CAPTURING ODE KNOWLEDGE FOR SCS

CAPTURING ORDINARY DIFFERENTIAL EQUATIONS KNOWLEDGE FOR SCS IN DRASIL

BY

DONG CHEN, M.Eng.

a Report
submitted to the Department of Computing and Software
and the School of Graduate Studies
of McMaster University
in partial fulfilment of the requirements
for the degree of
Master of Engineering

Master of Engineering (2022)	McMaster University
(Department of Computing and Software)	Hamilton, Ontario, Canada

TITLE:	Capturing Ordinary Differential Equations Knowledge for SCS in Drasil
AUTHOR:	Dong Chen M.Eng. in (Systems Engineering), Boston University, Massachusetts, USA
SUPERVISOR:	Spencer Smith and Jacques Carette
NUMBER OF PAGES:	**??**, **??**

Lay Abstract

A lay abstract of not more 150 words must be included explaining the key goals and contributions of the thesis in lay terms that is accessible to the general public.

Abstract

Abstract here (no more than 300 words)

Your Dedication

Optional second line

Acknowledgements

Acknowledgements go here.

Contents

List of Figures

List of Tables

Notation, Definitions, and Abbreviations

Notation

$A \leq B$	A is less than or equal to B

Definitions

Challenge	With respect to video games, a challenge is a set of goals presented to the player that they are tasks with completing; challenges can test a variety of player skills, including accuracy, logical reasoning, and creative problem solving

Abbreviations

AI	Artificial intelligence

Declaration of Academic Achievement

The student will declare his/her research contribution and, as appropriate, those of colleagues or other contributors to the contents of the thesis.

Chapter 1

Introduction

From the Industrial Revolution (1760-1840) to the mass production of automobiles that we have today, human beings never lack innovation to improve the process. In the Industrial Revolution, we start to use machines to replace human labour. Today, we have been building assembly lines and robots in the automobile industry to reach a scale of massive production. Hardware automation has been relatively successful in the past one hundred years, and they have been producing mass products for people at a relatively low cost. With the success story of automating hardware, could software be the next one? Nowadays, the software is used every day in our daily life. Most software still requires a human being to write them. Programmers usually write software in a specific language and produce other byproducts during development time. Whether in an enterprise or research intuition, manually creating software prone to errors, and it is not as efficient as a code generator. In the long term, a stable code generator usually beats programmers in performance. They will eventually bring the cost down because of the labour cost reduction. Perhaps this is why human beings consistently seek to automate work. History demonstrates that we successfully automate hardware. With fairly well-understood knowledge of software, creating a comprehensive system to produce software is not impossible. Can you imagine that programmers no longer programming in the future world? In this world, code generators will generate software. There will be a role called "code alchemist" who is responsible write the recipe for the code generator. The recipe will dictate what kind of software people want. In other words, the recipe is also a software requirement document that the code generator can understand. Once the code generator receives the recipe, it will automatically produce software artifacts. The described above is revolutionary if there is such a code generator, and the Drasil framework could be it.

The Drasil is a framework that generates software, including code, documentation, software requirement specification, user manual, axillary files, and so on. We call those artifacts software artifacts. By now, the Drasil framework targets generating software to overcome scientific problems. Recently, the Drasil team has been interested in expanding its knowledge to solve a high-order ordinary differential equation (ODE) through external libraries. There are two main reasons why we want to do that.

1. Scientists and researchers frequently use ODE as a model in scientific problems, and this model describes the nature phenomenons. Drasil is a framework able to generate software artifacts that solve scientific problems. Therefore, solving ODE in the Drasil framework would solve many scientific problems.

2. There are many reliable libraries, and the Drasil team is interested in how the Drasil framework interacts with external libraries. Libraries that solve ODEs are probably the most important ones. Once the team understands how to interact between the Drasil framework and external libraries, they will start to add more

external libraries. In this way, it would unlock the potential to allow the Drasil framework to solve more scientific problems than before.

However, the Drasil framework neither captures ODE knowledge nor solves high-order ordinary differential equations. The previous researcher researched to solve a first-order ODE, but it only covers a small area of the knowledge of ordinary differential equations. Adding high-order linear ODEs into the Drasil framework will expand the area where it has never reached before. Therefore, this research will incorporate high-order linear ODEs in a complex knowledge-based, generative environment that can link to externally provided libraries.

In order to solve a high-order ODE, we have to represent equations in the Drasil database. Then, we need to know how external libraries solve ODEs, what their capabilities are, and what interfaces look like. Last, we need to bridge the gap between the Drasil ODE data representation and external libraries. We can achieve that by generating proper interfaces.

Before conducting this research, the Drasil framework can solve explicit equations and numerically solve a first-order ODE. After this research, the Drasil framework will have full capability to solve a high-order linear ODE numerically. In addition, we will explore the possibility of solving a system of ODE numerically. We will introduce a new case study, the double pendulum. It contains an example that solves a system of non-linear high-order ODE.

Chapter 1 will cover how to represent the data of linear ODE in Drasil. Then, in Chapter 2, we will analyze external libraries. In Chapter 3, we will explore how to connect the Drasil ODE data representation with external libraries. Last, we will discuss a user's choice to solve ODE differently in this framework.

Chapter 2

Your Chapter Title

This is a sample chapter

If you need to use quotes, type it “like this”.

2.1 Referencing

These are some sample references to GAMYGDALA (**?**) from the `references.bib` file and state effects of cognition (**?**) from the `references_another.bib` file. These references are not in the same .bib file.

2.2 Figures

This is a single image figure (Figure **??**):

This is a multi-image figure with a top (Figure **??**) and bottom (Figure **??**) aligned subfigures:

Figure 2.1: This is a single figure environment

2.3 Tables

Here is a sample table (Table **??**):

A	$\longleftrightarrow$	B
C	$\longleftrightarrow$	D

Table 2.1: A sample table

2.3.1 Long Tables

A sample long table is shown in Appendix **??**.

2.4 Equations

Here is a sample equation (Equation **??**):

$$y = mx + b \tag{2.4.1}$$

(a) Figure 1

(b) Figure 2

Figure 2.2: A Multi-Figure Environment

Chapter 3

Conclusion

Every thesis also needs a concluding chapter

Appendix A

Your Appendix

Your appendix goes here.

Appendix B

Long Tables

This appendix demonstrates the use of a long table that spans multiple pages.

Col A	**Col B**	**Col C**	**Col D**
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D

Continued on the next page

Continued from previous page

Col A	**Col B**	**Col C**	**Col D**
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D