

Submitted in part fulfilment for the degree of MSc in Software Engineering.

Model-driven software migration between microcontrollers

Sophie Wood

Version 1.1, 2018-September-03

Supervisor: Simos Gerasimou

Number of words = 16478, as counted by wc -w.
This includes the body of the report, but not the appendices.

Abstract

Software becomes obsolete when it is no longer compatible with an updated system. For example, moving a system to a new hardware platform or using an alternative data format can obsolete the existing software. Business domains with long software life-cycles such as avionics/military and other “safety-critical” areas often face obsolescence problems as hardware parts are discontinued before the system is complete. This necessitates adaptation of the system to new components which obsoletes the software. Model-Driven Engineering (MDE) techniques use domain models to simplify software development and has been shown to work well in industry when applied to software migration projects.

This project aims to tackle the particular software obsolescence problem of migrating software between hardware platforms - in this case the Parallax Propeller Activity Board and Arduino Uno microcontrollers. In particular, the project utilises MDE to achieve this goal. Three main components have been developed to support migration. Firstly, a Fritzing netlist analysis component analyses a diagram of the target platform in order to calculate pin variables for the generated code. Secondly, an Arduino library suggestions component aids in mapping between source and target platform libraries. Finally, a Parallax source code analysis component is used to identify methods defined in source platform libraries and to generate empty methods for the target platform.

The project is evaluated by testing the functionality of these three main components and performing case studies involving the migration of a sample application to the target platform.

Acknowledgements

I would like to thank Simos Gerasimou for his guidance and support throughout this project.

Statement of Ethics

This project did not use any human participants.

Contents

1	Introduction	13
1.1	Background and Motivation	13
1.2	Project Goals	15
1.3	Project Scope	15
1.4	Report Structure	16
2	Literature Review	17
2.1	Existing Work	17
2.1.1	Approaches to Software Obsolescence	17
2.1.2	MDE Approaches to Software Obsolescence	18
2.1.3	Migration Between Microcontrollers	21
2.2	Model Driven Engineering	22
2.2.1	Domain-Specific Modelling Languages (DSMLs)	22
2.2.2	Model Transformations	22
2.2.3	The Epsilon Framework	23
2.3	Microcontroller Background	24
2.3.1	Parallax Propeller Activity Board	25
2.3.2	Arduino Uno	26
2.4	Fritzing	27
3	Methodology	28
3.1	Overview of Methodologies	28
3.1.1	Traditional Methodologies	28
3.1.2	Agile Methodologies	28
3.2	Choice of Methodology	29
3.3	Project Management	30
4	Requirements	32
4.1	Requirements Elicitation	32
4.1.1	Project Statement	32
4.1.2	Stakeholders	32
4.2	Stakeholder Requirements	33
4.3	System Requirements	33
4.4	Requirements Traceability Matrix	37

Contents

5 Design and Implementation	38
5.1 High-Level Architecture	38
5.1.1 Overview	38
5.1.2 Design Rationale	39
5.2 Fritzing Netlist Analysis	40
5.2.1 Design	40
5.2.2 Implementation	41
5.3 Arduino Library Analysis	43
5.3.1 Design	43
5.3.2 Implementation	44
5.4 Parallax Project Analysis	46
5.4.1 Design	46
5.4.2 Implementation	47
5.5 Supporting Migration Between Other Platforms	48
5.6 Current Issues	49
6 Evaluation	51
6.1 Test Plan	51
6.2 Test Cases	51
6.2.1 Fritzing Netlist Analysis Test Cases	51
6.2.2 Arduino Library Suggestion Test Cases	54
6.2.3 Parallax Source Code Analysis Test Cases	56
6.3 Case Studies	57
6.3.1 Case Study 1 (CS-1)	57
6.3.2 Case Study 2 (CS-2)	62
6.3.3 Case Study 3 (CS-3)	62
6.4 Evaluation of Modifiability	63
6.5 Test Summary	64
6.5.1 Current Issues	64
6.6 Validity	64
6.6.1 Internal Validity	64
6.6.2 External Validity	65
6.6.3 Construct Validity	65
7 Conclusion	69
7.1 Further Work	70
Appendices	74
A Fritzing Netlist Analysis Test Inputs	75
A.1 Test Case 1 Netlist	76
A.2 Test Case 2 Netlist	79
A.3 Test Case 3 Netlist	83

Contents

B Parallax Source Code Analysis Test Inputs/Outputs	88
B.1 Test Case 1 Parallax Source Input	89
B.2 Test Case 1 Parallax Source Output	91
C Case Studies Inputs/Outputs	93
C.1 CS-1 Output	94
C.2 CS-1 Migrated Code	96
C.3 CS-2 Input	99
C.4 CS-2 Output	100

List of Figures

2.1	The general model-driven migration process used by Sodifrance [10].	19
2.2	The stages involved in the “Direct Transformation Approach”.	19
2.3	Project migration cost as a function of its size [10].	20
2.4	The relationship between the key concepts in MDE [15].	23
2.5	Features of the Parallax Propeller Activity Board [22].	25
2.6	Features of the Arduino Uno [23].	26
2.7	An example Fritzing diagram.	27
2.8	Part of the netlist generated for Figure 2.7.	27
3.1	An overview of the iterative, adaptive and extreme agile methodologies [26].	29
3.2	GANTT chart for the project.	30
3.3	Trello board for the project.	31
5.1	A high-level overview of the approach.	39
5.2	A more detailed view of the Fritzing netlist analysis component of the system.	40
5.3	The EMF representation of the Fritzing metamodel.	41
5.4	A graphical representation of the model conforming to the metamodel shown in Figure 5.3 for the circuit shown in Figure 2.7.	42
5.5	Structure of the configuration file.	42
5.6	A more detailed view of the Arduino library analysis component of the system.	44
5.7	How the library suggestions are displayed to the user.	45
5.8	A more detailed view of the Parallax project analysis component of the system.	46
5.9	An example of a partially expanded AST for Parallax source code.	47
5.10	An example of the empty method generated from a Parallax library function.	48
6.1	Fritzing diagram for Fritzing netlist analysis test case 1.	52
6.2	Output of Fritzing netlist analysis test case 1.	52
6.3	Fritzing diagram for Fritzing netlist analysis test case 2.	53
6.4	Output of Fritzing netlist analysis test case 2.	53
6.5	Fritzing diagram for Fritzing netlist analysis test case 3.	53
6.6	Generated pin variables.	54
6.7	Output of Arduino library suggestion test case 1.	55

List of Figures

6.8	Output of Arduino library suggestion test case 2.	55
6.9	Output of Arduino library suggestion test case 3.	55
6.10	Parallax Propeller Activity board setup for CS-1.	58
6.11	Arduino Uno setup for CS-1.	59
6.12	Compile error due to header macro definition.	60
6.13	AST for the declaration using a macro definition. The value is resolved to a literal expression (highlighted in red).	61
6.14	Compile error due to no return.	61
6.15	First error thrown when performing migration using empty configuration file.	63
6.16	Second error thrown when performing migration using empty configura- tion file.	63

List of Tables

4.1	Stakeholder Requirements	34
4.2	Functional Requirements	35
4.3	Non-Functional Requirements	36
4.4	Requirements Traceability Matrix	37
6.1	Case Study System Test Results	66
6.2	Case Study Functional/Non-Functional Test Results	67

1 Introduction

1.1 Background and Motivation

Obsolescence is defined by Bartels et al. as “materials, parts, devices, software, services and processes that become non-procurable from their original manufacturer or supplier” [1]. Software becomes obsolete when a system is updated such that the original software is no longer compatible. Hardware obsolescence is principally driven by the lack of part availability.

Issues with hardware obsolescence have been driven by the growth of the electronics industry. This has reduced the life cycle of electronic parts as competitors release products with better functionality and features. Existing products are no longer commercially viable and therefore go out of production [1]. This is a particular issue in the defence/aerospace sectors as the typical life cycle of a system is 20-30 years or longer [2]. As such, parts often become unavailable before the system is completed. Singh et al. found that these systems “often encounter obsolescence problems before they are fielded and always during their support life” [3].

Obsolescence is also a concern in the software industry. Businesses must continuously update their products in order to stay ahead. Bill Gates has said:

“The only big companies that succeed will be those that obsolete their own products before someone else does.” [1]

There are three main causes of software obsolescence: logistical, functional and technological. Logistical causes include digital media obsolescence and formatting or degradation limiting/terminating access to software. Functional causes include hardware, requirements, or other software changes to the system obsoleting the functionality of the software. Finally, technological causes include the termination of sales and/or support for commercial off the shelf (COTS) software [4].

In addition to these sources, software obsolescence can be driven by hardware obsolescence and vice versa. For example, the obsolescence of a hardware component can drive the move to a new platform for a software, thus obsoleting that software.

A further issue contributing to software obsolescence is a lack of skills. For example, there may be a limited number of people that are competent in the language the system is written in. If these people leave the company, the system can no longer be maintained [5].

Again avionics/military and other safety-critical systems particularly struggle with software obsolescence issues as even small changes may have to go through extensive and costly qualification/certification processes [3]. Consequently, software obsolescence

1 Introduction

costs can equal or exceed that of hardware [4]. Despite this, strategies for obsolescence mitigation/management have generally focused on hardware obsolescence problems. It is important that both hardware and software obsolescence issues are tackled as these problems can be incredibly costly to these industries. For example, the US Navy estimates that obsolescence problems can cost up to \$750 million annually [6]. Sandborn and Myers also found that sustainment costs (which include costs related to obsolescence) dominate the system costs in the case of development of an F-16 military aircraft [7].

Current strategies for mitigating software obsolescence issues are insufficient. Pro-active measures (e.g. improving code portability or using open-source software) often require either a large amount of resources or resources that are unavailable. Reactive strategies for tackling obsolescence (e.g. software license downgrades, source code purchase or third party support) may not always be possible. When these methods are inadequate, the legacy system may have to be redeveloped or rehosted [4]. However, many such software modernisation projects are abandoned or not completed within the planned timescale/budget [8].

If redevelopment projects can be automated or even partially automated, they can more easily be completed on time. Code analysis and code-based transformations used in [9] partially automate the process of adapting software to work with different libraries as well as migrating software between hardware platforms. Although this approach is successful, there is still a lot of manual work required in mapping between libraries.

An alternative approach is to use model-driven software modernisation (MDSM) for (partial) automation of modernisation projects. MDSM is based on the use of model-driven engineering (MDE) principles such as models and transformations to facilitate the migration process (MDE and its application to software modernisation are discussed in more detail in Sections 2.1.2 and 2.2). For example, model-to-model transformations can be used to map a model of the legacy code to a model of the new system. A model-to-text transformation can then be applied to this new model to generate code for the new version of the system.

MDSM is a promising approach as it has already been proved feasible in industry by the Sodifrance¹ company. For example, they were able to migrate a large-scale banking system (at a size of around one million lines of code) from Mainframe to J2EE [10]. Kowalczyk and Kwiecińska have also demonstrated the viability of MDSM by modernising a project written in Java 1.4 and the Hibernate 2.x framework into Java 1.5 and Hibernate 3.x [8]. One benefit of MDSM is that tools (e.g. model discoverers) can sometimes be reused between projects making subsequent modernisation projects quicker and cheaper to complete. For example, MoDisco is an open-source model-discoverer that can generate models from Java code [11].

The Defence Science and Technology Laboratory (DSTL) at the Ministry of Defence (MoD) have identified several instances of software obsolescence they would like to tackle. A problem of particular interest is “the migration of an entire software system

¹<https://www.sodifrance.fr/>

from a legacy hardware platform to a modern more powerful platform” [9]. This report will explore how MDSM processes can be used to partially automate the migration of code between microprocessors. A case study implementing the migration of code between a Parallax Propeller Activity Board and an Arduino Uno will be used to demonstrate the approach developed.

1.2 Project Goals

The main goal of the project is to demonstrate how MDE may be used to support migration between microcontrollers. In particular, the project aims to:

- Design the architecture of an approach that is able to support the migration of source code from one platform to another for a particular source-target platform pair.
- Provide an implementation of this system as an Eclipse plugin to improve usability.
- Design the system such that it is easily modifiable to support migration between alternative platforms.
- Evaluate the success of the system in supporting code migration by performing a case study that migrates a sample application from the source to target platform.

1.3 Project Scope

The project is intended to support migration of software from a Parallax Propeller Activity Board to an Arduino Uno in order to demonstrate an approach to software migration that may be applicable to more platforms. The approach will support migration of code that has a functional equivalent on the target platform e.g. the system will not attempt to migrate code that uses the multi-core processor of the Parallax Propeller Activity board to the single-core Arduino Uno. The designed approach will be implemented as an Eclipse plugin that will take as input the code for a Parallax Propeller Activity board application and produces code that, when completed by a programmer, can be run on an Arduino Uno with the same functionality as the Parallax source code. The project will be evaluated by performing a case study of this migration process.

The project was able to produce a system consisting of three main components: a Fritzing netlist analysis component, an Arduino libraries suggestion component and a Parallax source code analysis component. The Fritzing netlist analysis component is used to generate pin variables for the target platform given the configuration of the Arduino Uno’s circuit. The Arduino libraries suggestion component is used to suggest possible replacement libraries for the Parallax libraries of the source code. Finally, the Parallax source code analysis component is used to identify the methods belonging

1 Introduction

to Parallax libraries and create corresponding empty methods (to be completed by developers) in the generated code.

The project was evaluated by a combination of test cases, to ensure the individual components work as expected, and case studies to validate that the requirements outlined in Chapter 4 were met. The evaluation demonstrated that the majority of the requirements were met and the main components worked as expected. In particular, an application involving the use of servos, infrared LEDs/sensors and a humidity/temperature sensor was able to be migrated to the Arduino Uno. However, further work needs to be done to improve handling errors of the system and migrating additional constructs (such as macro definitions) from the source to target platform.

1.4 Report Structure

The project report begins with an overview of MDE techniques and the existing research into software obsolescence solutions in Chapter 2. This chapter also contains an introduction to the approach used to tackle software obsolescence in this project as well as an overview of the platforms for which the system will aid in software migration. This is followed by a discussion of the software development methodology used for the project in Chapter 3. Chapter 4 summarises the requirements for the project as well as the elicitation method. The design and implementation of the system is contained in Chapter 5 and is evaluated in Chapter 6. Finally, Chapter 7 summarises the project and discusses areas for future work.

