Object Orientated and Reflective Programming

Software Language Engineering
MSci Final Year Project

Project Plan

Thomas Wilkinson
Supervisor: Adrian Johnstone
Department of Computer Science
Royal Holloway, University of London

Abstract

Programming languages, both high and low level, are a fundamental and vital part of the computer science ecosystem. They allow us to communicate with machines effectively so we can set them tasks to carry out; this communication has allowed us to engineer complex systems or pieces of software that have contributed immensely to the modern world we live in today. Without the ability to speak to a computer in a language that it can translate and understand, whether it is one of the high-level languages that are commonly used today, utilising English words and phrases, or one of the lower-level languages such as Assembly, or even the punch cards that were taken by the very first computers to deliver instructions, we simply would not have a vast amount of the modern technologies and systems we use on a daily basis. It cannot be overstated how important communication is – after all it was language that allowed humanity to express ideas and share discoveries with each other, a core reason as to why we have developed so far and spread all across the world. In fact, software language engineering is a field that shares a lot with linguistics – the definition of syntaxes and grammars, different languages being better at expressing different ideas and the relative difficulty to learn each unique language to name a few similarities.

Computers are the ultimate multi-purpose tools, however as of yet they cannot think for themselves. They can perform wondrous feats, calculations that would take our organic brains generations to complete they can perform in mere seconds every time without fail, but without the initial instructions for the task, usually written in one or more high-level languages, they have no way of knowing what they are expected to do. Therefore, to work in the computer science industry in later life as I hope to do, previous knowledge of not just using programming languages but understanding how these languages are created and operate under-the-hood will be an invaluable asset.

Object-orientated programming is a fundamental paradigm within the software language engineering sphere. Simula was the first proper object-orientated language, and it introduced ideas that are immensely important to the modern world of programming; the concept of objects, classes and inheritance are commonplace within many languages today such as Java, C# and to a lesser extent, Python. However, it was not until Smalltalk, more specifically Smalltalk-72, that interest in object-orientated languages grew, as it took what Simula achieved and carried to it a smoother model. In fact, the term “Object-Orientated” came from Smalltalk. This is why I have decided to learn and incorporate Smalltalk into my project; it represents the philosophy well as everything is an object; there are no primitives or control structures. Smalltalk also incorporates the idea of reflectivity, or the concept of allowing a program to inspect and to modify its own source code and metadata during runtime. This paradigm is incredibly interesting, especially when applied to a CAD programming language, as it allows the user to dynamically add shapes and change their properties without having to re-run the program, which will make my CAD language much more intuitive to use.

With this project I hope to gain a greater understanding and appreciation for how object-orientated languages really function, as this will be great knowledge to go into the industry with due to object-orientated programming languages being so commonplace. Reflectivity, while not quite as widely utilised, is also a fantastic paradigm to have a deeper understanding for, due to society relying more and more on dynamic systems that must react to the situation at hand. I will also acquire vital general experience in producing a project, using a version control system and working independently, which I will be able to tell future potential employers to my benefit, as these skills are valued highly within the computer science industry.

First Term Milestones

Smalltalk Setup and Proof-of-Concept Programs

Start Date: 10/10/22

Delivery Date: 30/10/22

For the first milestone, I will be using the language Smalltalk, as it is a strictly defined object-orientated language, as well as being one of the first. I have chosen to use Pharo as my implementation of the Smalltalk language as it provides a language almost identical to Smalltalk as well as an environment to run the programs in. Pharo also allows the user to view and change all the objects within the encapsulation rules, which will help when demonstrating reflection. This will require me to install and learn both Smalltalk and Pharo, as well as code two example programs. The first of these will be a customer ordering system, where the customers, orders, products and retailers will be objects which will inherit attributes from one another, for example customers and retailers which will inherit fields such as phone number or address from a parent class, which will demonstrate the concepts and power of inheritance and object-orientated programming.

The second program will create and open a graphical window in Pharo, the properties of which will be viewed through the use of inspectors; the tool that Pharo uses to allow the inspection and manipulation of objects. This is done by reification; turning a metaobject into a normal object which can be changed. The program will then demonstrate reflection by editing properties of this window through accessing the instance variables, which will visually change the window while the program is running, providing a clear display of introspection and intercession, as the modifications are passed back to the runtime system.

