Developing a tabulation approach to multiscale modelling for systems biology

MSc Project Background Report

COMP60990 - Research Skills and Professional Issues

Vasileios Liolios

MSc in Software Engineering
2010
Abstract

This report is the first stage of a larger project that also includes the implementation of a software component and the writing of a dissertation paper accompanying the component. The report deals with two main subjects. The first one is a background literature review that will help in the development of the software component that will use a tabulation approach to improve the performance of the computational platform for multiscale simulation of biological systems, being developed by Prof. Mendes research group. The platform combines various components that were independently implemented using the extreme programming methodology. The main objective of the platform is to simulate biological systems that integrate different levels of organisation. The software component will use the ISAT algorithm that is already successfully implemented in combustion chemistry problems by Pope (1997). Before ISAT can be used, it has to be modified to match the context of systems biology, which refers to studies on the biological systems as a whole. This means that except the examination of all the separate components of a system, a detailed analysis of how they interact should be also carried out.

The second part of the report presents some details of the In Situ Tabulator component that will be developed and submitted as the main deliverable of the whole project procedure. The tabulator will be developed in C++ using the Eclipse IDE in two stages. Each stage will produce a different version of the component. The first version will have no functionality and its purpose is to test that the integration in the platform is successful. After ensuring that the platform functions properly with the integrated software, the second stage will begin. The functionality of the final component that will be produced in this stage will be to approximate values for subsequent queries that it receives from the higher level of the platform instead of calling the lower level to calculate them. In this way, computational cost is reduced since the approximation takes less time than an accurate calculation of the low level. A major challenge for the implementation is to choose an appropriate data structure to represent the vast amount of data that are handled by the tabulator. After analysing different data structures, the most appropriate one for the components implementation seems to be a binary tree, which contains the simulation data at each leaf nodes while the non-leaf ones serve as cutting planes to assist on the binary search performed for each query.
Contents

1. INTRODUCTION ................................................................. 4

2. BACKGROUND ........................................................................... 6
   2.1. ISAT IMPLEMENTATION IN COMBUSTION CHEMISTRY ................. 6
       2.1.1. Problem Statement ..................................................... 6
       2.1.2. Fundamentals ......................................................... 6
       2.1.3. Algorithm description ............................................... 7
       2.1.4. Algorithm evaluation ............................................... 8
   2.2. SYSTEMS BIOLOGY .......................................................... 9
       2.2.1. Origin ................................................................ 9
       2.2.2. Definition ............................................................. 10
       2.2.3. Future challenges and prospects ................................ 10
   2.3. MULTISCALE COMPUTATIONAL MODELLING IN BIOLOGY ....... 10
       2.3.1. Definition ............................................................... 11
       2.3.2. Scale integration issues .......................................... 11
   2.4. COMPUTATIONAL PLATFORM FOR MULTI-SCALE BIOLOGY SIMULATIONS ......................................................... 12
       2.4.1. Web Interface ......................................................... 12
       2.4.2. Culture Simulator .................................................... 12
       2.4.3. Pathway Simulators Connector .................................. 12
       2.4.4. In Situ Tabulator ..................................................... 13
       2.4.5. CopasiWS Client ..................................................... 13
       2.4.6. CopasiWS Service .................................................. 13
       2.4.7. Copasi Simulation Engine ....................................... 13
       2.4.8. Simulation Database ............................................... 13

3. RESEARCH METHODS ................................................................ 14
   3.1. PROJECT OUTLINE ........................................................ 14
       3.1.1. Project overview and deliverables ................................ 14
       3.1.2. Software component functionality ................................ 14
       3.1.3. Development method ............................................. 15
   3.2. TOOLS ........................................................................... 16
       3.2.1. Programming language ........................................... 16
       3.2.2. Ellipsoid library ..................................................... 16
       3.2.3. Eclipse ................................................................. 16
   3.3. DATA STRUCTURES ANALYSIS ........................................ 17
   3.4. PROJECT PLAN .............................................................. 20

4. REFERENCES ........................................................................... 21

APPENDIX A ............................................................................... 23
1. Introduction

It is widely accepted that biological systems are complex. It is not trivial to understand how different systems interact even if you are a biology expert. The main reason for this is that unlike other complex systems, biological systems are not merely composed of similar simple elements that interact with each other using equivalent types of interaction (Kitano, 2002a). Even the same element of a system can use multiple ways to communicate with another element, depending on the operation they have to complete. Even when all the properties of the system modules are exhaustively analysed, system’s behaviour can still be unpredictable due to new properties or different functionalities that may arise as a result of the combination of the modules. In addition the interactions work mostly in nonlinear modes, making prediction and understanding much harder.

Biologists may tackle this complexity in two different ways (Hartmann, 1996):

• Experimental simulations. In this case, they can construct physical models that have the characteristics of the system they want to study. On these models, they can run either in vitro or in vivo experiments to test their hypothesis and retrieve information that can be further used in new simulations.

• Theoretical simulations. These simulations are based on computational models that are abstract representations of the system to be examined. A vast majority of theoretical simulations are computer simulations, which are a part of the rapidly growing section of Biology called “in silico Biology” (Goldstein et al., 2010).

Frequently, biologists use a combination of both methods since each of them may be useful in different stages of hypothesis stating and testing. This can help them validate the results of an experiment through extra computer simulations without any additional reagents cost.

Nowadays, computer simulations in biology are becoming more and more popular because of the great progress of software development and the computational power that the latest computer systems can offer. However, computer resources such as RAM and CPU power still remain finite. Hence, most of the times, the vast amount of highly demanding calculations that needs to be carried out for a simple simulation is prohibitive when the computational model contains extensive details of the modelled system.

Things get worse when it comes to multiscale computational biology modelling, which consists of representing processes at different hierarchical levels of organisation, where the processes that occur at one level influence the level above. It is hoped that the increased complexity of these models helps the understanding of biological function. This understanding can be complete and consistent only if all relevant information gained from previous simulations or experiments can be integrated in a manner that the model is as informative as possible (Southern et al., 2008). Thus there is currently a need in computational systems biology for efficient multiscale modelling approaches.