2 Literature Review

This chapter introduces the existing work on software obsolescence and the technical background required to understand this project. It begins with an overview of the existing approaches for software obsolescence mitigation in Section 2.1.1 and in particular, MDE-based strategies in Section 2.1.2. The specific problem of migration between microcontrollers and the existing methods for facilitating this are discussed in Section 2.1.3 as well as how the MDSM-based method explored in this project is related.

Sections 2.2 and 2.3 contain the technical background of the project including a brief overview of MDE and the chosen framework for this project as well as a description of the microcontrollers used for the case study.

2.1 Existing Work

2.1.1 Approaches to Software Obsolescence

Rojo et al. identify some of the main methods of tackling obsolescence in their paper “Obsolescence management for long-life contracts: state of the art and future trends” [2]. Although they identified many strategies for handling the obsolescence of electronic components, very few methods were relevant for dealing with software obsolescence.

The tools “se-Fly Fisher” and “R2T2” (Rapid Response Technology Trade) were identified as useful for managing software obsolescence. These tools are used for design refresh planning. Design refresh planning deals with obsolescence in a proactive manner by planning the best times for performing a redesign of the system during the sustainment stage of its lifecycle.

Se-Fly Fisher uses the technology curves of each part of the system to: forecast how often a system baseline should change; identify replacement resources and estimate the benefit of each system baseline change [2].

Similarly, R2T2 can: forecast system obsolescence; allow for comparisons to alternative solutions; produce a life cycle obsolescence plan for the elements that require refreshment and plan when element replacements should occur [12].

These forecasting tools may enable organisations to plan ahead and handle cases where equivalent parts/software are available. However, they cannot assist in the actual redesign or redevelopment process which can be time consuming and costly.

Similarly, Sandborn et al. identified very few approaches towards managing software obsolescence. The main methods they identified for mitigating software obsolescence were [4]:

2 Literature Review

(1) Software License Downgrade: users can purchase licenses for the current product and apply them to older versions.

(2) Source Code Purchase: customers purchase the source code for the product.

(3) Third Party Support: a third party is contracted to maintain support for the software.

When possible, these approaches could reduce costs as the software does not have to be maintained in house or redeveloped. However, sometimes these methods may not be possible. Additionally, given that the system owners are so dependent on the legacy code, it puts them in a poor position for negotiating the prices for these contracts and so this approach could become costly.

2.1.2 MDE Approaches to Software Obsolescence

More recently, MDE approaches have been used to tackle software obsolescence problems where legacy code must be redeveloped. MDE allows some of the stages in migration to be automated, consequently reducing the timescale and costs involved.

The company Sodifrance has successfully been using MDE for development and migration projects for over ten years [10]. The general MDE approach to migration used by Sodifrance is shown in Figure 2.1. The approach is separated into four stages as follow:

1. Firstly, the code of the legacy application is parsed in order to create a model of the legacy system. In Figure 2.1, L indicates the metamodel for the implementation language of the legacy code.
2. This model is then transformed to a Platform Independent Model (PIM) conforming to an ANT metamodel¹. This model represents a high-level view of the legacy code. This process is dependent on knowledge of the libraries and coding conventions used by the legacy platform.
3. Next the PIM has a model transformation applied to produce a Platform Specific Model (PSM) conforming to a UML metamodel. The high-level views from the PIM are adapted to fit the target platform.
4. Finally, code is generated from the PSM by using template-based text generation tools.

¹An ANT metamodel contains packages to represent:

- Static data structures (close to the UML class diagram).
- Actions and algorithms (it includes an imperative action language).
- Graphical user interfaces and widgets.
- Application navigation.

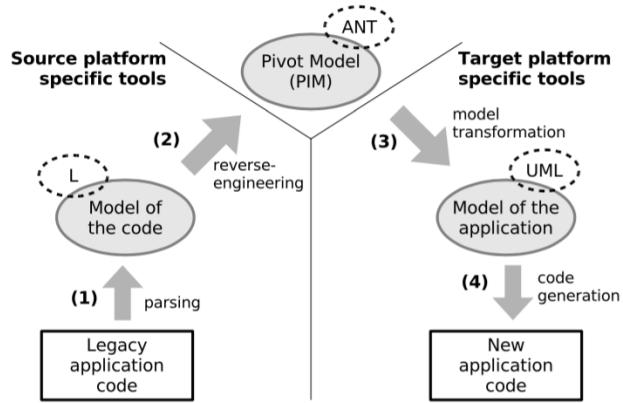


Figure 2.1: The general model-driven migration process used by Sodifrance [10].

The first two steps of the process can usually be fully automated. However, the remaining stages need some manual effort. Tasks that could not be completed automatically are indicated in the generated code (e.g. by TODO directives in Java applications) and summarised into a task list in order to allow the manual process to be completed more efficiently.

Sodifrance demonstrated the effectiveness of their approach using a case study of migrating a large-scale banking system from Mainframe to J2EE.

Kowalczyk and Kwiecińska have also explored the use of MDE in software migration [8]. They identified two main approaches: the “Reverse MDA approach” (RMA) and the “Direct Transformation Approach” (DTA).

The RMA approach follows the same stages as that used by Sodifrance. The DTA approach shown in Figure 2.2 reduces the number of stages involved in the transformation process by directly transforming between platform specific models.

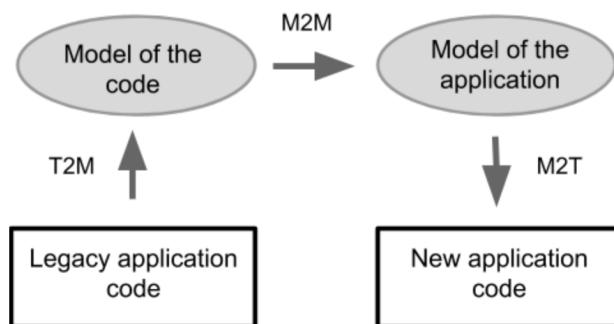


Figure 2.2: The stages involved in the “Direct Transformation Approach”.

2 Literature Review

The feasibility of both approaches was demonstrated by a case study migrating from Java 1.4 and the Hibernate 2.x framework to Java 1.5 and Hibernate 3.x. Based on qualitative analysis of the two approaches, they recommend to use an approach similar to DTA as often, a transformation from the legacy code model to the PIM will not exist. It is especially suited to cases where the code model and application model use the same metamodel.

MDE approaches to software migration have many benefits. The primary reason is the ability to automate some of the stages in the migration process, consequently reducing the cost and time scale of projects. Secondly, parts of the process (e.g. model discoverers/transformations, knowledge of migration patterns) can be reused between different migration projects which will again cut the cost of migration.

There are however, drawbacks to the MDE approach. There is a high cost of entry to start using MDE methods — the initial development of processes and tools can be time consuming and expensive if open-source tools are not available [10]. Secondly, a particular issue in commercial projects is that no code will be available until the initial analysis and tool development stage of the process is completed. In the case of the Sodifrance case study, code was not delivered until 10 months after the project had begun [10]. This can make customers nervous and reluctant to try this approach. On the other hand, once the initial stage is completed, the delivery rate of the code can be much faster than for a standard re-development approach.

One of the main considerations that must be made when choosing an MDE approach is the size of the project. If the project is small, it may be more expensive to use this method as the initial stages may take longer than manually migrating the code [10]. This is demonstrated in Figure 2.3.

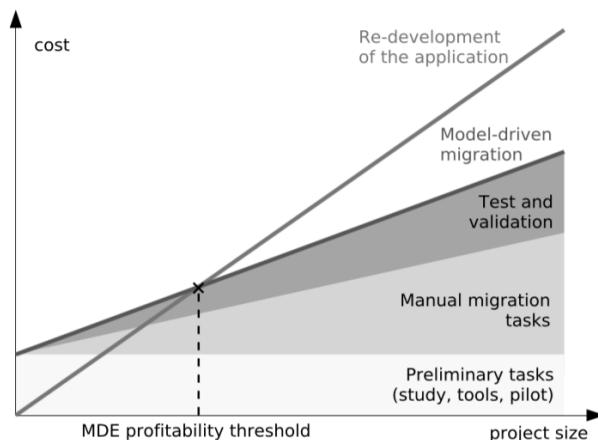


Figure 2.3: Project migration cost as a function of its size [10].

2.1.3 Migration Between Microcontrollers

As mentioned previously, a particular problem of interest to the MoD is “the migration of an entire software system from a legacy hardware platform to a modern more powerful platform” [9].

Atmel addressed the issue of migrating code between microcontrollers by reducing the learning curve for developers moving from development on 8-bit to 32-bit platforms. They aimed to achieve this by providing powerful debug facilities and development tools similar to established 8-bit development platforms [13].

Although this strategy should reduce the migration effort, the code will still have to be migrated manually which is error-prone and difficult. Additionally, the development of new tools adds extra overhead to the project, although these tools are reusable between projects.

Another approach for migrating code between microprocessors combines code analysis, code-based transformations and verification/validation techniques [9]. The procedure follows the following steps:

- (1) **Software system analysis (automated):** The source code and obsolete libraries are parsed to obtain abstract syntax trees (ASTs). The AST can be examined to indicate the elements using the obsolete library and help establish the system’s dependency level.
- (2) **Discovery of similar libraries (non-automated):** The development team identifies candidate libraries for replacement of the obsolete library.
- (3) **Compatibility analysis of the discovered libraries (non-automated):** Candidate libraries are rejected if they don’t conform to technical or semantic requirements.
- (4) **Data visualisation (automated):** Used to analyse the software system and its coupling with the obsolete library.
- (5) **Execute generation transformations (automated):** Firstly, an abstraction layer is generated that has the same usage behaviour as the obsolete library. Next the obsolete library usages are replaced with this abstract layer.
- (6) **Mappings inference (non-automated):** A developer creates a list of mapping rules by inspection of the obsolete and replacement libraries.
- (7) **Code population (non-automated):** A developer uses the mapping rules generated in stage 6 to populate the abstraction layer generated in stage 5.

This method was demonstrated to succeed in partially automating the migration of software between an Arduino and Raspberry Pi.

By automating some of the migration stages, the overall time and cost of software modernisation should be reduced. Another benefit is that the code analysis stages allow for potential risks to be detected early on. However, there is still a lot of manual effort required for migrating between libraries and in particular, generating mapping

2 Literature Review

rules requires a developer with a strong understanding of both the new and obsolete libraries.

My project aims to explore an alternative approach for migrating code between microcontrollers by applying an MDE approach similar to that discussed in Section 2.1.2. Consequently, I aim to (partially) automate the migration process and therefore reduce the time taken for migration.

2.2 Model Driven Engineering

Model Driven Engineering (MDE) was developed in order to address complexity within the problem space of computing [14]. In the past, approaches for enabling programmers to develop code more easily have focused on simplifying the solution space. This was achieved by providing abstractions such as higher-level programming languages or providing operating systems to manage the difficulty of programming hardware directly. However, these solutions are unable to express domain concepts effectively, unlike MDE techniques [14]. Furthermore, it has been found that MDE can also be utilised in automating the software development process [15].

The rest of this section introduces the terminology and main components of MDE as well as the MDE framework I will use in the project.

2.2.1 Domain-Specific Modelling Languages (DSMLs)

Meta-meta-modelling mechanisms such as the Object Management Group's (OMG) Meta Object Facility (MOF) can be used to create modelling languages for a given problem domain [16]. The key components of the OMG's approach to DSMLs are [17]:

Models: A model is a simplification of a system built with an intended goal in mind.

Models should be easier to use than the original system. This is achieved by abstracting out details of the system that are unnecessary for the target model's intended purpose.

Meta-models: A meta-model is the explicit specification of an abstraction (i.e. model).

Meta-meta-models: A meta-meta-model is used to define meta-models. In particular, the OMG define the MOF (Meta Object Facility) which is a self-defined meta-meta-model.

The relationships between these MDE concepts are summarised in Figure 2.4

2.2.2 Model Transformations

There are three main categories of model transformation:

Model-to-model (M2M): M2M transformations are used to translate source models to target models. The source and target models can be instances of the same or different meta-models [18].

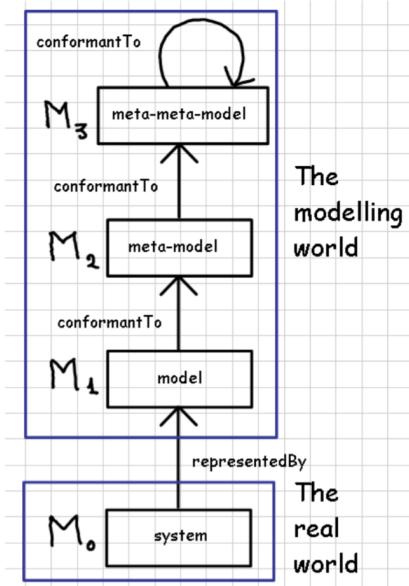


Figure 2.4: The relationship between the key concepts in MDE [15].

Model-to-Text (M2T): These can be considered a special case of M2M transformations.

It is often the case that M2T transformations are used for code generation. The most common approach to M2T transformation is a template-based approach. A template contains mixtures of static text and dynamic sections that can be used to access information from the source model [18].

Text-to-Model (T2M): T2M transformations can be used to transform code to a model (conforming to a language meta-model). Model discoverer tools such as MoDisco [11] are one of the simpler ways to perform T2M transformations on code [8].

2.2.3 The Epsilon Framework

Epsilon is a family of programming languages for model management tasks. It provides (among others) the following languages of interest [19]:

Epsilon Object Language (EOL): EOL is the common basis for the languages provided by Epsilon. It can be used for querying and modifying models. For example, selecting the elements of a model that fulfil a certain property or adding new elements to a model.

Epsilon Validation Language (EVL): EVL is used for model validation. EVL constraints (similar to OCL constraints) enable the model to be checked for errors as well as supporting additional functionality such as definition of custom error messages.

2 Literature Review

Epsilon Transformation Language (ETL): ETL is a rule-based language used for M2M transformations. For example, given an input model A that contains class elements and an output model B that contains table elements, ETL allows you to define a rule Class2Table that transforms the class elements to table elements in the output.

Epsilon Generation Language (EGL): EGL is a template-based language used for M2T transformations. EGL contains regions of dynamic code that can be populated with model data as well as static sections which will appear directly in the output.

Epsilon also enables the use of ANT tasks to create workflows of different tasks (e.g. a validation followed by a transformation followed by code generation) [19].