Java Proof-of-Concept Program

Start Date: 31/10/22

Delivery Date: 7/11/22

A proof-of-concept program that utilises reflectivity programming within Java via the use of JUnit test cases and the reflect package shipped with the JDK. This program will create a zoo of animals, which are objects with descriptive attributes that can be inherited from a parent class, as well as an interface allowing for generic methods to be implemented individually to a certain animal. These objects will be able to be added to the zoo and will be able to have both their attributes and methods altered during runtime, as test cases will be added to create an object, change a field or change a method. There will also be functionality to inspect its own metaobjects for metadata, such as a fields data type. This program will show that it is possible to alter Java programs while they are running.

Report on Object-Orientated Programming and Reflection

Start Date: 17/11/22

Delivery Date: 24/11/22

A report on object-orientated programming and reflection, detailing what these concepts are by explaining objects, metaobjects, introspection, intercession and how objects interact with one another. This report will then focus on explaining how Smalltalk was designed to be purely object-orientated through the use of messages rather than functions and without primitives or control structures. It will also detail how inheritance of attributes works in Smalltalk, as well as reflection, where the program can look at its own structure, such as its parse trees and datatypes, and even modify them during runtime. The report will include screenshots from the proof-of-concept programs to aid in the explanation and provide examples of reflection, inheritance and object interaction and creation. The report will then go on to compare reflectivity in Smalltalk against Java, outlining how Java's static typing is not completely designed with reflective programming in mind, but also how JUnit test cases naturally use reflectivity to test code while it is running and so it is therefore made possible. Screenshots and snippets of code will be taken from the Java proof-of-concept program to aid the explanation.

Interim Report and Video

Start Date: 25/11/22

Delivery Date: 2/12/22

A report containing the report written as well as explanations of the concepts of object-orientated and reflective programming, backed up with screenshots and snippets of code from the proof-of-concept programs written in first term. It will also include an up-to-date bibliography that contains all the sources that were consulted during the production of the plan, as well as the project diary containing an accurate timeline of what was worked on and when. Evidence of testing and correct use of the GitLab will be provided. Finally, a video showing the deployment of the proof-of-concept programs written will be filmed and submitted.

Second Term Schedule

Start Date: 5/12/22

Delivery Date: 9/1/23

A schedule for the second term's project work with start and delivery dates, as well as a brief description outlining the final report and what it will contain. It will also include a more in-depth description of the UKCAD language and the features I plan to add.

Second Term Milestones

Final Report

Start Date: 9/1/23

Delivery Date: 24/3/23

A comprehensive and detailed report containing all reports and proof-of-concept programs written from first term with screenshots and technical details from these programs. It will talk about how these programs cover object-orientated and reflective programming and how they were useful in aiding the construction of my own language with specific examples of functionality that was used in the UKCAD language, such as JUnit test cases. It will also contain a write-up about UKCAD itself, including screenshots and explanations of interesting or important parts. Also, it will include a completed bibliography from the entire project.

Continuation of UKCAD language

Start Date: 9/1/23

Delivery Date: 24/3/23

The continuation and expansion of my UKCAD language which I created during the software language engineering course. This language allows the creation of shapes and provides functionality so that the user can use these shapes to create 3-dimensional objects. The language has its own syntax and interpreter, as well as a 360 degree camera. I will dramatically expand on this, adding many new functions and control structures such as the ability to create while and for loops through the interpreter and eSOS rules, so that the user can create more and more complex objects. The language will also be redesigned to allow for reflectivity, so that the user can add or edit shapes while the program is running through the use of JUnit test cases and the reflect package, meaning that the modelling of objects is made much more intuitive as the user will not have to rerun the program each time they wish to edit a shape.

UKCAD Implementation Timeline

- Redesign of UKCAD to incorporate reflectivity via JUnit test cases and the reflect JDK package
- Addition of while, for and other control structures – will follow both SOS and attribute-action specifications
- New features such as the ability to have shapes pass through one another and the functionality to create a custom shape through the use of the mesh method in JavaFx – will follow both SOS and attribute-action specifications

Risks and Mitigations

Time Management

Naturally with a project of this scope, my time will have to be planned to ensure I work as effectively as possible. Therefore, I have assigned start and delivery dates for each deliverable. However, there is a possibility that things will go wrong, such as a program taking me longer than I anticipated. To counteract this, I have decided to produce my proof-of-concept programs first and to write my reports after, as I believe that writing reports is much easier to plan timewise as compared to coding. I will also plan my deliverables more in-depth when I start them, to ensure I work efficiently.

