, no. Ahfe, pp. 2003–2010, 2015. [Online]. Available: <http://dx.doi.org/10.1016/j.promfg.2015.07.247>
- [30] D. Bolchini, A. Finkelstein, V. Perrone, and S. Nagl, "Better bioinformatics through usability analysis," *Bioinformatics*, vol. 25, no. 3, pp. 406–412, 2009.
- [31] B. Mirel, "Supporting cognition in systems biology analysis: findings on users' processes and design implications." *Journal of Biomedical Discovery and Collaboration*, vol. 4, p. 2, 2009.
- [32] D. J. Mayhew, "The usability engineering lifecycle," *CHI '99 extended abstracts on Human factors in computing systems - CHI '99*, p. 147, 1999. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=632716.632805>
- [33] P. Senadheera, M. Kumarasinghe, P. Perera, J. Wattewidanage, and T. K. Weerasinghe, "Identification of User-Friendly Bioinformatics Tools for Courses in Open and Distance Learning," *IJODEL*, vol. 2, no. 2, pp. 31–38, 2016.
- [34] K. Pavelin, J. A. Cham, P. de Matos, C. Brooksbank, G. Cameron, and C. Steinbeck, "Bioinformatics meets user-centred design: A perspective," *PLoS Computational Biology*, vol. 8, no. 7, 2012.
- [35] B. Mirel and Z. Wright, "Heuristic Evaluations of Bioinformatics Tools: A Development Case," *Human-Computer Interaction, Pt I*, vol. 5610, pp. 329–338, 2009.
- [36] P. de Matos, J. A. Cham, H. Cao, R. Alcántara, F. Rowland, R. Lopez, and C. Steinbeck, "The Enzyme Portal: a case study in applying user-centred design methods in bioinformatics," *BMC Bioinformatics*, vol. 14, no. 1, p. 103, 2013. [Online]. Available: <http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-14-103>
- [37] J. Bartlett and T. Neugebauer, "Supporting Information Tasks with User-Centred System Design : The development of an interface supporting bioinformatics analysis," *Galperin*, vol. 1, no. Galperin 2006, pp. 1–12, 2006.
- [38] T. D. Erickson, "Working with interface metaphors," *Baecker et al*, vol. 11, p. 147, 2000.
- [39] M. Wertheimer, "Gestalt Theory," *A Source Book of Gestalt Psychology*, vol. 11, no. 1, pp. 1–11, 1938. [Online]. Available: <http://www.jstor.org/stable/40982002http://content.apa.org/journals/bul/22/5/261>

- [40] T. Taylor, “How to Use the Gestalt Principles for Visual Storytelling #PoDV,” 2014. [Online]. Available: <http://www.fusioncharts.com/blog/2014/03/how-to-use-the-gestalt-principles-for-visual-storytelling-podv/>
- [41] T. Yeh, T.-H. Chang, B. Xie, G. Walsh, I. Watkins, K. Wongsuphasawat, M. Huang, L. S. Davis, and B. B. Bederson, “Creating Contextual Help for GUIs Using Screenshots,” in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '11. New York, NY, USA: ACM, 2011, pp. 145–154. [Online]. Available: <http://doi.acm.org/10.1145/2047196.2047214>
- [42] J. Nielsen, “Usability inspection methods,” in *Conference companion on Human factors in computing systems*. ACM, 1994, pp. 413–414.
- [43] ———, “Ten usability heuristics,” 2005.
- [44] W. Lidwell, K. Holden, and J. Butler, “Consistency,” in *Universal Principles of Design*, 2010, pp. 56–57.
- [45] ———, “Development Cycle,” in *Universal Principles of Design*, 2010, pp. 78–79.
- [46] J. Nielson, “Trip report,” *ACM SIGCHI Bulletin*, vol. 22, no. 2, pp. 20–25, 11 1990. [Online]. Available: <http://doi.acm.org/10.1145/122475.122479>
- [47] P. Thomas, H. Matuschek, and R. Grima, “Intrinsic noise analyzer: A software package for the exploration of stochastic biochemical kinetics using the system size expansion,” *PLoS ONE*, vol. 7, no. 6, 2012.