Epsilon has been chosen as the framework for this project over others for the following reasons:

- It is well documented with many tutorials as well as “The Epsilon Book” being freely available. Additionally, there is an active forum for any questions not answered by these sources [20].
- It is well integrated with Eclipse (for example, syntax/error highlighting is provided in the editor and there are graphical tools for running/debugging programs) and has been an official Eclipse project since 2006 [20].
- Epsilon provides a connectivity layer (EMC). This layer allows EOL programs to access models of different modelling technologies including Eclipse Modelling Framework (EMF) models and XML documents [20].
- I am already familiar with how to use the framework.

2.3 Microcontroller Background

Microprocessors consist of one or more CPUs, memory and input/output peripherals on an integrated circuit (IC). They are primarily used in embedded systems such as automobiles or medical equipment but are also used by hobbyists due to their low cost. Microcontrollers are popular since they can reduce the size of circuits by replacing multiple other ICs and increase flexibility as they can be reprogrammed [21].

The project involves migration of code from a Parallax Propeller Activity Board to an Arduino Uno in order to demonstrate an approach to migrating software between platforms. These microcontrollers were chosen as they are popular and their use is well-documented. The key features of these microcontrollers are highlighted and an overview of the board layout is provided in sections 2.3.1 and 2.3.2 below.

2.3 Microcontroller Background

The main differences that will need to be taken into account for software migration are:

- The Parallax Propeller Activity Board has a multi-core microcontroller whereas the Arduino Uno uses a single core microcontroller.
- The Parallax Propeller Activity Board only supports C whereas the Arduino Uno supports C and C++.
- The two boards have different numbers of digital/analog pins.
- The boards have different sized memories.

Code migration is a challenge as microcontroller architectures can vary widely. Different functions and libraries will be used depending on the specific architecture and so code cannot just be copy-pasted to the new platform even if it uses the same language. Instead functions from source platform libraries must be replaced with those from target platform libraries. Furthermore, the two platforms may demonstrate different hardware capabilities (e.g. single vs. multi-core processors or different sizes of memory). This also must be taken into account during migration.

2.3.1 Parallax Propeller Activity Board

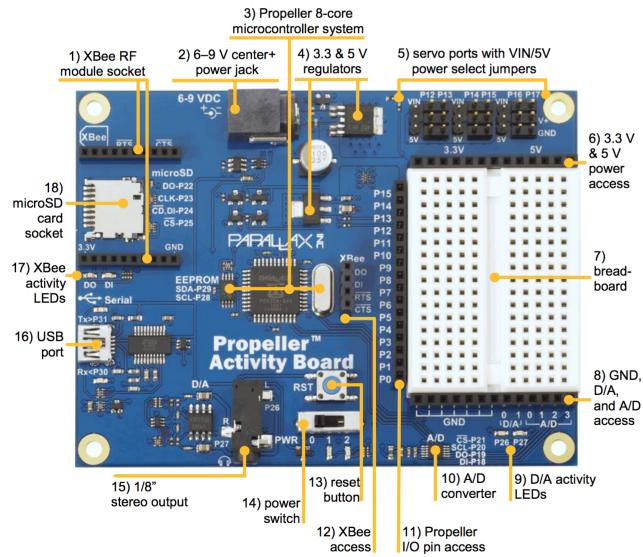


Figure 2.5: Features of the Parallax Propeller Activity Board [22].

The key features of the Parallax Propeller Activity Board are [22]:

- Built-in 8-core Propeller P8X32A microcontroller
- 64KB EEPROM

2 Literature Review

- XBee wireless module socket
- 16 digital I/O pins
- 4 Analog-to-Digital pins
- 2 Digital-to-Analog pins

2.3.2 Arduino Uno

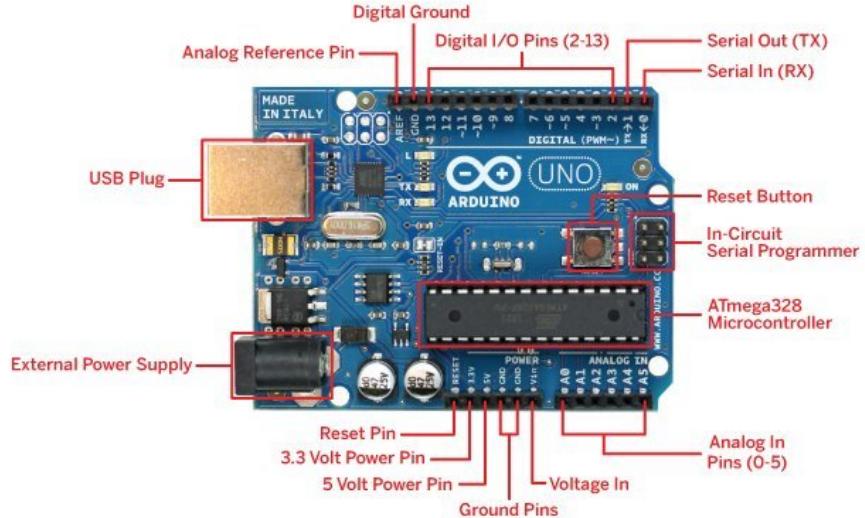


Figure 2.6: Features of the Arduino Uno [23].

The key features of the Arduino Uno are [24]:

- Built-in single-core ATmega328 microcontroller
- 32KB flash memory
- 2KB SRAM
- 1KB EEPROM
- 14 digital I/O pins (6 provide PWM output)
- 6 analog input pins

2.4 Fritzing

Fritzing² is open-source software that (among other uses) enables the user to create circuit diagrams and export a netlist³. Figures 2.7 and 2.8 show a Fritzing diagram and part of a netlist for that diagram.

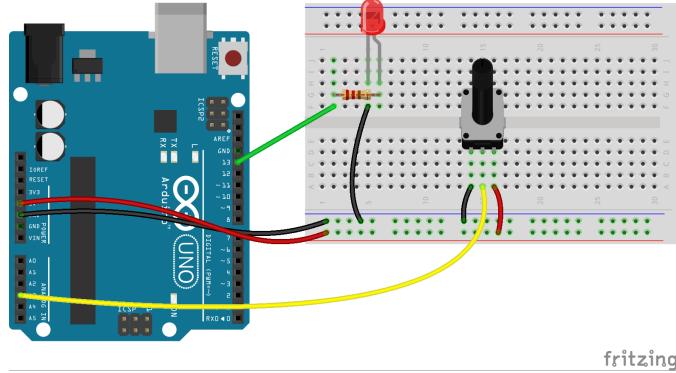


Figure 2.7: An example Fritzing diagram.

To handle differences in pin configurations between the source and target platforms as discussed in Section 2.3 a Fritzing netlist of the target board can be analysed in order to deduce the pin variables required in the migrated code (this is discussed in more detail in Section 5.2).

```

<net>
  <connector name="wiper" id="connector1">
    <part id="17619560" title="Rotary Potentiometer (Small)" label="R1"/>
  </connector>
  <connector name="A3" id="connector3">
    <part id="17626170" title="Arduino Uno (Rev3)" label="Arduino2"/>
  </connector>
</net>
<net>
  <connector name="leg2" id="connector2">
    <part id="17619560" title="Rotary Potentiometer (Small)" label="R1"/>
  </connector>
  <connector name="5V" id="connector87">
    <part id="17626170" title="Arduino Uno (Rev3)" label="Arduino2"/>
  </connector>
</net>

```

Figure 2.8: Part of the netlist generated for Figure 2.7.

²<http://fritzing.org/home/>

³An XML file encapsulating all the parts and connections in the circuit

3 Methodology

This section summarises the methodology and tools used for managing this project. It begins with an overview of project management methodologies in Section 3.1 and discusses the methodology chosen for this particular project in Section 3.2. Finally, the tools used for project management are discussed in Section 3.3.

3.1 Overview of Methodologies

Project management methodologies can be roughly categorised as either traditional or agile methodologies. Each have their own strengths and weaknesses and they are applicable to different types of systems and application domains. Sections 3.1.1 and 3.1.2 provide an overview of these methodologies.

3.1.1 Traditional Methodologies

Traditional approaches include the waterfall/linear methodology and the incremental strategy. The common features of these methodologies are that they are very structured and there is little room for deviation from the original plan. For example, the waterfall methodology splits the project into separate stages for software requirements gathering, systems requirements gathering, analysis, design, coding, testing and operations such that each section is completed before the next begins [25].

These strategies are best suited to projects that have a clear goal and clear solution. In this case, the use of these strategies can have many benefits. For example, the entire project is scheduled and resource requirements are known from the beginning. Additionally, this allows for team members to be distributed [26]. However, heavy documentation is required and a detailed plan is needed from the start of the project. As such, it is difficult to adapt this plan to changes in requirements [26].

3.1.2 Agile Methodologies

Agile methodologies are generally better at accommodating change than traditional methodologies as they repeatedly return to earlier stages in the development process. For example, Figure 3.1 shows a (simplified) view of the stages involved in the iterative, adaptive and extreme agile methodologies [26]. Each approach goes through the full development process, tests and/or system deployment and optionally returns to either the build, design or scope stage depending on the outcome of testing and deployment.

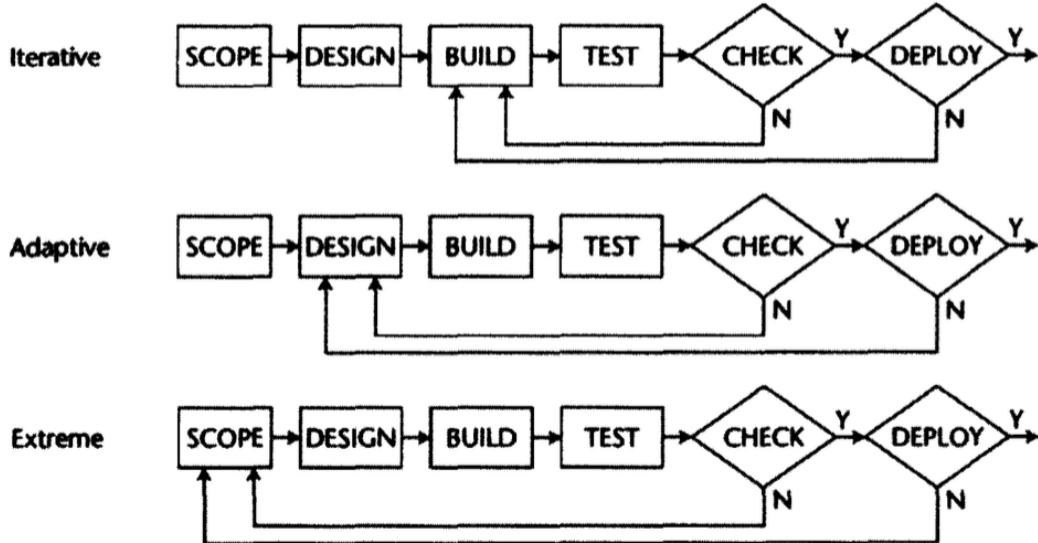


Figure 3.1: An overview of the iterative, adaptive and extreme agile methodologies [26].

This cycle can occur multiple times in the project lifecycle and is referred to as a sprint. Sprints generally last a few weeks but this can vary depending on the project.

As previously mentioned, the ability to return to earlier stages in the development process can help accommodate changing requirements. For example, the client can review what is produced at the end of each sprint and clarify or change requirements depending on the outcome. However, an agile approach requires much closer communication with the client which may not always be possible or desirable. Additionally, since requirements are not fixed for the duration of the project, the final product is not always clear from the start of the project [26].

3.2 Choice of Methodology

The approach for the project was fairly well defined i.e. an MDE-based method would be used to perform software migration. However, it was likely that the requirements would change throughout the course of the project depending on the success of system prototypes. As such, an agile methodology was chosen for project development as it could more easily cope with the potential changes to requirements. Additionally, as meetings were scheduled with the project supervisor on a weekly basis this would allow us to work on updating the requirements as necessary and enable the project to be easily split into sprints beginning and ending at these meetings. In particular, the extreme methodology was decided to be the most suitable as it would have allowed for changing the scope if early research indicated the scope was unsuitable for the time allocated.

In practice, this approach worked well when applied to the design, development

3 Methodology

and testing of the three main components of the project (see Chapter 5). The project ended up being roughly divided into six sections for the literature review, requirements gathering, development of the three main components and the project writeup. The literature review was conducted at the start of the project in order to gain the necessary background knowledge and enable the requirements to be defined. With these requirements in mind, development of the main components could begin. Each component was developed using sprints. The functionality of the component was designed, implemented and tested for basic cases and then refined in subsequent sprints. Use of the extreme methodology in these sprints allowed for the scope of the components to be changed based on discoveries during implementation. For example, during development of the Fritzing netlist analysis component (see Chapter 5) it was decided that it would only handle a subset of the parts in Fritzing as it would be outside of the timescale of the project to handle every part.

3.3 Project Management

This section discusses the tools used in order to manage the project. This includes tools for task management as well as version control tools for the source code and report.

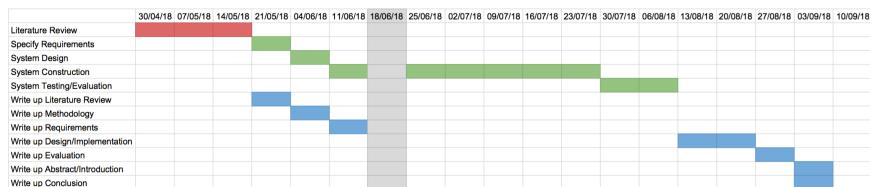


Figure 3.2: GANTT chart for the project.

At the start of the project a GANTT chart was created (see Figure 3.2) in order to give a rough estimate for when certain parts of the project would be completed. This was not updated throughout the project and doesn't particularly fit with the agile methodology but it was helpful to ensure progress was being made at about the right pace. In particular, this was helpful for writing the report.

The GANTT chart was included in the project page which was created using Google Sites. This page was used to contain all the information for managing the project. In particular it was used for organising the literature review but also links to the task management tool for the project. Google Sites was chosen as it is free and can be easily adapted to suit the project. The literature review was organised by creating pages on the site for each category within the literature review (e.g. MDE background) and collecting notes from the relevant papers on these pages. The site also contained a list of papers to be read and a link to the Trello board (see Figure 3.3) for managing the writing and coding tasks. Trello is used as it is free and makes it easy to prioritise and organise tasks.