Many research groups in various fields of science try to find efficient ways to deal with the problem of demanding calculations that arises in computational modelling. One of the scientific fields that many different solutions have been proposed and implemented is computational chemistry. DOLFA software is an implemented solution for combustion chemistry and reacting flow simulations that combines different algorithms, which improve the performance of database techniques that are used to approximate values of computational functions (Veljkovic et al., 2003; Veljkovic and Plassmann, 2005). Another given solution is the in situ adaptive tabulation (ISAT), which is a computational technique that reduces the computer time required to solve chemistry equations in reactive flow calculations (Pope, 1997).

In this paper, we will focus on ISAT technique. I will examine its implementation in combustion chemistry in detail and will seek an efficient way to implement it in the context of multiscale computational systems biology. Different data structures are going to be analysed.
in terms of memory efficiency and the most appropriate will be chosen. The software
component that is going to be developed will be a part of a larger system for multi-scale
simulation of biological systems, which uses the COPASI biochemical reaction simulator
(Hoops et al., 2006) as its core component.

The remaining of this paper has two main sections and it is organised as follows:

Section 2 consists of five subsections that provide some technical background knowledge. Subsection 2.1 describes the ISAT algorithm and how it was implemented in combustion chemistry. Subsections 2.2 and 2.3 introduce the concepts of “systems biology” and “multiscale computational modelling in biology” respectively, which will familiarise us with the general context that the component will be applied to. Finally, Subsection 2.4 gives a short description of the larger system in which the software component is going to be integrated into.

Section 3 includes five subsections presenting a plan for the actual implementation of the software component, including a rationale for the decisions that have to be made concerning different possible techniques to be adopted. Subsection 3.1 presents the deliverables of the project. The functionality of the software component that will be developed is given in Subsection 3.2. Subsection 3.3 studies in details different data structures that could be used in the implementation of the software module and presents the one that is going to be selected along with the reasons for this choice. Subsection 3.4 lists some tools that I decided to use together with some minor decisions that had to be taken concerning the general procedure of the component’s development. The project plan in the form of a Gantt diagram containing the important dates of the project duration is presented in the final Subsection (3.5).
2. Background

In this section I describe algorithms and software packages that are already implemented and will be used by our software component (Subsections 2.1, 2.2, 2.5). In addition, I will present specific aspects of the field that our software is going to be used in (Subsections 2.3, 2.4).

2.1. ISAT implementation in combustion chemistry

ISAT algorithm is already successfully implemented in reacting flow simulations so it is wise to analyse this implementation before we proceed with the development of our software component. Pope (1997) presents the ISAT technique in the context of probability density function (PDF) methods (Pope, 1985), but he suggests that it can be used efficiently in several approaches of reactive flows calculations.

2.1.1. Problem Statement

According to Pope (1997), in a reactive flow, the mass fractions of the species involved, the enthalpy and the pressure determine the state of the gas, which is represented by many computational particles according to PDF methods (Pope, 1985). Each particle’s composition changes on a specific timescale, which may vary among different particles. This evolution is represented via ordinary differential equations (ODEs).

The calculation of all the ODEs, that can be done starting from all the given initial conditions, finally results in the simulation of the whole computational model. Simulating the model is the main objective, so Pope (1997) has to cope with the problem of a computationally efficient calculation of all the equations. To emphasize the size of the problem, he gives an example of a full-scale PDF method calculation, saying that there may be 500,000 particles in it and each of those particles can evolve along 2000 timesteps, so equations must be solved approximately $10^9$ times.

2.1.2. Fundamentals

Before I get into details of the ISAT implementation itself, I have to present some notions and parameters that will help us understand what the algorithm actually does and what data it handles.

To begin with, composition $\varphi$ is the linearly independent subset of all the states of the mixture in a reactive flow. The accessed region is defined as all the compositions $\varphi$ that occur in the flow and it is a subset of the realizable region of the flow’s composition space (Pope, 2004). Pope (1997) uses the tabulation technique on accessed region instead of the realizable one and since the former depends on different aspects of the flow, it is not possible to construct the table before the actual calculation takes place. This is where the term “in situ” that characterises this technique comes from.

Reaction mapping $R(\varphi^0)$ is defined as the solution of the ODE that represents each particle’s evolution over time from an initial state composition $\varphi^0$ (Pope, 1997, 2004). These two values are stored in the table in addition with a mapping gradient matrix $A(\varphi^0)$. Pope (1997) obtains $R(\varphi^0)$ and $A(\varphi^0)$ using DDASAC code, which is an algorithm to calculate parametric sensitivity for systems modelled by ODEs (Caracotsios and Stewart, 1985). Using $R(\varphi^0)$, $A(\varphi^0)$ and $\varphi^0$, a linear approximation $R_l(\varphi^q)$ can be calculated with $\varphi^q$ being a given query point.

Local error $\varepsilon$ is defined as

$$\varepsilon = \left| B[R(\varphi^q)] - R_l(\varphi^q) \right|,$$

where $B$ is a matrix used to scale the different components of the mapping function (Pope, 1997).

Finally, special attention should be paid on what is called region of accuracy since it is the main controller of the method’s precision. The region is a hyper-ellipsoid, it contains $\varphi^0$ and it
Background Report: Developing a tabulation approach to multiscale modelling for systems biology

Vasileios Liolios, Student ID: 7575863
Page 7/23

consists of $\phi^o$ points for which $\varepsilon < \varepsilon_{tol}$, where $\varepsilon_{tol}$ is the defined error tolerance (Pope, 1997). A linear approximation $R'$ is only calculated when $\varepsilon$ is acceptable.