- [48] A. Darwiche and M. Goldszmidt, “On the Relation between Kappa Calculus and Probabilistic Reasoning,” *ArXiv e-prints*, 2 2013.
- [49] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine, “Rule-Based Modelling of Cellular Signalling,” in *CONCUR 2007 – Concurrency Theory: 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007. Proceedings*, L. Caires and V. T. Vasconcelos, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 17–41. [Online]. Available: https://doi.org/10.1007/978-3-540-74407-8_3
- [50] Harvard Medical School, “KaSim of The Kappa Language,” 2017. [Online]. Available: <http://dev.executableknowledge.org>
- [51] W. Lidwell, K. Holden, and J. Butler, “Alignment,” in *Universal Principles of Design*. Rockport Publishers, 2010, pp. 24–25.
- [52] S. Teller, *Data Visualization with D3.js*. Packt Publishing, 2013.
- [53] B. Eich, “JavaScript at ten years,” *ACM SIGPLAN Notices*, vol. 40, no. 9, p. 129, 2005.
- [54] Microsoft Corp, “Visual Stochastic Pi Machine,” 2008. [Online]. Available: <https://www.microsoft.com/en-us/research/project/stochastic-pi-machine/>

- [55] L. Phillips Andrew Cardelli, G. Castagna, and G. Castagna, “A graphical representation for biological processes in the stochastic pi-calculus,” *Lecture Notes in Computer Science*, vol. 4230, pp. 123–152, 2006.
- [56] L. Moroney, *Introducing Microsoft Silverlight: PRO-Developer*. Microsoft Press, 2007.
- [57] W. Lidwell, K. Holden, and J. Butler, “Hick’s Law,” in *Universal Principles of Design*. Rockport Publishers, 2010, pp. 120–121.
- [58] Microsoft Corp, “User Interface Principles,” 2017. [Online]. Available: [https://msdn.microsoft.com/en-us/library/windows/desktop/ff728831\(v=vs.85\).aspx#the_basic_principles_of_proper_ui](https://msdn.microsoft.com/en-us/library/windows/desktop/ff728831(v=vs.85).aspx#the_basic_principles_of_proper_ui)
- [59] B. L. Rogers and B. Chaparro, “Breadcrumb navigation: Further investigation of usage,” *Usability News*, vol. 5, no. 2, pp. 1–7, 2003.
- [60] W. Bailey, B. Dillon, M. Labarge, J. Taylor, K. Gill, D. S. Lloyd, and W. M. Person, “Ribbon-style user interface for a software application,” 2008. [Online]. Available: <https://www.google.com/patents/US20080244440>
- [61] Microsoft Corp, “Microsoft Office,” 2017. [Online]. Available: <https://www.office.com/>
- [62] W. Lidwell, K. Holden, and J. Butler, “80/20 Rule,” in *Universal Principles of Design*, 2010, pp. 14–15.
- [63] A. McCrae, “CPi-IDE: A GUI for Continuous Pi Calculus,” Ph.D. dissertation, The University of Edinburgh, 2014.
- [64] Mathworks Inc., “Private Functions - MATLAB & Simulink - MATLAB United Kingdom,” 2017. [Online]. Available: https://uk.mathworks.com/help/matlab/matlab_prog/private-functions.html
- [65] —, “Add Help for Your Program - MATLAB & Simulink - MathWorks United Kingdom,” 2017. [Online]. Available: https://uk.mathworks.com/help/matlab/matlab_prog/add-help-for-your-program.html
- [66] T. Seemann, “Ten recommendations for creating usable bioinformatics command line software,” *GigaScience*, vol. 2, p. 15, 2013. [Online]. Available: <http://www.gigasciencejournal.com/content/2/1/15>
- [67] F. d. V. Leprevost, V. C. Barbosa, E. L. Francisco, Y. Perez-Riverol, and P. C. Carvalho, “On best practices in the development of bioinformatics software,” *Frontiers in Genetics*, vol. 5, no. 10, pp. 1451–1455, 7 2014. [Online]. Available: <http://journal.frontiersin.org/article/10.3389/fgene.2014.00199/abstract>
- [68] Balsamiq Studios LLC, “Balsamiq Mockups,” 2017. [Online]. Available: <https://balsamiq.com/products/mockups/>

- [69] W. Lidwell, K. Holden, and J. Butler, "Ockham's Razor," in *Universal Principles of Design*, 2010, pp. 172–173.