3.3 Project Management

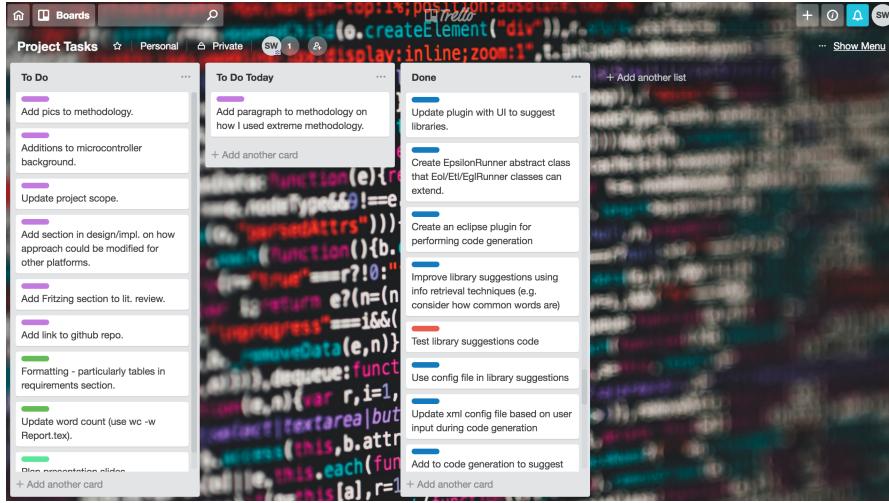


Figure 3.3: Trello board for the project.

Git was used as the version control system for both the source code of the system and the documentation as it is free and I already had experience using it. The remote repositories were hosted on GitHub as it allows students to use private repositories for free. In particular, Git was chosen for managing the documentation over e.g. Google Drive as the report was written in \LaTeX and so updates to the main report document could require changes to e.g. image files or the bibliography file. As such, it is easier to backup all these files at once using a single Git command rather than adding and deleting files from Google Drive. Additionally, the file history is saved using Git so it is easier to return to previous versions of the documentation.

4 Requirements

This chapter discusses the requirements that must be satisfied in order to consider the project successful. Firstly, Section 4.1 introduces the sources used to derive requirements including the relevant stakeholders for the project. Then Sections 4.2 and 4.3 summarise the stakeholder and system requirements respectively. Each requirement is assigned an identifier and a priority based on the MoSCoW categorisation system [27]. In decreasing order, the priority levels each requirement can have are **M** (must have), **S** (should have), **C** (could have) and **W** (won't have).

4.1 Requirements Elicitation

This section discusses the main sources considered for deriving requirements. Firstly, a summary of the initial project statement is presented in Section 4.1.1 as this is helpful for deriving the functional requirements for the project. Secondly, stakeholders for the project are identified in Section 4.1.2 in order to enable gathering of stakeholder requirements.

4.1.1 Project Statement

The initial project statement is as follows:

“This project aims to design and develop an extensible MDE-based infrastructure for migrating software applications between microprocessors. In particular, the project will enable:

- analysis and extraction of source code deployed on the old microprocessor
- migration of extracted source code to the new microprocessor”

4.1.2 Stakeholders

Stakeholders are parties that are interested in the development of the project and therefore their goals must be taken into account when deriving requirements. The rest of this section introduces the roles of the primary stakeholders and gives a high-level overview of their objectives for the project.

Student/Project Developer: Responsible for research, system design, development and testing. Primarily interested in demonstrating the efficacy of an MDE-based approach to software migration and successfully completing the project on time.

4.2 Stakeholder Requirements

Project Supervisor: Responsible for meeting regularly with the student and guiding the research for the project. Their goal is also for the system to be finished within schedule.

Software Obsolescence Researchers: Although not directly involved in the project, the results of the project may be of interest to researchers in the field of software obsolescence. In particular, the project explores an alternative approach to the migration of software between microcontrollers presented in [9]. This is a particular problem of interest to the DSTL. However, they are specifically investigating migration between their own microcontrollers such as the FS P4080. Therefore, they would require that the system is easily understandable and can be adapted to work with microcontrollers other than the Parallax Propeller Activity Board and the Arduino Uno.

End Users: No end users are directly involved in the project. However, some people may want to use the resulting system as is to migrate software from a Parallax Propeller Activity Board to an Arduino Uno. In this case, their main priority would be usability of the system. Additionally, it is important that the performance of the migrated software is not significantly worse than the source application. However, as this is a very specific scenario, it is unlikely that this user group will be very large. Similarly to software obsolescence researchers, it may be more useful to end users if the system can be easily modified to support migration between microcontrollers other than the Parallax Propeller Activity Board and Arduino Uno.

4.2 Stakeholder Requirements

Table 4.1 summarises the stakeholder requirements for the system. Each requirement has an identifier of the form S-X (where X is an integer).

4.3 System Requirements

The following section summarises the functional and non-functional requirements of the system. Each functional requirement is assigned an identifier of the form F-X (where X is an integer) and each non-functional requirement is assigned an identifier of the form NF-X (where X is an integer).

Note that valid source code refers to an application which compiles and runs correctly on the source microprocessor and only uses hardware capabilities provided by both microcontrollers.

Table 4.2 summarises the functional requirements and their acceptance criteria for the system. Similarly, Table 4.3 summarises the non-functional requirements and their acceptance criteria for the system.

Table 4.1: Stakeholder Requirements

ID	Priority	Description
S-1	M	Users can provide the system with the source code for a Parallax Propeller Activity Board and target code for the Arduino Uno will be generated.
S-2	M	Users can easily adapt the system to support migration of software between different microprocessors.
S-3	S	Users can easily determine sources of errors when using the system.
S-4	S	Users can easily identify areas of the target code that cannot automatically be migrated and must be manually completed.
S-5	S	The performance of the migrated code is of an acceptable level to the stakeholder.

Table 4.2: Functional Requirements

ID	Priority	Description	Acceptance Criteria
F-1	M	The target microcontroller running the migrated code (with manual additions) displays the same behaviour as the source microcontroller, provided the source code is valid.	Manual inspection of the migrated application confirms the behaviour is the same as the source application.
F-2	M	The migrated code deploys without errors on the target microcontroller if the provided source code is valid.	Manual inspection of the migration process confirms a valid source application can be migrated without errors.
F-3	S	The system displays appropriate errors when the source code cannot be migrated (e.g. there are errors in the source code or hardware incompatibilities).	Errors are thrown when invalid source code is provided to the migration process.
F-4	S	The system clearly indicates any errors that have occurred during its use (e.g. the migration process fails).	Errors are thrown if the migration process fails. The errors clearly indicate the source of the problem.
F-5	S	The system should indicate any parts of the code that must be manually completed.	Manual inspection of the migrated code confirms that all parts of the code that must be manually completed are clearly indicated (e.g. by TODO directives).

Table 4.3: Non-Functional Requirements

ID	Priority	Description	Acceptance Criteria
NF-1	M	The system should be easy to adapt to support migration between different microcontrollers.	Manual inspection of the system should indicate that it is easy to adapt. For example, if the system is based on an established MDE framework it will be easier to adapt as there will be more documentation available.
NF-2	S	The system should display a high-level of encapsulation.	Manual inspection of the system should indicate measures have been taken to facilitate encapsulation. For example, separation of hardware-dependent parts of the code.
NF-3	S	The migrated code should not be significantly larger than the source code.	Migrated code is of a size acceptable to the stakeholder.
NF-4	S	The migrated code should exhibit similar performance to the source code.	Migrated code runs in a duration acceptable for the stakeholder.

4.4 Requirements Traceability Matrix

4.4 Requirements Traceability Matrix

The requirements traceability matrix shown in Table 4.4 shows how stakeholder requirements correspond to system requirements.

Table 4.4: Requirements Traceability Matrix

Stakeholder Requirements	System Requirements
S-1	F-1, F-2
S-2	NF-1, NF-2, NF-3
S-3	F-2, F-3, F-4
S-4	F-5
S-5	NF-4

5 Design and Implementation

This chapter of the report discusses the decisions made in designing and implementing the system for supporting the migration of software between platforms. The system is implemented as an Eclipse plugin to assist in usability, reusability, extensibility and cross-platform compatibility. The plugin deals with the specific case of migration from a Parallax Propeller Activity Board to an Arduino Uno (these will be referred to as the source and target respectively). However, the approach is intended to be extensible to other platforms - Section 5.5 discusses what changes would need to be made to support migration between other platforms. The rest of the chapter explores the design and implementation of the system. Firstly, the high-level architecture of the plugin will be presented in Section 5.1 along with the design rationale. This will be followed by an examination of the design and implementation of these components in Sections 5.2 to 5.4. Finally, limitations of the approach are discussed in Section 5.6.

5.1 High-Level Architecture

5.1.1 Overview

Figure 5.1 shows the high-level architecture of the approach. There are three main components to the approach: Fritzing netlist analysis, Arduino libraries analysis and Parallax source code analysis. Each of these components produces information that contributes to code generation. The code generation is coordinated by an EGL transformation that takes the data from these components and generates a C file as output. This file acts as a basis for code that is compatible with the target platform.

The goal of the Fritzing netlist analysis component is to make it simpler to handle cases where the source and target platform have different pin configurations for a functionally equivalent circuit. For example, an LED may be attached to pin 12 on the source board and pin 6 on the target. Manually changing the references to this pin number can be error prone. Given a Fritzing netlist and a configuration file (containing additional information about Fritzing parts) this component calculates which pin of the board each part is connected to. This information is passed to the Code Generation EGL transformation script and is used to generate pin variables in the output file.

The goal of the Arduino libraries analysis component is to aid in mapping between source and target platform libraries. This is achieved by suggesting to the user suitable target platform libraries for the parts in the Fritzing netlist. Analysis is performed based on these parts and the Arduino header files to propose suitable libraries. Additionally, chosen libraries are recorded in the configuration file and this is used to inform the

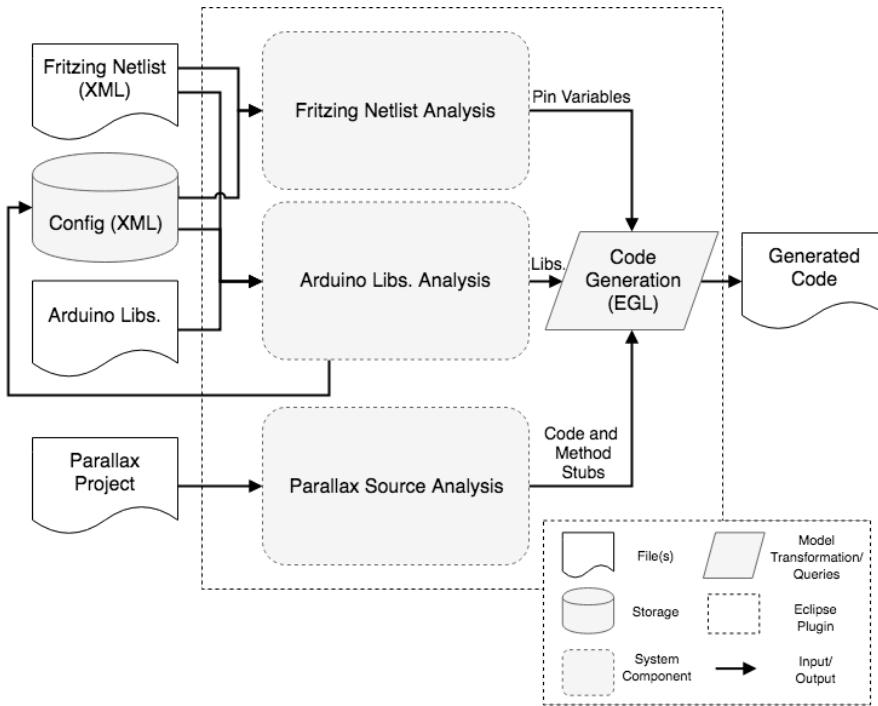


Figure 5.1: A high-level overview of the approach.

suggestions when the system is run subsequently. The libraries chosen by the user are passed to the Code Generation EGL transformation script and used to add suitable include statements to the output file.

Finally, the goal of the Parallax source analysis component is to take the code for the source platform and modify it to make it suitable for the target platform and indicate any areas of the code that need manual modification. Source-platform-dependent constructs are identified during the analysis and this information is passed to the Code Generation EGL transformation script. Code that does not need modification is passed as is. For example, the method signatures of source platform library function calls are identified in the analysis. These signatures are then used to generate empty method stubs in the output file with TODO directives indicating that they need to be populated by software engineers.

5.1.2 Design Rationale

Based on the requirements and stakeholders outlined in Chapter 4, reusability and modifiability were identified as the key quality attributes for the approach. Based on the potential usage of Software Obsolescence Researchers, it is important that the approach exhibits reusability so that successful components of the system are able to be used easily within other projects. As such, the system was designed to have loose coupling between components, in particular the three main components shown

5 Design and Implementation

in Figure 5.1. This diagram clearly shows each of these elements works independently of the others and so could easily be reused.

It is also crucial that the system displays a high degree of modifiability to support requirement S-2: “*Users can easily adapt the system to support migration of software between different microprocessors.*”. This is again supported by loose coupling between components. Another decision to support modifiability was to use a configuration file as input to the components analysing the Fritzing netlist and Arduino libraries. This file contains information that could have been hardcoded into each component. However, using a configuration file allows this to be easily replaced for supporting migration between different platforms.

5.2 Fritzing Netlist Analysis

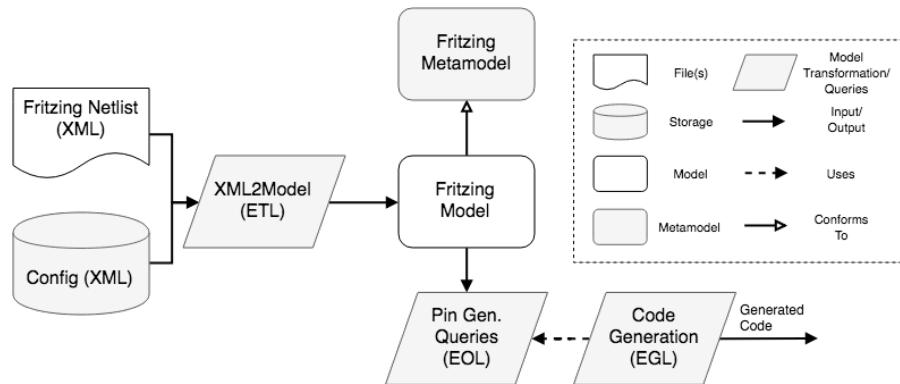


Figure 5.2: A more detailed view of the Fritzing netlist analysis component of the system.