Pope (1997) presents the way that ISAT represents and estimates the region of accuracy as follows. The actual region of accuracy can be estimated by a hyper-ellipsoid called the ellipsoid of accuracy (EOA). This estimation takes place before the beginning of the procedure. Each time a query $\phi^o$ is found that is inside the region of accuracy but it is not included in the estimated EOA, EOA grows to cover this $\phi^o$. An illustration of this procedure is given in Figure 1.

![Figure 1: An illustration of how the pre-estimated EOA grows after a $\phi^o$ that is inside the region of accuracy occurs. Source: (Pope, 1997).](image)

2.1.3. Algorithm description

First, I will present a short description of the whole system that implements a reacting flow calculation using the ISAT technique and then I will analyse how ISAT algorithm is integrated in this system. As listed by Pope (1997), the software components that comprise the system are the following:

- **Reactive Flow Code (RFC):** responsible for passing timescale $\Delta t$, $\varepsilon_{tol}$ and scaling matrix $B$ to the ISAT component. In addition, it repeatedly provides ISAT with different queries $\phi^o$ and waits for the appropriate response.

- **In Situ Adaptive Tabulator (ISAT):** receives queries $\phi^o$ and returns mapping $R(\phi^o)$ using the values taken from Reaction Mapping Calculator. ISAT receives these values after passing $\Delta t$ and $\phi^0$ to the mapping calculator.

- **Reaction Mapping Calculator (RMC):** calculates $R(\phi^0)$ and $A(\phi^0)$ by using $\Delta t$ and $\phi^0$ and returns them to ISAT.

The table constructed by ISAT component contains a binary tree similar to the one shown in Figure 2. Each leaf of the tree represents a record that contains $R(\phi^0)$, $A(\phi^0)$, $\phi^o$, $Q$ (EOA unitary matrix) and $\lambda$ (diagonal elements of EOA matrix $A$). $Q$ and $\lambda$ are used to determine EOA after each growth as described in 2.1.2. Each non-terminal node of the tree represents a cutting plane defined by a vector $v$ and a scalar $a$. ISAT uses these two values to control how the tree is going to be traversed.

Following is a list of the steps of the ISAT algorithm based on Pope (1997) and then a depiction of these steps in a flowchart diagram (Figure 3).

When the algorithm gets the first query $\phi^o$ as an input, it simply sets $\phi^0 = \phi^o$ and returns the exact value $R(\phi^0) = R(\phi^o)$. For the rest queries ISAT behaves in the following way:
i) Accepts query $\phi^q$ and traverses the binary tree until $\phi^0$ is reached.

ii) Checks whether $\phi^0$ is inside EOA or not.

iii) If $\phi^0$ is inside EOA, it returns the linear approximation (retrieve action-R)

\[
R'(\phi^q) = R(\phi^0) + A(\phi^0)(\phi^q - \phi^0)
\]  

(2)

iv) If $\phi^0$ is outside EOA, it calculates $R(\phi^0)$ and $\varepsilon$.

v) If $\varepsilon$ is less than $\varepsilon_{\text{tol}}$, the EOA is grown and ISAT returns $R(\phi^0)$ (growth action-G).

vi) If $\varepsilon$ is greater than $\varepsilon_{\text{tol}}$, ISAT generates a new record based on $\phi^q$ and adds it in the table (addition action-A).

![Binary tree contained in the table generated by ISAT algorithm.](image)

**Figure 2:** Binary tree contained in the table generated by ISAT algorithm.

As I mentioned before, the cutting planes that are stored in the nodes of the binary tree are values determining how the tree is going to be traversed in a way such that the $\phi^0$ we find at step (i) is close to the given $\phi^q$. According to Pope (1997), the ideal case would be that ISAT could find the closest $\phi^0$ but this is computationally inefficient, so the binary tree search method (Cormen et al., 2001) is used instead. Cutting plane values also decide how the addition action in (vi) is performed. The orientation of vector $v$ chosen in such way that $\phi^0$ is on the left of the plane and $\phi^q$ on the right.

### 2.1.4. Algorithm evaluation

After testing the algorithm in different situations, Pope (1997) found out that the speed up factor

\[
\text{speed-up factor} = \frac{\text{CPU time for } Q \text{ queries using Direct Integration}}{\text{CPU time for } Q \text{ queries using ISAT}}
\]

(3)

is approximately 1000 when $Q$ is greater than $10^6$. When $Q$ is less than $10^6$ ISAT is still faster than Direct Integration but the speed factor is not greater than 10.
2.2. Systems biology

After analysing the ISAT implementation in chemistry we are ready to become more familiar with the context that our algorithm will be used in, by trying to shed some light on systems biology.

2.2.1. Origin

Before we attempt to define what systems biology is, we should examine how it arose. Systems biology is not new in the sense that back in 1940, Norbert Wiener constructed mathematical models representing complex systems and the ways they function in different time points (Wiener, 1961). However, the exponential growth of this field has been observed after the year 2000.

The main reason for this growth seems to be the significant progress of molecular biology and the vast amount of data that can be obtained by studies on genotype constitution (Kitano, 2002b). The size of the existing data pools concerning genomics makes the existence of models and simulations inevitable since a single biologist cannot understand and productively use all the available information without the help of a computer system. But even if this was possible, data obtained from individual living components cannot explain the complete...
functionality of an organism or a system comprised of these components due to emergence. Emergence is a property of systems that present different behaviour than what it was expected by analysing the characteristics of their components (Aderem, 2005; Levesque and Benfey, 2004). It is obvious that a need for a field that can combine huge amount of data about individual system components and monitor the systems functionality as a whole has emerged. This field is systems biology.

2.2.2. Definition

Many definitions given for systems biology can be found in literature relevant to this field. It has been defined as a procedure of ‘studying the interrelationships of all the elements in a system rather than studying them one at a time’ (Hood, 2003), as ‘study of the behaviour of complex biological organization and processes’ (Kirschner, 2005), as ‘a comprehensive quantitative analysis of the manner in which all components of a biological system interact functionally over time’ (Aderem, 2005) and has been characterized as ‘the united Nations of science’ (Naylor and Cavanagh, 2004). What is common in all the given definitions is that systems biology studies the interactions between individual parts of a system and not their properties in isolation.