Version Control System Mistake

During this project I will be using GitLab to manage my code and reports, making sure I utilise branches and Git commands properly. However, there is a chance that I may make an error such as merging an incorrect branch and deleting some of my work. Therefore, I will brush up on my Git commands, so that I have a firm grasp on when to push, commit, merge and create branches properly.

Forgotten Details

As this project is quite large, there is very little chance of me being able to remember all the details of what happened while working on the project, such as bugs that took me a while to fix or ideas that influenced a particular design decision. Therefore, I will keep a project diary. Even still, if this diary is not added to every time I do work on the project, I may still miss details. To counteract this, I will ensure I add to the diary whenever I work on the project, and I will start to write my final report while continuing my CAD language in second term.

General Rabbit-Holing

While coding, it is inevitable that I will take longer amounts of time to fix certain problems compared to others. This may result from my knowledge of tools such as ART, eSOS or Pharo being not as strong as they should be and could cause me to use my time inefficiently while fixing a certain bug. Therefore, I will ensure that I use the resources available to me so that I can debug my code as effectively as possible. I will also use TDD to make sure I can hunt down any bugs as soon as they appear, as this method will identify mistakes as quickly as possible through extensive testing.

Local Hardware Fault

Machines are not perfect and they can unexpectedly go wrong. Although it is impossible to predict, I will attempt to counteract this by using GitLab as my VCS and regularly committing my code and reports whenever I have worked upon them, so that if anything happens to my machine, my work will not be lost.

Learning Smalltalk Syntax and Semantics

Due to Smalltalk being only a purely object-orientated language, and also being relatively old, its syntax is unlike most modern programming languages I have encountered. Although this is good in terms of expanding my knowledge of software language engineering, it may take me a while to fully grasp how it functions. This, if not properly prepared for, will cost me valuable time which could be spent elsewhere. Therefore, I will start with Smalltalk early, ensuring I have time to read the documentation and to have a go at making a few programs.

Reflective Programming Side-effects

Although reflectivity within programming languages is interesting, it is also an incredibly powerful tool that makes normally illegal actions possible. Therefore, while attempting to implement the paradigm, especially in Java - a language that unlike Smalltalk is not fully built to work around it, I will have to be careful to ensure that it does not have unintended side-effects, such as effects on performance or leaving code dysfunctional. To do this, I will ensure that I am implementing the technique properly via reading documentation and being careful where and what I apply it to.

Bibliography

Software Language Engineering. Adrian Johnstone.

Textbook for the software language engineering module. Used for a general overview of object-orientated programming and other concepts which are wrote about in the reports. Also used as a guide for eSOS and ART.

Object-Orientated Programming. Tim Rentsch. Computer Science Department, University of Southern California. 1982.

Used as an introduction to Smalltalk and how it is object-orientated, as well as its history.

Smalltalk-80, The Language and Its Implementation. Adele Goldberg and David Robson. 1983.

Detailed textbook outlining Smalltalk and its syntax. Used as a guide to learn and understand Smalltalk to aid the creation of the programs written.

A Little Smalltalk. Timothy Budd. May 24 1985.

Another textbook which goes into detail about Smalltalk, its concepts, syntax and applications. Used alongside other resources to learn Smalltalk.

Pharo By Example 5. Stéphane Ducasse, Dimitris Chloupis, Nicolai Hess, and Dmitri Zagidulin. September 29, 2018.

Textbook explaining Pharo, its environment and virtual machine. Provides lots of examples of Pharo programs and used to learn Pharo, as well as understand reflection and inheritance.

Reflective Programming in Smalltalk. M. Denker and S. Ducasse. 2005.

Slides that provide a general overview of reflectivity in Smalltalk including meta programming and metaobjects. Used as an aid to understand reflection in Smalltalk.

Advanced Run-Time Reflection. Pavel Krivanek. May 27, 2020.
https://thepharo.dev/2020/05/27/pharo-features-advanced-run-time-reflection/

Used to explain inspectors within Pharo for the second proof-of-concept program.

Java Reflection in Action. Ira R.Forman and Nate Forman. 2005.

Textbook going over reflectivity in Java and various reflective methods for retrieving metadata and other information. Used to gain a greater understanding of how Java can utilise reflectivity.

Guide to Java Reflection. Baeldung. July 5 2022. https://www.baeldung.com/java-reflection

A guide on how to use JUnit test cases to provide reflectivity in Java. Used for the Java proof-of-concept program.