5.2.1 Design

The Fritzing netlist analysis component first consists of an XML to model transformation to extract the key information from the netlist. EOL queries are applied to this model to extract the connections from the parts to the target board. The Code Generation transform uses these queries in generating the pin variables for the output.

Although the netlist could be queried directly, the resulting model contains the information in a format making the queries easier to write as well as combining information from the configuration file. For example, Appendix A.3 shows how large/complex a netlist is for the diagram shown in Figure 6.5. This adds extra processing time but makes the code easier to understand which is more important given the requirements for the system.

Similarly, the output model from the transformation script could have been queried directly from the Code Generation EGL transformation script rather than having a

separate EOL file to perform the queries. However, separating out the queries makes the logic in the EGL transform simpler as well as helping decouple the code generation from the implementation of the model/query logic. This improves the modifiability and reusability of this component.

5.2.2 Implementation

```

class Circuit {
    val Part[*] parts;
}

class Part {
    attr String partId;
    attr String title;
    attr Type type;
    val Connector[*]#parent connectors;
}

enum Type {
    Board;
    Input;
    Output;
    Other;
}

class Connector {
    attr String name;
    ref Connector connectedTo;
    ref Part#connectors parent;
}

```

Figure 5.3: The EMF representation of the Fritzing metamodel.

5.2.2.1 Model Transform and Metamodel Design

Figure 5.3 shows the format for the Fritzing metamodel shown in Figure 5.2. The metamodel captures the essential information required to resolve the connections from parts to the Arduino board. Models conforming to this metamodel represent the circuit as a graph where nodes are parts and edges represent connections between the connectors of these parts. A graphical example of a model for the circuit in Figure 2.7 is shown in Figure 5.4.

Most of the information for instantiating these models is extracted directly from the netlist. However, additional information about the type of parts is required. For example, when creating the pin variables in the output, we only want to generate values for devices read/written by the Arduino such as LEDs or potentiometers. Other parts, such as resistors, should not have pin variables generated since they will be accessed from the code. The configuration file (an example can be seen in Figure 5.5) contains mappings between Fritzing part names and the type of the part. The part type is used to indicate whether or not variables should be generated for this component in the

5 Design and Implementation

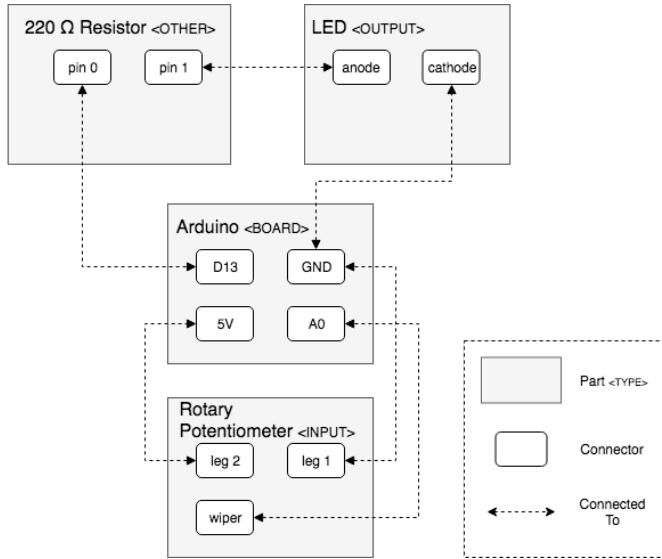


Figure 5.4: A graphical representation of the model conforming to the metamodel shown in Figure 5.3 for the circuit shown in Figure 2.7.

output file. This type information is stored as an attribute of the Part class in the model and is taken into account by the queries used to generate pin variables.

```

<parts>
  <!-- Output parts -->
  <part title="led" type="OUTPUT"/>
  <part title="7 segment display" type="OUTPUT"/>
  <part title="lcd" type="OUTPUT"/>
  ...
  <!-- Input parts -->
  <part title="potentiometer" type="INPUT"/>
  <part title="humidity and temperature sensor" type="INPUT">
    <library count="2" name="DHT.h"/>
  </part>
  <part title="rfid reader" type="INPUT"/>
  ...
  <!-- Boards -->
  <part title="arduino uno" type="BOARD"/>
  ...
</parts>
```

Figure 5.5: Structure of the configuration file.

5.2.2.2 Pin Generation Queries

To find out which pins of the board parts in the circuit are connected to, a modified breadth-first search is performed on the Fritzing model. For each of the I/O connectors on the board, the connections in the model are followed until an input/output part is found. For example, resolving the connection to digital pin 13 (D13) in Figure 5.4

follows the connection to pin 0 of the 220Ω resistor, sees that this is not an input/output part so follows the outgoing connections from this component (i.e. pin 1) to the anode of the LED. As this is a part tagged as an output type, a variable is then generated associating the LED with pin 13 on the Arduino board. Algorithms 1 and 2 below demonstrate the algorithm used for resolving the connections from the board to the input/output parts.

Algorithm 1 Calculate mapping from each pin on the board to its connected sensor(s)

```

1: procedure RESOLVEBOARDCONNECTIONS(board)
2:   for connector in board.connectors do
3:     connectedSensor  $\leftarrow$  bfsResolveConnections(connector)
4:     connections.put(connector, connectedSensor)
5:   return connections

```

Algorithm 2 Calculate set of connectors connected to *start*

```

1: procedure BFSRESOLVECONNECTIONS(start)
2:   queue.add(start)
3:   visited  $\leftarrow$  {}
4:   while not queue.isEmpty() do
5:     con  $\leftarrow$  queue.removeFirst()
6:     if not visited.includes(con) then
7:       visited.add(con)
8:       type  $\leftarrow$  con.parent.type
9:       if type == INPUT or type == OUTPUT then
10:        connectors.add(con)
11:      else
12:        queue.add(con.connectedTo)
13:        if not type == BOARD then
14:          for c in con.parent.connectors do
15:            queue.add(c)

```

5.3 Arduino Library Analysis

5.3.1 Design

The Arduino Library Analysis component is used to suggest target platform libraries for each part in the Fritzing netlist. There are two methods used for library suggestion:

Library Ranker: For each part, the Library Ranker component calculates a ranking for the suitability of each Arduino library (discussed in more detail in Section 5.3.2.1).

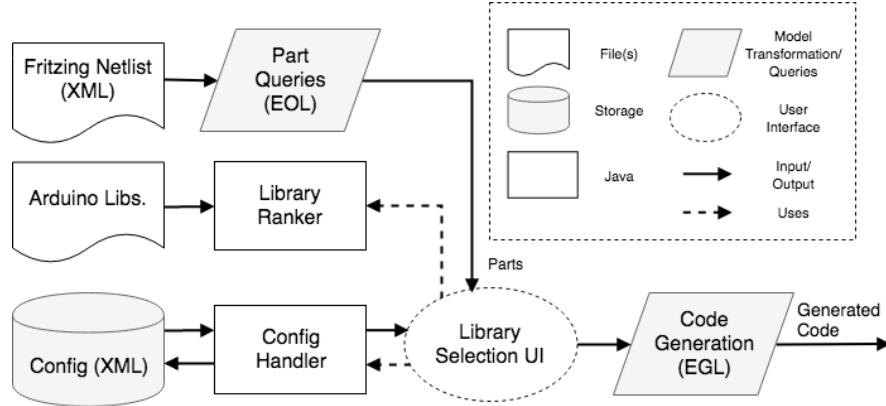


Figure 5.6: A more detailed view of the Arduino library analysis component of the system.

Configuration: The configuration file stores information about the previously used libraries for a given part. Whenever a library is chosen for a given part, the configuration file is updated with this choice.

The Library Selection UI combines the information from the Library Ranker component and configuration file and displays library suggestions to the user. Once the choices have been made, it also coordinates the update of the configuration file and passes on the library choices to the Code Generation EGL transform. As the only information passed to the Code Generation transform is a list of libraries, there is no separate EOL file used in this component unlike in the Fritzing netlist analysis component (e.g. the Pin Gen. Queries EOL file in Figure 5.2).

As these library suggestion sub-components are logically independent, they have been separated to improve modifiability and reusability. Furthermore, the use of the configuration handler between the library selection UI and the configuration file abstracts away from the format of this file. This makes it easier to modify this subcomponent. For example, changing the format of the configuration file will not require a change in the library selection UI.

The configuration file is shared with the Fritzing Netlist analysis component of the system. Although this increases coupling between the components, it prevents duplication of information and therefore is a suitable solution in this case.

5.3.2 Implementation

Figure 5.7 demonstrates how the library suggestions are displayed to the user. For each part in the netlist, a scrollable checklist allows users to select the appropriate library (or libraries). The libraries are displayed based on the ranking provided by the Library Ranker component. The highlighted libraries are those that have previously been chosen for that part (as recorded by the configuration file).

5.3 Arduino Library Analysis

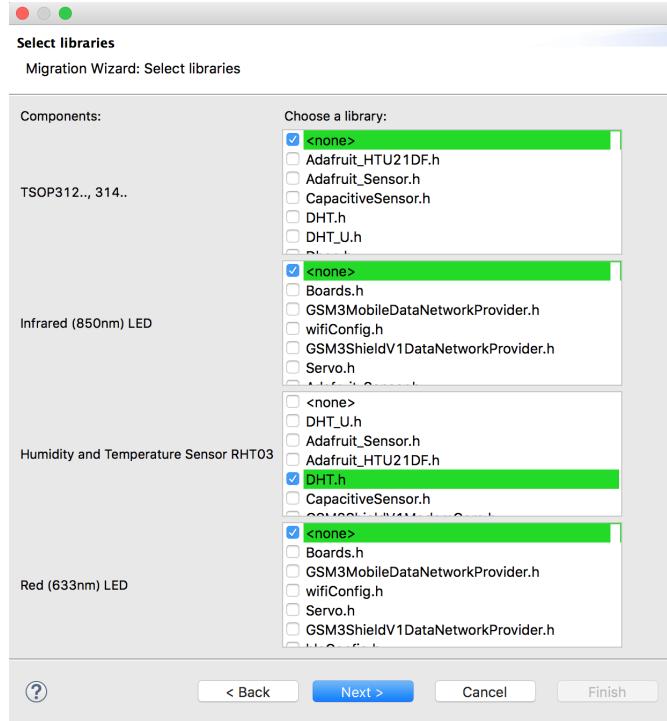


Figure 5.7: How the library suggestions are displayed to the user.

5.3.2.1 Library Ranker

Given a part name and the Arduino library header files (the root directory containing the Arduino libraries is input to the Eclipse plugin by the user), the Library Ranker component calculates a ranking for the suitability of each library. The ranking is treated as an information retrieval task based on the part name. In other words, the Arduino libraries' header files are ranked based on similarity to the part's name. This makes the assumption that the most suitable library for a part will make some reference or include similar terms to the part's name. Although this is not intended to be an exact recommendation, it provides a useful heuristic for library suggestion.

For each library, the term frequency-inverse document frequency (TFIDF)[28] value is calculated based on the part name.

TFIDF is defined as:

$$w_{t,d} = (1 + \log(tf_{f,d})) \times \log\left(\frac{N}{df_t}\right) \quad (5.1)$$

where:

df_t is the number of documents term t occurs in

N is the number of documents in the collection

$tf_{t,d}$ is the number of occurrences of term t in document d

5 Design and Implementation

In this case a document corresponds to a header file and a term corresponds to a word in the part name. The total ranking for a library is calculated by summing the TFIDF score for each term in the part name. TFIDF was chosen as the ranking metric as it takes into account both the frequency of occurrence of a term in the library as well as its “informativeness”. That is, a term is more informative if it appears in fewer documents of the collection (“informativeness” effectively corresponds to the $\log\left(\frac{N}{df_t}\right)$ term in the TFIDF equation). For example, if we consider the part *humidity and temperature sensor* the term *sensor* is less informative than *humidity* as many parts have the term *sensor* in the title and so will many header files. If we don’t take the “informativeness” into account, any part that contains *sensor* in the title is likely to rank highest the header file with the most occurrences of *sensor* in the text, regardless of the type of sensor. Since *humidity* is a less common term, using the TFIDF ranking will instead rank header files containing the less frequently occurring term *humidity* higher.

5.3.2.2 Configuration

The configuration file stores the name of libraries chosen for each part as well as the number of times it has been chosen. For example, Figure 5.5 shows that the library "DHT.h" has been chosen twice for the humidity and temperature sensor. Over time, this should improve the accuracy of library suggestions under the assumption that the correct library is chosen most frequently.

5.4 Parallax Project Analysis

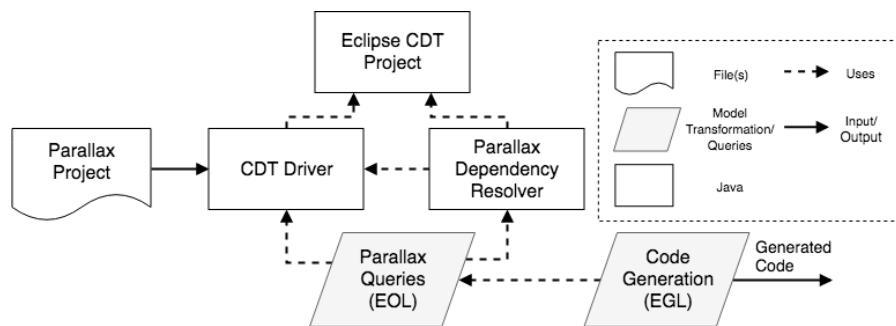


Figure 5.8: A more detailed view of the Parallax project analysis component of the system.

5.4.1 Design

For a similar reason as in the Fritzing netlist analysis component, the information extracted in the Parallax project analysis stage is coordinated by an EOL file that is used by the Code Generation EGL transform.

5.4 Parallax Project Analysis

The logic contained in the Parallax Dependency Resolver Java code could have been implemented in the Parallax Queries EOL file. However, it would have been much more difficult to write. As such, a Java class encapsulating the required logic was created. This class is added to a plugin (setup as in [29]) so that it can be accessed from EOL as a Native type.

The Eclipse CDT Project provides a fully functional C and C++ Integrated Development Environment based on the Eclipse platform [30]. The CDT Driver extends the Eclipse CDT Project, adding C/C++ support to the Epsilon framework (e.g. by providing support for C/C++ models) [31]. These projects are used to support the analysis of the Parallax project.