Interactions between system components define and explain the whole functionality of the system and how it is affected by alterations in the way the components cooperate. To achieve a good understanding of this functionality, a framework with specific steps is proposed (Ideker et al., 2001):

i) Identify all the system components.

ii) Consecutively alter the way the components interact and monitor them by performing experiments.

iii) Change the model such that its predictions match the outcomes of the experiments.

iv) Design and perform new experiments to test the new model that was created in the previous step.

2.2.3. Future challenges and prospects

As I mentioned before, systems biology started developing recently despite its early appearance. Thus, it is unavoidable to face some challenges like the ones following, which were spotted by different scientists working in the field. Aderem (2005) focuses on different technical challenges such as data quality and standardisation, measurements accuracy, network biology immaturity and imaging extension needs. He also refers to academic challenges concerning teamwork, crediting and data ownership issues, difficulties in accessing the appropriate technology and funding problems. Naylor and Cavanagh (2004) are concerned with the different science fields involved in systems biology and the fact that the language that has to be used by the different specialists is not fully defined yet. Finally, Levesque and Benfey (2004) are also worried about data sharing issues and the wide range of knowledge from various scientific contexts that has to be combined in systems biology.

Despite the concerns, it is widely accepted that systems biology is going to continue its evolvement and will play an important role to the understandings of life. The knowledge that systems biology generates can be applied to many different practical fields such as medicine, pharmacy, agriculture and biotechnology.

2.3. Multiscale computational modelling in biology

As we mentioned in subsection 2.2, for complex systems to be analysed, there is a great need of data integration in terms of systems biology. This integration is not experimentally possible because of the extremely big amounts of available data. Hence, we concluded that models should be constructed so the experiments could be simulated and analysed in a slightly easier way. However such an approach has its own difficulties with the main one being the
computational cost of simulations. In this subsection we are going to present some details about one of the most demanding forms of simulations, multiscale computational modelling.

2.3.1. Definition

A given definition to multiscale modelling is ‘a model which includes components from two or more levels of organisation or if it includes some processes that occur much faster in time than others’ (Southern et al., 2008). The second part of the definition is self-explanatory since it matches the term ‘scale’ with the notion of a time scale. To fully understand the first part though, we have to elaborate on the meaning of term ‘levels of organisation’.

‘Life has so many scales’ (Schnell et al., 2007). In addition to time scales, a scale in biology is also a level of organisation. It is a discrete, but not a hundred percent accurately definable, position in biological hierarchy depending on which point of this hierarchy the biological processes take place (Southern et al., 2008). Each level of organisation contains specific processes and can be modelled in a determined way. These one-level simulations can be easily performed and they are not necessarily complex. Hence, many software tools have been developed for single scale modelling and one of them is COPASI simulator of biochemical reaction networks that will be the core of the multiscale modelling platform described in 2.4. In Table 1, we give a summary of the levels of organisation together with which model can represent them and what kind this model is as defined in Sothern et al. (2007).

We have to point out at this point, that as we move from smaller (i.e. quantum, molecular) to larger ones (i.e. organism, environment), the computational cost to model each level by integrating the previous ones is increasing. This rise of the cost is due to the involvement of lower levels into modelling of higher ones. For example, before simulating an organ system you need to simulate the organs and for the modelling of the organs you need to formulate tissue models. This is the main reason that only limited amount of work on high level of organisation modelling has been completed successfully so far.

Table 1: Levels of biological organisation and characteristics of their models (Southern et al., 2008).

<table>
<thead>
<tr>
<th>Level name</th>
<th>Model</th>
<th>Model type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantum</td>
<td>Simplified models expressed in Schrödinger equation.</td>
<td>Stochastic</td>
</tr>
<tr>
<td>Molecular</td>
<td>Calculating the forces using Newton’s laws.</td>
<td>Stochastic</td>
</tr>
<tr>
<td>Macro-molecular</td>
<td>Calculating the forces using Newton’s laws.</td>
<td>Stochastic</td>
</tr>
<tr>
<td>Sub-cellular</td>
<td>PDEs and spatially varying density functions.</td>
<td>Deterministic</td>
</tr>
<tr>
<td>Cellular</td>
<td>Conservation laws in the form of ODEs.</td>
<td>Deterministic</td>
</tr>
<tr>
<td>Tissue</td>
<td>Mechanical, consisting of PDEs.</td>
<td>Deterministic</td>
</tr>
<tr>
<td>Organ</td>
<td>Integrating separate tissue models and geometry of the organ.</td>
<td>Deterministic</td>
</tr>
<tr>
<td>Organ system</td>
<td>Two or more organ models and a model for the interfaces between them.</td>
<td>Deterministic</td>
</tr>
<tr>
<td>Organism</td>
<td>Integrating organ and organ system models.</td>
<td>Deterministic</td>
</tr>
<tr>
<td>Environment</td>
<td>Empirical data defining probabilities of events occurrence.</td>
<td>Deterministic</td>
</tr>
</tbody>
</table>

2.3.2. Scale integration issues

By observing Table 1, we can notice that the first three levels are modelled using stochastic methods while the rest are done by deterministic methods, a fact that causes some of the
problems discussed in this section. It has been proven difficult to combine these two different types of models when you need to model a set of organisation levels that includes both stochastic and deterministic processes (Southern et al., 2008). Hence, not much has been done towards this direction and, usually, these two separate kinds are being treated separately.

The issue that rises when stochastic processes are being scaled with each other is that the computational cost grows rapidly as the number of levels increases, because low-level processes are performed rather quickly. When deterministic processes of different scale levels are being integrated, the most common issue observed is the trend to simply link the existing models without creating anything that will serve as an interface between them. As a result to this common mistake, reasonable outcomes derived from one model can cause problems to other models that take it as an input without the appropriate pre-modification.