- [70] —, "Fitts' Law," in *Universal Principles of Design*. Rockport Publishers, 2010, pp. 98–99.
- [71] —, "Symmetry," in *Universal Principles of Design*. Rockport Publishers, 2010, pp. 234–235.
- [72] —, "Confirmation," in *Universal Principles of Design*. Rockport Publishers, 2010, pp. 54–55.
- [73] —, "Flexibility-Usability Tradeoff," in *Universal Principles of Design*. Rockport Publishers, 2010, pp. 102–103.
- [74] J. Nielsen and T. K. Landauer, "A Mathematical Model of the Finding of Usability Problems," in *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, ser. CHI '93. New York, NY, USA: ACM, 1993, pp. 206–213. [Online]. Available: <http://doi.acm.org/10.1145/169059.169166>
- [75] CIAT, "genebank4 Pic by Neil Palmer (CIAT). Plant samples in the gene bank at CIAT's Genetic Resources Unit, at the institution's headquarters in Colombia," 2010. [Online]. Available: <https://www.flickr.com/photos/ciat/4331057760/>
- [76] A. Arkagorodsky, "File:Software engineering intern.png - Wikipedia," 2008. [Online]. Available: https://en.wikipedia.org/wiki/File:Software_engineering_intern.png
- [77] Mathworks Inc., "Creating a GUI with GUIDE - Video - MATLAB," 2017. [Online]. Available: <https://uk.mathworks.com/videos/creating-a-gui-with-guide-68979.html>
- [78] —, "MATLAB App Designer - MATLAB," 2017. [Online]. Available: <https://uk.mathworks.com/products/matlab/app-designer.html>
- [79] M. Inc., "Graphics Support in App Designer - MATLAB & Simulink - MathWorks United Kingdom," 2017. [Online]. Available: https://uk.mathworks.com/help/matlab/creating_guis/graphics-support-in-app-designer.html
- [80] S. F. Conservancy, "Git," 2017. [Online]. Available: <https://git-scm.com/>
- [81] W. M. Vagias, "Likert-type Scale Response Anchors. Clemson International Institute for Tourism," & *Research Development, Department of Parks, Recreation and Tourism Management, Clemson University*, 2006.
- [82] W. Fontana, "Cellucidate," 2009. [Online]. Available: <http://fontana.med.harvard.edu/www/Documents/WF/Pages/cellucidate.wf.htm>
- [83] The COBRA Toolbox developers, "The COBRA Toolbox," 2017. [Online]. Available: <https://opencobra.github.io/cobratoolbox/latest/index.html>

- [84] J. Joe, S. Chaudhuri, T. Le, H. Thompson, and G. Demiris, “The use of think-aloud and instant data analysis in evaluation research: Exemplar and lessons learned,” *Journal of Biomedical Informatics*, vol. 56, pp. 284–291, 2015. [Online]. Available: <http://dx.doi.org/10.1016/j.jbi.2015.06.001>
- [85] C. J. Banks and I. Stark, “A logic of behaviour in context,” *Information and Computation*, vol. 236, no. 2014, pp. 3–18, 2014. [Online]. Available: <http://dx.doi.org/10.1016/j.ic.2014.01.009>
- [86] The University of Cambridge, “Sample Consent Forms,” 2017. [Online]. Available: <https://camtools.cam.ac.uk/wiki/site/e30faf26-bc0c-4533-acbc-cff4f9234e1b/exampleconsentform.html>