Initially, the intention for this component was to follow a more similar approach to those discussed in Section 2.1.2. However, this would require the implementation of a fully functional C/C++ metamodel which would be time-consuming and error prone. Therefore, the decision was made to query the AST (extracted using the CDT tools) directly.

5.4.2 Implementation



Figure 5.9: An example of a partially expanded AST for Parallax source code.

CDT contains two parsers, for C and C++, which are able to generate an abstract syntax tree (AST) from source code [32]. Figure 5.9 shows a partially expanded AST (more information on the structure of the AST can be found at [33]). The AST is generated for the input Parallax project and this is analysed to determine which function calls are part of Parallax libraries. These are then annotated in the generated code so that they can easily be replaced with equivalent functionality from the Arduino libraries. For example, functions from Parallax libraries are replaced with empty methods in the output file of the system. TODO directives within these empty methods make it clear they should be populated by engineers. Figure 5.10 shows an example of a generated method.

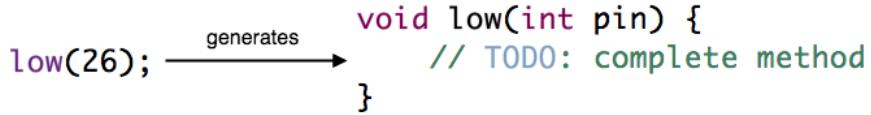


Figure 5.10: An example of the empty method generated from a Parallax library function.

The methods defined in Parallax libraries are identified as follows. The `ParallaxDependencyResolver` class extends the `ASTVisitor` class¹. This class is used to visit each `IASTNode` in the AST (each element of the AST in Figure 5.9 is an `IASTNode`) by calling `accept` on the root. When a node is visited, if it is not an instance of an `IASTFunctionCallExpression`, the node's children are visited. Otherwise, the following procedure is used to determine whether this function was defined in one of the Parallax libraries (and therefore needs replacing in the generated output):

1. The index is created for the project. This contains all the binding information for the project. Since creating the index is a time-consuming process requiring parsing all the project code, resolving the bindings and writing them to the index, the index is only created once.
2. The `resolveBinding` method is called on the `IASTName` of the node. This returns an `IBinding` instance.
3. The `IBinding` is used to look up the `IIndexName` for any declaration/definition of the node in the index.
4. The origin file can be found for this `IIndexName`. The file location is checked to see if it is contained within the directories containing the Parallax libraries. If so, it can be concluded that this node is part of a Parallax library and must be replaced in the output code.
5. If the node is determined to be part of the Parallax libraries, the declaration/definition of the node is collected.

Once all the nodes have been processed, there will be a list of all the declarations/definitions of methods that must be replaced in the output code. These are passed to the Code Generation EGL transform to create empty method stubs. Any other code is passed as is to the Code Generation EGL transform.

5.5 Supporting Migration Between Other Platforms

This section proposes how the approach developed for migrating between a Parallax Propeller Activity board and an Arduino Uno may be modified to support migration

¹This class implements the visitor design pattern and is the recommended method for traversing the AST based on [32].

between other platforms.

Firstly, to migrate between other platforms using C/C++ the following changes would need to be made:

- The configuration file would need to be updated with entries for the additional platforms so that the correct type information is available to the Fritzing netlist analysis component.
- The Fritzing netlist analysis component may need to be updated depending on how the pins are labelled in the netlist of the new target platform. Currently, the component implements logic to ignore, for example, the pins labelled GND / 5V (as pin variables are only generated for digital/analog pins, not power/ground) but other boards may label these pins differently in Fritzing.
- The code generation EGL transformation will need to be adapted as currently `setup()` and `loop()` methods are generated to be compatible with the Arduino Uno rather than the more usual `main()` method.

The Arduino library suggestion component should not require any changes provided the target platform uses C/C++ and the configuration file contains entries for all the parts in the Fritzing netlist. Additionally, the Parallax source code analysis component should be applicable to source code written in C and can be adapted to work with C++ code by implementing mappings from the source to generated code for the additional constructs (e.g. classes).

The following further changes would be required if the platforms use a language other than C/C++:

- The Parallax source analysis component would require the most modification as the use of the Eclipse CDT project parser would need to be replaced with a parser for the new source language. The `ParallaxDependencyResolver` class and `Parallax` queries EOL script would then require major modifications to process the output of the new parser.
- The Arduino library suggestion component would need to be modified to process the format of the libraries for the target platform (currently, the component searches for `.h` files within a provided root directory for the libraries).
- The code generation EGL transformation script would need to be updated to generate code in the format of the new target language.

5.6 Current Issues

This project is intended to be a proof-of-concept rather than a complete implementation. As such, there are a few issues in the approach which are discussed below.

One of the main issues of the system is the LibraryRanker for suggesting suitable Arduino libraries. Firstly, there's no guarantee that the Fritzing part names will actually

5 Design and Implementation

be useful in searching for suitable libraries. For example, in Figure 5.7 the name of the first part is shown as "TSOP312.., 314.." (this is actually an infrared receiver) - this doesn't give any indication of what the part is for and is unlikely to appear in any libraries as method names or in comments. Therefore, the suggestions for this part will be poor. However, it does provide a useful heuristic and could easily be replaced with a more accurate library suggestion system given more time. Furthermore, the issue is partially addressed by the use of the configuration file to store/indicate earlier choices, assuming that the correct library has been chosen in the past.

Another issue is the assumption that the netlist input to the system is valid. For example, that the provided netlist corresponds to a valid circuit and that it matches the configuration of the board that the generated code is intended to run on. As there is no easy way to check the correctness of this input, this is again outside the scope of the project.

A further issue with the Fritzing netlist analysis component is that there is no mapping between pin variables declared in the source code and the variables generated for the target platform. For example, suppose the source code contains a variable `int humidityPin = 12` which is used in various method calls for accessing a humidity sensor. The Fritzing netlist analysis component generates a variable `int humidityPinGenerated = 6` which needs to replace `humidityPin`. There is currently no mechanism for automating this mapping and so pin variables will have to be replaced manually.

Additionally, due to time limitations the configuration file used for the Fritzing netlist analysis component does not contain information for all the components available in Fritzing. However, it would be fairly easy (albeit time-consuming) to update this with a more comprehensive file.

Finally, the system doesn't handle all Parallax library dependent constructs - only methods are migrated. For example, `typedefs` are not migrated. Based on the timescale of the project, the decision was made to focus on migrating function calls as these are the most frequently used platform-dependent construct. However, other source code elements could be migrated in a manner similar to functions.

6 Evaluation

This chapter aims to evaluate the software migration system against the requirements set out in Chapter 4. Section 6.1 outlines how this will be achieved followed by Sections 6.2-6.4 describing the actual testing performed.

Tables 6.1 and 6.2 summarise the results of the evaluation against the corresponding requirements. For each requirement from Chapter 4 they indicate whether this requirement was met and include in the comments column which test case(s) or other analysis prove this is the case.

Finally, Sections 6.5 and 6.6 summarise the testing and discuss the limitations of the project respectively.

6.1 Test Plan

The software migration system consists of three main components: Fritzing Netlist Analysis, Arduino Libraries Analysis and Parallax Source Analysis. Each of these components works independently of the others to generate outputs used by the Code Generation EGL transform. As such, each component can be tested separately to ensure the correct outputs are produced. Section 6.2 outlines the test cases used for this part of the evaluation.

The full system must then be evaluated against the requirements in Chapter 4. Case studies in Section 6.3 are used to evaluate whether all requirements except S-2, NF-1 and NF-2 are met.

The requirements S-2, NF-1 and NF-2 concern the modifiability of the system. These requirements and whether they have been met are discussed in Section 6.4.

6.2 Test Cases

6.2.1 Fritzing Netlist Analysis Test Cases

The following test cases check that the Fritzing Netlist Analysis component generates pin variables as expected. Each test case contains a description of the goal of that particular test, the input netlist, expected variables and the actual output. Finally, Section 6.2.1.4 contains a discussion of the results of these tests.

6.2.1.1 Test Case 1

Description: Check that a component with multiple pins is handled correctly.

6 Evaluation

Input: Netlist shown in Section A.1 for the diagram in Figure 6.1.

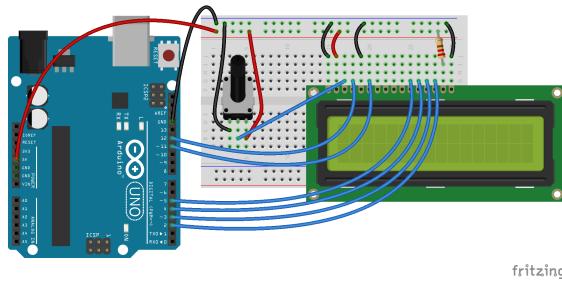


Figure 6.1: Fritzing diagram for Fritzing netlist analysis test case 1.

Expected Output: lcd → 2, 3, 4, 5, 11, 12

Actual Output: See Figure 6.2.

```
const int LC1WritePin1 = 3;
const int LC1WritePin2 = 4;
const int LC1WritePin3 = 5;
const int LC1WritePin4 = 11;
const int LC1WritePin5 = 12;
const int LC1WritePin6 = 2;
```

Figure 6.2: Output of Fritzing netlist analysis test case 1.

Result: Test case passed

6.2.1.2 Test Case 2

Description: Check that multiple components connected to one pin are handled correctly.

Input: Netlist shown in Section A.2 for the diagram in Figure 6.3.

Expected Output: led1 → 13; led2 → 13

Actual Output: See Figure 6.4.

Result: Test case passed

6.2.1.3 Test Case 3

Description: Check that a more complex netlist with many components is handled correctly. Additionally check that intermediate components between input/output parts (e.g. resistors) are ignored.

6.2 Test Cases

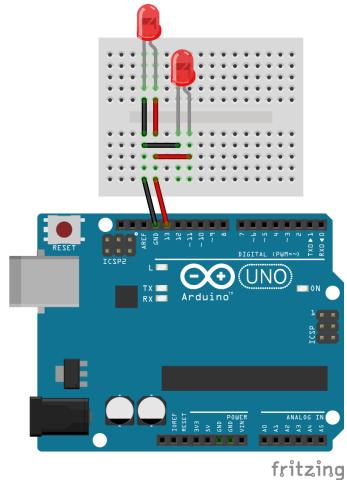


Figure 6.3: Fritzing diagram for Fritzing netlist analysis test case 2.

```
const int red633nmLED2WritePin1 = 13;  
const int red633nmLED1WritePin1 = 13;
```

Figure 6.4: Output of Fritzing netlist analysis test case 2.

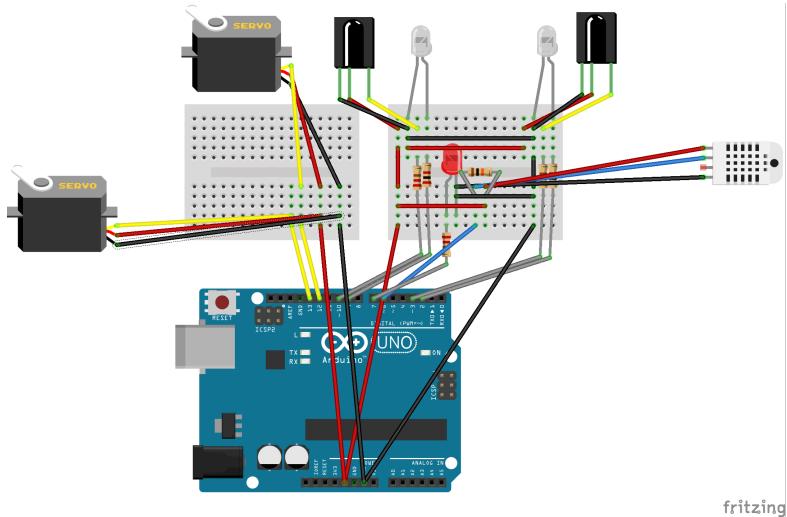


Figure 6.5: Fritzing diagram for Fritzing netlist analysis test case 3.

Input: Netlist shown in Section A.3 for the diagram in Figure 6.5.

Expected Output: • IR receivers → 10, 3

- IR LEDs → 9, 2
- Servos → 13, 12

6 Evaluation

- Red LED → 7
- Humidity and temp. sensor → 6

Actual Output: See Figure 6.6.

```
const int basicServo1WritePin1 = 12;
const int humidityandTemperatureSensorRHT031ReadPin1 = 6;
const int tS2ReadPin1 = 3;
const int infrared850nmLED1WritePin1 = 2;
const int tS1ReadPin1 = 10;
const int infrared850nmLED2WritePin1 = 9;
const int basicServo2WritePin1 = 13;
const int red633nmLED1WritePin1 = 7;
```

Figure 6.6: Generated pin variables.

Result: Test case passed

6.2.1.4 Discussion

These test cases show the Fritzing netlist analysis component performs as expected. In addition, the generated variables have intelligible names for most cases. However, the variables `tS2ReadPin1`/`tS1ReadPin1` are not particularly useful. This is because the name of the IR receivers in Fritzing is “TSOP312.., 314..” and so this can’t be used to generate a useful variable name. Additionally, the “d” has been cut off in the case of the LCD variables, for example, `LC1WritePin1`. This indicates more work needs to be done on the logic to generate the variable names.

6.2.2 Arduino Library Suggestion Test Cases

The following test cases evaluate the quality of the output from the Arduino libraries suggestion component. Three circuit parts that require the use of Arduino libraries have been chosen and this component has been used to generate a list of candidate libraries. For each component, the suggestions are inspected to determine the rank of the expected library. Section 6.2.2.4 discusses the results of these test cases.

6.2.2.1 Test Case 1

Input: LCD Screen

Expected Library: `LiquidCrystal.h`

Actual Output: See Figure 6.7.

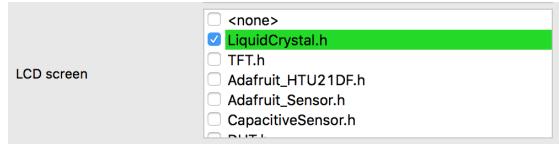


Figure 6.7: Output of Arduino library suggestion test case 1.

6.2.2.2 Test Case 2

Input: Humidity and temperature sensor (DHT22)

Expected Library: DHT.h

Actual Output: See Figure 6.8.

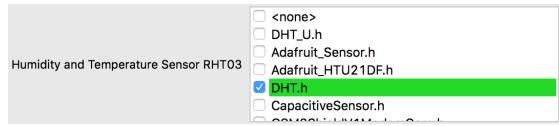


Figure 6.8: Output of Arduino library suggestion test case 2.