Note that multiscale models link two or more levels but they do not necessarily link all the possible levels of Table 1. For example, a multiscale model can consist of the cellular and organ levels without considering the tissue level, as long as the modellers are able to postulate the appropriate links between them.

Despite the fact that multi-scale modelling is still in a fairly undeveloped stage, extended work has been done in some areas such as ion channels simulations and heart modelling (Southern et al., 2008).

2.4. Computational platform for multi-scale biology simulations

As mentioned in the introduction, the tabulator component that is going to be created will be integrated in a larger system that is already under development in Prof. Mendes research group. The purpose of the system is to ease the simulations of single-scale or multi-scale complex systems using a number of components that integrate a wide range of technologies. Tabulator will be an enhancement to the system that will improve the time required to run a simulation, but the system remains functional even if the tabulator is not present. Figure 4 shows the structure of the system, how the components interact and where the tabulator is going to be added. In the following subsections, we will briefly present the functionality of each component.

2.4.1. Web Interface

Users of the system interact with this component to submit cell dynamic models in SBML format with custom model parameters. Simulation results are displayed using this interface.

2.4.2. Culture Simulator

This is the component that runs the simulation of the top level of organisation. In this case the top level represents a culture of yeast cells in a well-stirred liquid medium. In this configuration, each cell interacts directly with the medium (and therefore indirectly with each other). This simulator is based on an agent-based approach. Multi-agent systems are systems that integrate a number of intelligent agents that interact with each other to solve a problem in an easier way than a single agent would (Wooldridge, 2009). This component uses such a multi-agent system approach where three types of software agents take part: cell, medium and master agents. The medium agent manages environmental variables and interacts with the cell agents. The master agent regulates the simulation by controlling the behaviour of the medium and cell agents. It is developed in Java and connects to Pathway Simulators Connector via a Java native interface which is a framework that enables non-Java software to run Java code (Liang, 1999).

2.4.3. Pathway Simulators Connector

It prepares the network model and provides the initial conditions for it. It either connects to CopasiWS Client or to the In Situ Tabulator to run CopasiWS Service. The user can determine with which of the two components Pathway Simulators Connector is going to interact with.
2.4.4. **In Situ Tabulator**

It implements the ISAT algorithm attempting to reduce the computational cost of simulations and details of its functionality are given in Section 3.2. This is the component that the present project is developing.

2.4.5. **CopasiWS Client**

It uses a gSOAP interface to connect to the lower component, CopasiWS Service. CopasiWS Client returns a matrix of ODE solutions extracted from a comma separated values file or an SBRML format, which is an XML-based language that connects models with different datasets to help sharing of simulations results in a standardised way (Dada et al., 2010).

2.4.6. **CopasiWS Service**

This component is a web service implementation of COPASI and its purpose is to prepare the input and output files for the Copasi Simulation Engine before it runs the engine (Dada and Mendes, 2009). The input model is converted to CopasiML format and the given simulation parameters are included in it. The output is converted to SBRML or tab delimited format.

2.4.7. **Copasi Simulation Engine**

This component could be characterised as the core component of the system since it is the calculation engine that simulates dynamic models and produces time course data as a result of the simulation. This is the component that will run the simulations at the cellular level (ie the lower level). Extended information for COPASI simulator is provided in Hoops et al. (2006).

2.4.8. **Simulation Database**

The database stores data and cell dynamic data provided by the first two components.
3. Research methods

In this section, I will initially give an overall description of the project and its deliverables (Subsection 3.1). In addition, I will discuss the tools that will be used to produce the main deliverable of the project and some decisions that had to be made to make this deliverable as efficient as possible (Subsections 3.2, 3.3). Finally, a project plan will be presented, showing how I intend to carry out the whole project (Subsection 3.4).

3.1. Project outline

3.1.1. Project overview and deliverables

As I mentioned several times, the project is a part of a larger project that is already under development (details given in 2.4). The wider project aims to provide a flexible system for multi-scale simulation of biological systems that can be modified to function properly for any kind of biological systems. My project involves the development of a software component called In Situ Tabulator that will be embedded in the larger system and enhance its functionality.

The aim of the In Situ tabulator is to improve the performance of the system that simulates multi-scale biological systems by reducing the computational cost of these simulations. The reduction is going to be done using a slightly modified version of ISAT algorithm described in 2.1. The modification concerns the transfer of the algorithm form chemistry to biology context.

The deliverables of the project are the following:

- The In Situ Tabulator software module developed in C++ and its source code.
- A dissertation paper complying with University’s rules. The paper will give extended details about the design of the module and its implementation and testing.

3.1.2. Software component functionality

The main functionality of the In Situ Tabulator is to approximate values that are used by the platform to perform a simulation and therefore, reduce the total time required to finish the representation of a multi-scale biological system. The reduction is due to the less time needed for an approximation than for an accurate calculation of the values that form the simulation by solving the ODEs that represent the modelled system. A major factor that will play an important role on the efficiency of the tabulator is the data structure that we will choose to store the solutions of the ODEs in memory. The choice is important because a correctly chosen structure will help us handle the available memory in the best possible way to achieve high performance. A detailed analysis on different data structures and which one will be finally selected is given in section 3.

This operation is performed as follows. Pathway Simulators Connector, which is a level higher in the platform system, calls the tabulator and provides a set of initialisation parameters (initial condition) to it. The tabulator stores this initial condition together with its ODE solution in memory and continues to receive subsequent requests from the connector for further timesteps. These requests include new values that represent different conditions of the modelled system. For each request received from the connector, if there are similar conditions for which the tabulator has already received the ODE solutions from CopasiWS client in previous timesteps, it calculates an approximate solution and returns the values to the connector after storing it in its memory. If no similar conditions exist, CopasiWS is called and the request from the connector is passed to it to run the simulation engine. An exact calculation of the ODE solution is returned to the tabulator and is stored in its memory to be used for future approximations. A high level representation of this procedure is given in Figure 5.

We are going to check whether a condition of the modelled system represented in a request to the tabulator is similar to a previous one or not, by using ellipsoids and algorithms to
transform them appropriately as Pope (1997) does in his own implementation of ISAT algorithm. Some details for these ellipsoid algorithms are given later in subsection 3.2.2.

![Diagram of In Situ Tabulator's functionality](image)

**Figure 5:** A high level representation of In Situ Tabulator's functionality.

### 3.1.3. Development method

The platform is developed using extreme programming (XP) methodology and the same method will be used for the In Situ Tabulator. The main advantage of the XP method is that every component of the platform can be developed individually. The changes in the implementation of such a component do not affect the other components due to connectivity through interfaces between different levels. Another benefit of this technique is that it supports incremental software development, feature that we will take advantage of, because the implementation will be done in two discrete phases described next.

In the first phase, I will create a software component with minimum functionality. This functionality includes the ability of the module to connect with the upper and the lower levels of the platform to exchange data. This first version will not transform the data in any way. It will just be a bridge between Pathway Simulators Connector and CopasiWS Client. Hence, running a simulation with or without the first version of the tabulator will have the same outcome in terms of time needed for the simulation to be completed. After the first version is developed, it will be integrated in the platform and tested in example simulations to assure that no connectivity issues arise.

During the second phase of the implementation I will add the complete functionality that the final version of the In Situ Tabulator should have as it has been already described in 3.1.2. After the end of the second phase implementation, the component is going to be integrated in the platform and tested using simulations. To evaluate the component, the time that each simulation needs to be performed with the tabulator’s help will be calculated and compared to the corresponding time that the same simulation consumed without using the tabulator module. The whole development procedure is shown in Figure 6.
3.2. Tools

In this section I list the tools I will use for the tabulator’s development together with some reasons to justify these choices. The need of more tools may arise during the actual production of the software module, so what follows is more a list of decisions taken before the beginning of the code development rather than an exhaustive list of everything that may be useful for our development. The final list will be presented in the dissertation paper that will be delivered together with the final version of the tabulator.

3.2.1. Programming language

I chose C++ as the programming language to develop the software component. The other choice considered was Java but was eliminated since the main goal is efficiency of the component in terms of performance. Java software modules need the Java Virtual machine to be launched before they run and this could be a considerable cost in execution time. On the other hand, C++ programs do not need any external application to run in order to be executed. Another advantage of C++ in this case is the ability to manually allocate memory, something that could be crucial since the main objective is to handle memory efficiently to improve the performance of the tabulator. The feature of Java called garbage collector would provide this opportunity without giving me the ability to manually control it. Finally, the ellipsoid library described in 3.2.2 that will be used by our algorithm is written in FORTRAN. Hence, it was easier to develop an interface in C++ for this library than it would be to create a Java one, since FORTRAN subroutines can be called as C++ functions without any further complex transformation beyond linking binaries.

3.2.2. Ellipsoid library

As I have mentioned in 3.1.2, a check is needed to decide whether a given request is close to one that has been already calculated using ellipsoids and algorithms to transform them. These algorithms do not need to be implemented here since there is a FORTRAN library called ELL_LIB that contains all the necessary algorithms to perform basic geometric operations on ellipsoids (Pope, 2008). I will not give a detailed description of all the algorithms implemented in the library as this goes beyond the scope of this report. A summary of the routines can be found in Appendix A, while Pope (2008) gives an extensive description of their implementation.

3.2.3. Eclipse

I will develop the code using Eclipse Integrated Development Environment (IDE) and the C++ Development Toolkit (CDT), which is a set of plug-ins written for Eclipse platform (Eclipse, 2010a). CDT supports full development of C and C++ applications with several helping features such as pre-constructed code templates and code assisting, which are expected to ease the implementation procedure. These features exist in other IDEs too, but what made me choose Eclipse is the extensibility of the platform provided by the numerous plug-ins that are written for various purposes. Such a plug-in that I downloaded and installed is Photran, which provides Eclipse with the feature of FORTRAN development (Eclipse, 2010b). I need Photran to integrate the ELL_LIB library into the software component.
3.3. Data structures analysis

As we have seen in previous sections of this report, the main goal of this project is to improve the performance of the simulation platform described in 2.4 by integrating the tabulator in it. Thus, I have to optimize the tabulator in a way that it needs the least possible amount of computational power to run with its execution time being minimised. A significant factor that could increase the tabulator’s efficiency is how the data that the component handles are being represented in memory. A selection of the appropriate data structure could be proved valuable in terms of performance of the overall system since the software module is expected to handle vast amounts of data received by both upper and lower layers of the platform.

Before I am able to chose a suitable data structure, I need to examine the nature of the data and check how they are related. When I conclude what the exact connection of the data is, I will be able to proceed to the next step, which is to analyse different structures and finally, select the most proper one.

In situ Tabulator is mainly required to handle two kinds of data. The first kind of data consists of filenames of the model files and the initial conditions corresponding to these models. These pairs of data are subsequently received as queries from the higher level during the whole simulation. ODE solutions are the second kind of data received from the lower layer and are linked with the model name and initial condition that was passed to this level to compute them. Each pair of a model filename and its initial condition that enters the tabulator has only one set of ODE solutions that are either calculated by the lower level or returned as an approximation by the tabulator. However, each set of ODE solutions may be attached to more than one pairs of filename and initial condition. This is because an approximation means that for similar initial conditions for the same filename, we return exactly the same set of ODE solutions without calling the lower level to compute them again, provided that the timescales are identical too. Timescales need to be identical because the ODE solutions are time course data, which means that they need to include the same amount of timesteps. If different amount of timesteps is included, then the ODE solution for the received query must be calculated by the lower layer, despite the fact that the initial condition and the model’s filename are the same. An illustration of how the data are manipulated by the tabulator is given in Figure 7.