6.2.2.3 Test Case 3

Input: Servo

Expected Library: Servo.h

Actual Output: See Figure 6.9.

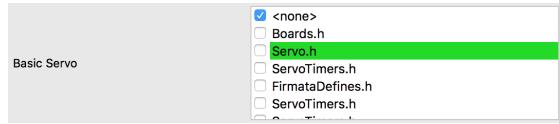


Figure 6.9: Output of Arduino library suggestion test case 3.

6.2.2.4 Discussion

For all three test cases the correct libraries were shown in the top 5 (out of 71) suggestions showing that, in this case, LibraryRanker provides a useful heuristic for narrowing down the libraries for the target platform code. The ranking algorithm proved particularly effective for the LCD display component with the appropriate `LiquidCrystal.h` library being ranked top. Although the correct library was only ranked fourth in the case of the humidity and temperature sensor, the more highly suggested libraries are all for humidity/temp. sensors just not the specific DHT22 sensor used in this circuit.

6 Evaluation

A potential issue with the ranking algorithm is highlighted by the servo test case. The Boards.h library is ranked higher than Servo.h as it contains macro definitions for variables such as MAX_SERVOS for many different types of board (e.g. Arduino Mega, Primo etc.) and therefore the term “servo” appears many times, contributing to a higher TF-IDF rank. A potential improvement to the ranking heuristic would also take file size into account (e.g. consider the proportion of the file containing the term “servo” as opposed to the raw count). This would handle cases like this better as although the raw count of the term “servo” is high, the file is so large that it only makes up a small proportion of the file. Whereas, in Servo.h, although the raw count is not as high, it makes up a larger proportion of all the terms in the file.

6.2.3 Parallax Source Code Analysis Test Cases

The expected behaviour for code generation is:

- Parallax library include statements *are not* present in the output
- Any other library include statements *are* present in the generated code
- Empty methods with TODO directives are generated for any Parallax library methods used in the input code
- Code from the main method outside of the `while(1)` loop in the input is copied to a `setup()` method in the output
- Code from the `while(1)` loop in the input is copied to the `loop()` method of the output
- Any other methods defined in the input are copied to the output
- Global variables in the input are copied to the output

The test case in Section 6.2.3.1 checks that the Parallax source code analysis component generates code as expected.

6.2.3.1 Test Case 1

Description: The input source code contains a mixture of Parallax library methods and other functions as well as global variables, user-defined methods and include statements for non-Parallax libraries. This test case checks all the criteria for a successful migration (as discussed in Section 6.2.3) are met.

Input: The source code in Appendix B.1.

Expected Output: 1. Methods generated with the following signatures:

- `void drive_setRampStep(int stepsize)`
- `void drive_rampStep(int left, int right)`

- int input(int pin)
 - void high(int pin)
 - void low(int pin)
 - void freqout(int pin, int msTime, int frequency)
 - char dht22_read(int dht_pin)
 - int dht22_getTemp(char temp_units)
 - int dht22_getHumidity()
2. Only stdio.h include in output.
 3. The global variables int irLeft, irRight; int humPin = 6; int ledPin = 8; and float humThresh = 25.0; are present in the output.
 4. The statements low(26); low(27); and drive_setRampStep(12); are included in the setup() method of the output.
 5. All other statements in the main method of the input are in the loop() method of the output.
 6. The myPrintf method from the input is copied to the output.

Actual Output: See Appendix B.2.

Result: Test case passed.

6.3 Case Studies

The following case studies are used to evaluate whether the requirements set out in Chapter 4 have been met. The first case study in Section 6.3.1 demonstrates the operation of the system when provided with correct inputs and is used to evaluate requirements S-1, S-4, S-5, F-1, F-2, F-5, NF-3 and NF-4. The second and third case studies (in sections 6.3.2 and 6.3.3) are used to evaluate the system when there are errors in the migration process. These case studies are used to evaluate requirements F-3 and S-3/F-4 respectively. A summary of the results of these case studies along with the corresponding requirements are contained in tables 6.1 and 6.2.

6.3.1 Case Study 1 (CS-1)

Figure 6.5 shows the Fritzing diagram for the target platform in CS-1 and figures 6.10 and 6.11 show the set the setup of the source and target boards respectively. The source code, Fritzing netlist and generated code for this case study can be viewed in Appendices B.1, A.3 and C.1 respectively. For this case study, the correct functioning of the Fritzing netlist analysis, Arduino libraries suggestion and Parallax source code analysis components have been evaluated in the earlier sections 6.2.1.3, 6.2.2.2/6.2.2.3

6 Evaluation

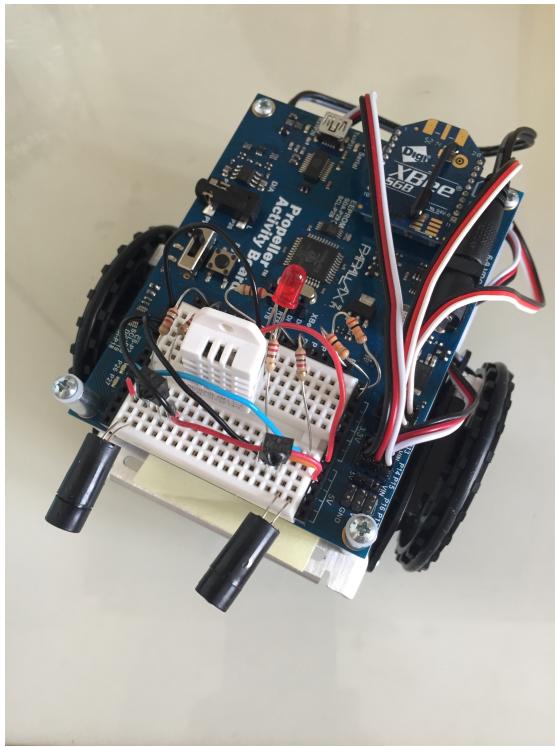


Figure 6.10: Parallax Propeller Activity board setup for CS-1.

and 6.2.3.1 respectively. The migrated code (i.e. the output code from the system with manual additions) is shown in Appendix C.2.

6.3.1.1 Description

The source code and circuit uses infrared receivers/LEDs to detect and move around its surroundings using servo motors. Periodically, a humidity reading is taken from the humidity/temperature sensor. If this is above some threshold, an LED lights up. When connected to a computer, the humidity and temperature readings can be displayed to the console. Since Parallax only supports a subset of C, in order to write to the console a function myPrintf is defined in the source code. This performs the same function as the standard libraries' printf.

This case study is used to demonstrate the migration of code for components that require different complexities of mapping between the source and target platforms. Firstly, it is fairly simple to map between the libraries for the humidity and temperature sensor.

In the Parallax source, this sensor can be used as:

```
#include "dht22.h"
```

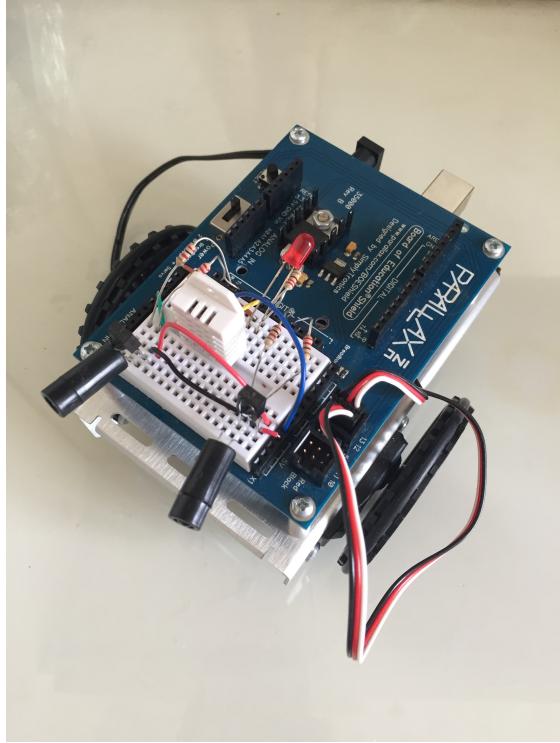


Figure 6.11: Arduino Uno setup for CS-1.

```
dht22_read(sensorPin);           // Get most recent readings
dht22_getHumidity();            // Get humidity reading
dht22_getTemp(CELSIUS);         // Get temperature reading
```

To use the same functionality in the Arduino code requires the following:

```
#include "DHT.h"

DHT humiditySensor(DHTPIN, DHTTYPE); // Setup
humiditySensor.readHumidity();        // Get humidity reading
humiditySensor.readTemperature();     // Get temperature reading
```

Aside from some small differences in how the sensors are initialised, there is an almost exact mapping between the methods for reading humidity/temperature values.

Mapping between methods to use the servos is more difficult. For example, to control the servos in Parallax, you can provide the speed to each servo in ticks per second (where a tick is $\frac{1}{64}$ of a wheel rotation). The Parallax libraries also include methods to gradually accelerate to a specified speed (e.g. `drive_RampStep`). On the other hand, the Arduino libraries control the servos using pulse width (see [34] for more information on how this is used to control servos). See below for an example of how servos are controlled in Parallax compared to Arduino.

6 Evaluation

```
/* Parallax */
#include "abdrive.h"

// Perform two servo rotations per second
// Note we do not need to set up pins for the servos in Parallax
drive_rampStep(128 /*left*/, 128 /*right*/);

/* Arduino */

Servo servoLeft, servoRight;

// Setup servo pins
servoLeft.attach(13);
servoRight.attach(12);

servoLeft.writeMicroseconds(1600);
servoRight.writeMicroseconds(1600);
delay(20);
```

6.3.1.2 Analysis

Firstly, the system was successful in demonstrating that, provided with correct inputs, it is able to generate code (i.e. Appendix C.1) that with manual additions (i.e. Appendix C.2) can be run on the target platform, consequently satisfying requirement S-1.

Two types of compile-time errors were present in the generated output (see Appendix C.1) from the migration system. Firstly, a macro definition from the Parallax dht22 library (CELSIUS) caused an error as it was not defined in the generated output (see Figure 6.12). Other than this compile time error, there was no indication that this variable needed to be replaced in the generated code which is in conflict with requirements S-4/F-5. However, other than this small issue, the methods that needed to be completed were clearly marked.

```
float temp = dht22_getTemp(CELSIUS) / 10.0;
```

Figure 6.12: Compile error due to header macro definition.

Figure 6.13 shows the AST of the source code for this declaration. Although CELSIUS is used in the code, this is resolved to the literal expression having a value of 0 in the AST. As such, there is no way to tell from the AST that this value comes from a definition in a library file. One solution would be to copy all the macro definitions from the includes in the Parallax code. However, this could become very messy and increase the file size a lot particularly when there are many included files. A better solution would require creating a parser that would indicate these macros within the AST but this is outside the scope of this project.

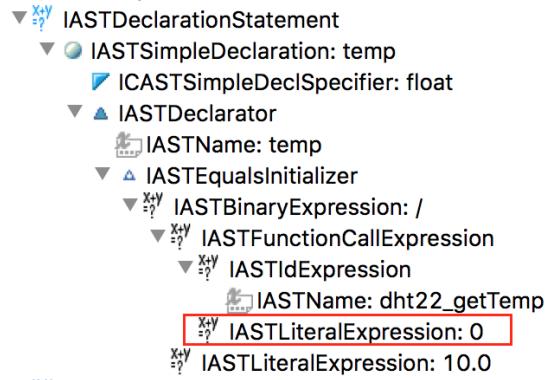


Figure 6.13: AST for the declaration using a macro definition. The value is resolved to a literal expression (highlighted in red).

The second type of compile-time error is due to no return types as can be seen in Figure 6.14. It would be fairly easy to add default return values. However, this compile-time error can work as an extra indication that this method needs completing manually and so it is useful to leave this procedure as it is.

```

char dht22_read(int dht_pin) {
    // TODO: complete method
}
    
```

Figure 6.14: Compile error due to no return.

As these errors were resolved with the manual additions required for migration, it was demonstrated that requirement F-2 was satisfied and the code in Appendix C.2 was able to be deployed on the Arduino without errors.

The size of the Parallax source code was 1,506 bytes compared to 2,156 bytes for the generated code and 2,922 bytes for the manually completed code. As such, the migrated code is roughly twice the size of the input. However, some unused functions have been left in the migrated code for demonstration purposes so the migrated code could be made smaller. It should be expected that there will be a constant factor increase in the size of the migrated code since empty methods are generated for each of the Parallax library functions in the source code. In the worst case, each line of the input will make use of a Parallax library function, requiring three generated lines of code in the output (method declaration; TODO statement; closing bracket). There may also be additional lines added for generated pin variables and target platform include statements. However, the size of these additions is likely to be small compared to the size of generated methods. Therefore, an upper bound on the generated code (without manual additions) is $3L$ where L is the number of lines in the input. For most cases this would be an acceptable increase in size and so NF-3 is satisfied. However, of course

6 Evaluation

this depends on the requirements for the particular application. Furthermore, the size of the output depends on the mappings between library functions for the source and target platforms. For example, I found that to perform the same functionality using Arduino libraries tends to require more lines of code since setup is generally made more explicit than when using Parallax libraries (e.g. consider the servo code in Section 6.3.1.1). As such, it may be expected that programs rewritten directly for the target platform may be larger anyway without the use of this migration system.

Based on an inspection of the application run on both the source and target platforms, there was no perceptible difference in performance and so requirements S-5/NF-4 are satisfied. The only potential reduction in performance (other than platform differences) could come from the use of additional method calls from the generated methods. However, compilers optimized for speed can inline functions in this case so that this does not occur. Otherwise, the difference is likely to be negligible unless there is a cache miss [35]. Other than this, differences in performance are likely to be much more dependent on the efficiency of the corresponding platform libraries and the hardware specifications for the microcontrollers themselves.

Between the source and target applications there are some minor differences in behaviour because of the differences in libraries. For example, the servos do not accelerate when running the code on the Arduino as they do on the Parallax board since there is no equivalent function in the Arduino libraries (see Section 6.3.1.1). Other than this, from manual inspection, the migrated code performs the same functionality as the source code (i.e. using infrared LEDs/receivers to maneuver and periodically take humidity/temp. readings etc.) and so requirement F-1 is met.

Overall, with some minor exceptions, this case study has demonstrated that requirements S-1, S-4, S-5, F-1, F-2, F-5, NF-3 and NF-4 have been met.