![Figure 7: Illustration of data handling by In Situ Tabulator.](image)

After examining the kinds of data, I can now explain their types. The filename is a string that contains the name of the file where modelled system’s information is being held. A new
unique name is generated for each model file entered in the platform by the platform itself. Initial condition is a set of numerical values that refer to various parameters of the system and determine its state. Both initial condition and filename can vary during a single simulation depending on the nature of the modelled system. ODE solutions are numerical values of different parameters calculated for sequential time steps (time course data) by the Copasi Simulation Engine and passed to the tabulator via CopasiWS client.

It is time to analyse how the data that the tabulator handles are related. As I have mentioned earlier, each pair of a filename and an initial condition (often referred as a query) is linked with a group of ODE solutions. Nevertheless, there is no pattern that defines how these queries are associated with each other. The only thing that we can assume is that the closer in terms of time steps a query is to another query, the more possible it is to have similar initial conditions passed to the tabulator. However, this assumption is not safe because the behaviour of the biological system is rather unpredictable. Hence, it will not be taken into consideration at this stage of the analysis.

Before I start the analysis of the different possible data structures that could be used, it is useful to present the tool that will help me chose one. It is called the Big-Oh notation and it is used to analyse the running time of different algorithms and actions performed on various data structures. Its formal definition is that if \( f(n) \) is a function that represents the execution time \( T(n) \) of an action on a specific data structure, then the action is \( O(g(n)) \) if, for all values \( n > n_0 \), it is \( T(n) = O(g(n)) \), where \( n^c \) and \( c \) are constants (Budd, 1997a). A more intuitive explanation is that I will try to analyse the performance of different actions on data, based on the size of the data (\( n \) in this case), ignoring technical specifications of the computer system where the data are stored and processed.

In Table 2 we can see a list of the most commonly used data structures along with a brief explanation of what each of them is as presented in Budd (1997b). With a quick look at Table 2, it is clear that some of the structures are not appropriate for the tabulator before I even examine the performance of the actions carried out on them. The first of the structures that is not suitable is vectors, because their size must be determined in advance of execution of the algorithm. Moreover, stacks and queues are not helpful because insertions and removals can be performed only at their edges. This is not convenient for the implementation of the tabulator since the ability to insert new or remove stored triads of values (\( Ic, nm, ODE \)) from any point of the structure is crucial when the memory of the system is full. The last structure that I will not examine in terms of performance is priority queues due to the fact that among the data that the tabulator manipulates, there is not notion of a largest element.

The remaining structures that are subject to performance evaluation are lists, double-ended queues, sets, maps and binary trees. A summary of their performance for the four basic operations that the tabulator needs to perform on the data is presented in Table 3. The retrieval operation is going to be performed each time the tabulator finds a similar initial condition with the same timescale to the one that the latest query contains. On the other hand, insertions are executed when the query contains an initial condition that is no similar with any condition previously entered in the tabulator. Removals will only occur when the memory is full, something that is not expected to happen very often especially for small simulations. For longer simulations, which are also the most computationally demanding, the retrievals are expected to be the most often occurring operations as the simulation proceeds. Special attention needs to be paid on the procedure of searching for a similar condition in the structure, which is executed for each received query. Hence, the performance analysis of different structures that follows is based mainly on the operations of searching and retrieval.

Looking at Table 3, it is obvious that in terms of retrievals, binary tries are the most efficient data structures. For small values of \( n \) the difference between \( O(n) \) and \( O(log n) \) is not big, but as \( n \) grows, \( O(log n) \) is significantly faster than \( O(n) \). In our case, \( n \) is the amount of \( Ic, nm, \) and \( ODE \) triads of values that are stored in tabulator’s memory and this amount is expected to be substantial. Another advantage is that a binary search, which is of complexity \( O(log n) \), can
be easily carried out in a binary tree and it is the most efficient among the examined structures. Lists and double-ended queues have a great performance in terms of insertions but as I have mentioned before, insertions happen less often than retrievals in large simulations as the simulation advances in time. In addition, a major disadvantage of these two structures is that they have to be searched sequentially, which is computationally prohibitive. The same drawback can be observed in maps and sets because the data that the tabulator handles cannot be order by an appropriate value as these two structures demand.

**Table 2:** A list of the most commonly used data structures and a brief description for each one of them. Source: (Budd, 1997b).

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>A fixed length group of elements indexed by integer keys.</td>
</tr>
<tr>
<td>List</td>
<td>A collection of elements with unbounded length. The elements are not indexed.</td>
</tr>
<tr>
<td>Double-Ended Queue</td>
<td>A collection of indexed elements with no size limit. Insertions and removals of elements can be done at both ends and they are also allowed in the middle.</td>
</tr>
<tr>
<td>Stack</td>
<td>Specialised form of a Double-ended Queue with the difference being that elements can only be removed or added only at the front of it.</td>
</tr>
<tr>
<td>Queue</td>
<td>Specialised form of a Double-Ended Queue with the difference that insertions are performed only at the back of the queue, while removals are done only from its front.</td>
</tr>
<tr>
<td>Priority queue</td>
<td>The element with the largest value is always accessible, while the rest of them are unordered.</td>
</tr>
<tr>
<td>Set</td>
<td>A collection of elements with unique values, usually ordered.</td>
</tr>
<tr>
<td>Map</td>
<td>An indexed collection of pairs of keys and corresponding values. Keys can be data of any kind. A map can be ordered.</td>
</tr>
<tr>
<td>Binary Tree</td>
<td>A collection of nodes where each of them has zero to two children nodes and only one parent node except the root that has none. Nodes with no children are called leaf nodes.</td>
</tr>
</tbody>
</table>

**Table 3:** Evaluation of the three main actions that tabulator performs on the data it handles for four different data structures. Source: (Budd, 1997b).

<table>
<thead>
<tr>
<th>Structure</th>
<th>Insertion</th>
<th>Removal</th>
<th>Retrieval</th>
<th>Searching</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Double-Ended Queue</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Set (unordered)</td>
<td>O(logn)</td>
<td>O(logn)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Map (unordered)</td>
<td>O(logn)</td>
<td>O(logn)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Binary Tree</td>
<td>O(logn)</td>
<td>O(logn)</td>
<td>O(logn)</td>
<td>O(logn)</td>
</tr>
</tbody>
</table>

In conclusion, a binary tree seems to be the best choice to represent the data controlled by the In Situ Tabulator. Similarly to what Pope (1997) has implemented, the tree’s non-leaf nodes will contain sets of values calculated by the ELL_LIB library that will be the cutting planes determining how the binary search will be performed on the trees. The actual data that are useful for the simulation will be stored in the leaf nodes of the tree. These data are the model filename, the initial condition, the given timescale and the related ODE solution. Non-leaf nodes will control which of the leaves will be accessed each time a query is received from the upper layer. After the node is accessed, a comparison between the stored values and the query will take place. Ellipsoid algorithm will decide if the compared values are similar and according to this decision the tabulator will either retrieve the ODE or add another leaf in the tree after it receives the new calculated ODE from the lower layer. In Figure 8, a random instance of the binary tree that will be integrated in the tabulator is presented. I have to point
out here that the decision for the data structure to be used has been taken before the beginning of the actual implementation of the tabulator and it may be subject to changes later on, after testing the software component as a part of the multiscale simulation platform.

Figure 8: Random instance of the binary tree representing the data that the tabulator controls.

3.4. Project Plan

As we can see in Figure 9, the project will be completed in three distinct stages. The first stage is about studying all the useful background information and writing a report that summarizes what has been done before we begin the development of the main deliverable of the project, which is the tabulator component. After the first stage, exams take place and they are considered as an external task. During the second stage, the actual development of the component is performed. This stage is divided into two phases, one for each version of the tabulator. These two phases include a small period of testing the component’s functionality. Finally, the third stage is about dissertation writing and reviewing before its submission.

The whole project has two milestones marking the end of stages one and three. The first is the delivery of the background report on 14th of May and the second is the submission of the final versions of the project’s deliverables on 10th of September.

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
<th>Oct</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stage 1: Background literature review</td>
<td>75 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Studying relevant literature</td>
<td>42 days</td>
<td>1/2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Creating the structure of the background report</td>
<td>9 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Writing the background report</td>
<td>23 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Submitting background report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14/5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Studying for the exams</td>
<td>14 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8/5</td>
<td>26/5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Stage 2: Software development</td>
<td>48 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Developing 1st version of the software module</td>
<td>14 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>27/5</td>
<td>15/6</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Developing final version of the software module</td>
<td>34 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16/6</td>
<td>2/8</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Stage 3: Dissertation paper writing</td>
<td>29 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Writing the dissertation paper</td>
<td>22 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3/8</td>
<td>1/9</td>
</tr>
<tr>
<td>12</td>
<td>Reviewing the dissertation paper and making the final corrections</td>
<td>5 days</td>
<td>1/9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/9</td>
<td>7/9</td>
</tr>
<tr>
<td>13</td>
<td>Submitting source code and the dissertation paper</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10/9</td>
</tr>
</tbody>
</table>

Figure 9: Gantt chart for the project plan.
4. References


### Appendix A

**Table 4:** A list of the routines contained in ELL_LIB library. $E_i$ represents an ellipsoid in $n$ spatial dimensions and $p$ is a given point (Source: Pope (2008)).

<table>
<thead>
<tr>
<th>Routine</th>
<th>Task performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ell_rad_lower</td>
<td>Determine the smallest principal semi-axis of $E$.</td>
</tr>
<tr>
<td>ell_rad_upper</td>
<td>Determine the largest principal semi-axis of $E$.</td>
</tr>
<tr>
<td>ellu_rad_upper</td>
<td>Determine the smallest and largest principal semi-axes of $E$.</td>
</tr>
<tr>
<td>ell_pt_in</td>
<td>Determine whether $p$ is covered by $E$.</td>
</tr>
<tr>
<td>ell_pt_dist</td>
<td>Determine the relative distance from $p$ to the boundary of $E$.</td>
</tr>
<tr>
<td>ell_pt_near_far</td>
<td>Determine the point in $E$ which is nearest to or furthest from $p$.</td>
</tr>
<tr>
<td>ell_pt_modify</td>
<td>Determine $E_V$, the modification of $E$ of least change of content whose boundary intersects with $p$.</td>
</tr>
<tr>
<td>ell_pt_shrink</td>
<td>Shrink $E$ based on $p$.</td>
</tr>
<tr>
<td>ell_pt_hyper</td>
<td>Determine a separating hyperplane between $E$ and $p$.</td>
</tr>
<tr>
<td>ellPts_uncover</td>
<td>Determine an ellipsoid (of large content), which does not cover any given point.</td>
</tr>
<tr>
<td>ell_line_proj</td>
<td>Project $E$ onto a given line.</td>
</tr>
<tr>
<td>ell_aff_pr</td>
<td>Project $E$ onto a given affine space.</td>
</tr>
<tr>
<td>ell_pair_shrink</td>
<td>For concentric $E_1$ and $E_2$, shrink $E_2$ (if necessary) so that it is covered by $E_1$.</td>
</tr>
<tr>
<td>ell_pair_separate</td>
<td>Determine if $E_1$ and $E_2$ intersect; and, if they do not, determine a separating hyperplane.</td>
</tr>
<tr>
<td>ell_pair_cover_query</td>
<td>Determine whether $E_1$ covers $E_2$.</td>
</tr>
<tr>
<td>ell_pair_cover</td>
<td>Determine an ellipsoid (of small content), which covers $E_1$ and $E_2$.</td>
</tr>
</tbody>
</table>