6.3.2 Case Study 2 (CS-2)

This case study is used to evaluate whether the requirement F-3 is met (i.e. the system displays appropriate errors when the source code cannot be migrated). The input to the system is shown in Appendix C.3. This is a subset of the input code from CS-1 but without the required include statements and so the source code displays many compile-time errors. Based on F-3, the system should indicate that the migration of this file is not possible. However, no errors are thrown and the code in Appendix C.4 is produced. Therefore, requirement F-3 has not been met. Handling invalid inputs should be part of future work. This could be addressed by, for example, compiling the input before the migration process is run and checking for any errors.

6.3.3 Case Study 3 (CS-3)

This case study is used to evaluate whether the requirements S-3/F-4 are met (i.e. the system clearly indicates any errors that have occurred during its use). The inputs to the system are the same as CS-1 (i.e. the source code is as contained in Appendix B.1 and

6.4 Evaluation of Modifiability

the netlist is the same as Appendix A.3). However, an empty configuration file is used meaning any attempts to look up part types or previously used libraries will fail.

When the system is run, the errors shown in Figures 6.15 and 6.16 are thrown.

```
java.lang.NullPointerException  
at org.sew569.softwaremigrator.MigrationWizardPageTwo.createControl(MigrationWizardPageTwo.java:178)
```

Figure 6.15: First error thrown when performing migration using empty configuration file.

```
[Fatal Error] empty_config.xml:1:1: Premature end of file.  
Internal error: org.xml.sax.SAXParseException; systemId: file:/Users/sophie/eclipse-projects/thesis/transform_v2/empty_config.xml; lineNumber: 1; columnNumber: 1;
```

Figure 6.16: Second error thrown when performing migration using empty configuration file.

With knowledge of the project, this has occurred because the second page of the Eclipse wizard (i.e. MigrationWizardPageTwo) uses the configuration file to find which parts of the input netlist need a library suggestion UI generating. However, without this understanding, it is not at all clear that an empty configuration file has caused this error. Therefore, requirements S-3/F-4 have not been met. Handling this should be part of future work. Before the migration process is run, the configuration file should be checked to ensure it contains an entry for all the required parts, otherwise an exception should be thrown (with a useful error message).

6.4 Evaluation of Modifiability

Modifiability was one of the main factors taken into account when designing the system as discussed in Chapter 5. Some of the decisions made to support modifiability include:

1. Use of the popular and well-documented Epsilon MDE framework.
2. Separation between the three main components of the system (Fritzing netlist analysis, Arduino library suggestions and Parallax source code analysis).
3. Use of Java plugins for more complex logic rather than just using Epsilon languages. For example, the code for resolving which methods come from source-platform libraries (ParallaxDependencyResolver) or the library ranking code (LibraryRanker).
4. Using additional .eol files to simplify the coordinating model transform.

These decisions support requirements S-2/NF-1 and items 2/4 support requirement NF-2.

6.5 Test Summary

The tests in Section 6.2 demonstrate that the three main components of the system (Fritzing netlist analysis, Arduino library suggestions and Parallax source code analysis) work as expected.

Tables 6.1 and 6.2 summarise, based on the results from the case studies in Section 6.3, whether each of the requirements set out in Chapter 4 was met. Note that in table 6.2 the headings “Priority” and “Satisfied” have been shortened to S and P respectively to more easily display the table. A ✓ symbol indicates the requirement was met and a ✗ indicates it wasn’t. A * next to a checkmark indicates the requirement has been satisfied with some minor exception which will be mentioned in the corresponding “Comments” column of the table. These tables show that the majority of requirements have been met. In particular, all of the high priority (M/Must) requirements have been met aside from some minor exceptions. The remaining issues with the system are summarised in Section 6.5.1 below.

6.5.1 Current Issues

- The Arduino library suggestions component could be improved, particularly in the case of large header files as discussed in Section 6.2.2.4.
- Macro definitions from source-platform libraries have no clear indication that they need replacing in the generated code (discussed in Section 6.3.1.2).
- The system does not display appropriate errors when the source code can’t be migrated (see Section 6.3.2).
- The system does not clearly indicate errors that have occurred during its use (see Section 6.3.3).

6.6 Validity

This section discusses the validity of the evaluation in order to determine the reliability of the conclusions drawn from the testing/case studies performed. The analysis will consider the internal, external and construct validity in sections 6.6.1 to 6.6.3.

6.6.1 Internal Validity

Internal validity threats occur when the procedure used to obtain results introduces bias. The test cases were designed to limit bias by examining a range of scenarios of different complexity. For example, the Fritzing netlist analysis tests dealt with parts having different numbers of read/write pins and a more complex circuit with multiple sensors. A possible limitation to the internal validity is that only one test case was used to analyse the Parallax source code analysis component. However, this test case was

fairly complex and covered all of the cases for expected behaviour of this component discussed in Section 6.2.3. Therefore, it was deemed unnecessary to perform further test cases. Similarly, there was only one case study where the full migration procedure was performed. However, this was again a fairly complex case and was therefore considered sufficient, particularly in combination with the testing performed of the individual components in Section 6.2.

6.6.2 External Validity

External validity threats limit the generalisation of results, in this case, the application of this approach to different microprocessors. Firstly, the Parallax source code analysis component should be applicable to any source code written in C and can be adapted to work with C++ code by implementing mappings from the source to generated code for the additional constructs (e.g. classes). Similarly, the Fritzing netlist analysis component should work with other boards provided that Fritzing contains a component for this additional board and the configuration file is updated with this part. The Arduino libraries suggestion component should also be generalisable to other target platforms given an updated configuration file. However, some minor modifications may be required for the coordinating code generation EGL transform. For example, the generated code currently conforms to the Arduino code structure of `setup()` and `loop()` methods whereas most C code uses a `main()` method instead.

To generalize the approach to platforms using languages other than C/C++ would require more extensive modification. This concern has been addressed by prioritising an easily modifiable architecture in the design stage (as discussed in Chapter 5) and making use of the well-documented Epsilon MDE framework.

6.6.3 Construct Validity

Construct validity threats can occur from assumptions and simplifications in the case studies. A potential threat is that the case studies did not migrate real-world applications. However, the case studies were designed to migrate functionality that is likely to appear in real-world applications. For example, CS-1 demonstrates environment sensing, movement and humidity/temperature sensing.

Table 6.1: Case Study System Test Results

ID	Priority	Description	Satisfied	Comments
S-1	M	Users can provide the system with the source code for a Parallax Propeller Activity Board and target code for the Arduino Uno will be generated.	✓	See Section 6.3.1.2.
S-2	M	Users can easily adapt the system to support migration of software between different microprocessors.	✓	See Section 6.4.
S-3	S	Users can easily determine sources of errors when using the system.	✗	See Section 6.3.3.
S-4	S	Users can easily identify areas of the target code that cannot automatically be migrated and must be manually completed.	✓*	See Section 6.3.1.2. *Except macro definitions.
S-5	S	The performance of the migrated code is not significantly worse than the source application.	✓	See Section 6.3.1.2.

Table 6.2: Case Study Functional/Non-Functional Test Results

ID	P	Description	Acceptance Criteria	S	Comments
F-1	M	The target microcontroller running the migrated code (potentially with manual additions) displays the same behaviour as the source microcontroller, provided the source code is valid.	Manual inspection of the migrated application confirms the behaviour is the same as the source application.	✓	See Section 6.3.1.2.
F-2	M	The migrated code deploys without errors on the target microcontroller if the provided source code is valid.	Manual inspection of the migration process confirms a valid source application can be migrated without errors.	✓*	See Section 6.3.1.2. *Once manual additions have been made.
F-3	S	The system displays appropriate errors when the source code cannot be migrated (e.g. there are errors in the source code or hardware incompatibilities).	Errors are thrown when invalid source code is provided to the migration process.	✗	See Section 6.3.2.
F-4	S	The system clearly indicates any errors that have occurred during its use (e.g. the migration process fails).	Errors are thrown if the migration process fails. The errors clearly indicate the source of the problem.	✗	See Section 6.3.3.

F-5	S	The system should indicate any parts of the code that must be manually completed.	Manual inspection of the migrated code confirms that all parts of the code that must be manually completed are clearly indicated (e.g. by TODO directives).	✓*	See Section 6.3.1.2. *Except macro definitions from source platforms libraries.
NF-1	M	The system should be easy to adapt to support migration between different microcontrollers.	Manual inspection of the system should indicate that it is easy to adapt. For example, if the system is based on an established MDE framework it will be easier to adapt as there will be more documentation available.	✓	See Section 6.4.
NF-2	S	The system should display a high-level of encapsulation.	Manual inspection of the system should indicate measures have been taken to facilitate encapsulation. For example, separation of hardware-dependent parts of the code.	✓	See Section 6.4.
NF-3	S	The migrated code should not be significantly larger than the source code.	Migrated code is of a size acceptable to the stakeholder.	✓	See Section 6.3.1.2.
NF-4	S	The migrated code should exhibit similar performance to the source code.	Migrated code runs in a duration acceptable for the stakeholder.	✓	See Section 6.3.1.2.

7 Conclusion

Obsolescence is increasingly challenging for many technical projects, particularly in safety-driven domains with long project life-cycles where software/hardware often become non-procurable before the project is complete. Although hardware and software obsolescence have a similar impact on issues and costs in these projects, there has been much more research and success in dealing with the hardware obsolescence problem.

This project aimed to build on existing research in dealing with the software obsolescence problem and in particular tackle the “migration of an entire software system from a legacy hardware platform to a modern more powerful platform”. The developed approach aids in the specific case of migrating code compatible with a Parallax Propeller Activity Board to an Arduino Uno in order to demonstrate a feasible approach to the more general problem of migrating between platforms. Based on the success of MDE practices to update legacy applications in industry, the approach was designed to use these techniques.

In the course of the project, three main components were developed. Firstly, a Fritzing netlist analysis component enabled a diagram of the circuit for the target platform to be supplied in order to calculate pin variables for the generated code. This component simplifies the migration process when the source and target platforms have functionally equivalent circuits using different pin configurations. Secondly, an Arduino library analysis component was produced in order to process the target platform libraries and suggest suitable replacements for the source platform libraries. This component helps simplify the manual procedure of mapping between libraries which is a particularly difficult task to automate. Finally, a Parallax source code analysis component enables the source platform dependent functions to be identified so that these can be replaced with empty method stubs in the generated code. This reduces the difficulty in mapping between source and target platform methods. From these components, a C file is generated that, with some manual modifications, is compatible with the target platform.

Based on the evaluation of the system performed in Chapter 6, the system was successful in supporting migration between the Parallax and Arduino platforms for a case study of reasonable complexity. Furthermore, the project has been designed in such a way that the individual components of the system can easily be modified/reused in supporting migration between different platforms as discussed in Chapter 5. The source code for the project can be found at:

<https://github.com/sofaw/thesis-code/tree/master/SoftwareMigrator>

7.1 Further Work

If work is continued on the project, the first areas to address are the issues found during the evaluation as summarised in Section 6.5.1. Once these issues have been tackled the following avenues could be investigated for improving the functionality of the system:

1. As mentioned in Section 5.4.2, only the methods from source platform libraries are migrated to the target platform. A top priority for future work would be making sure that all constructs (e.g. typedefs, macro definitions) from Parallax libraries are migrated. This can be achieved in a similar fashion to the method for migrating functions.
2. As well as using the part names for identifying the mappings between source and target libraries, the source platform library names may provide a good indication of the libraries required for the target platform. For example, the library suggestions for the humidity and temperature sensor in Section 6.2.2.3 were all related to humidity and temperature sensing but some of the more highly ranked libraries than the required `DHT.h` were not for the particular sensor in question e.g. the `Adafruit_HTU21DF.h` library. Since the Parallax `dht22.h` library is used in the source platform code, this would give a good indication that the `DHT.h` library is the most suitable in this case.
3. When performing the manual aspect of the migration for the case study in Section 6.3.1, I found it difficult to determine what functionality the generated empty methods should be replaced with. A simple modification to the existing system would copy across any comments associated with the method in the source platform library in order to give a better indication of the functionality of the method. Consequently, this would make it easier to determine what functionality each migrated method should perform.
4. Related to the previous suggestion, the source and target platform libraries could be analysed to determine suggested replacement methods (similarly to how replacement libraries are currently suggested) in order to further simplify the manual migration process.

Bibliography

- [1] B. Bartels, U. Ermel, P. Sandborn and M. G. Pecht, *Strategies to the prediction, mitigation and management of product obsolescence*. John Wiley & Sons, 2012, vol. 87.
- [2] F. J. R. Rojo, R. Roy and E. Shehab, 'Obsolescence management for long-life contracts: State of the art and future trends', *The international journal of advanced manufacturing technology*, vol. 49, no. 9-12, pp. 1235–1250, 2010.
- [3] P. Singh and P. Sandborn, 'Obsolescence driven design refresh planning for sustainment-dominated systems', *The engineering economist*, vol. 51, no. 2, pp. 115–139, 2006.
- [4] P. Sandborn, 'Software obsolescence-complicating the part and technology obsolescence management problem', *Ieee transactions on components and packaging technologies*, vol. 30, no. 4, pp. 886–888, 2007.
- [5] S. Rajagopal, J. Erkoyuncu and R. Roy, 'Software obsolescence in defence', *Procedia cirp*, vol. 22, pp. 76–80, 2014.
- [6] C. Adams, 'Getting a handle on cots obsolescence', *Avionics magazine*, vol. 1, 2005.
- [7] P. Sandborn and J. Myers, 'Designing engineering systems for sustainability', in *Handbook of performability engineering*, Springer, 2008, pp. 81–103.
- [8] K. Kowalczyk and A. Kwiecinska, *Model-driven software modernization*, 2009.
- [9] S. Gerasimou, D. Kolovos, R. Paige and M. Standish, 'Technical obsolescence management strategies for safety-related software for airborne systems', in *Federation of international conferences on software technologies: Applications and foundations*, Springer, 2017, pp. 385–393.
- [10] F. Fleurey, E. Breton, B. Baudry, A. Nicolas and J.-M. Jézéquel, 'Model-driven engineering for software migration in a large industrial context', in *International conference on model driven engineering languages and systems*, Springer, 2007, pp. 482–497.
- [11] H. Bruneliere, J. Cabot, G. Dupé and F. Madiot, 'Modisco: A model driven reverse engineering framework', *Information and software technology*, vol. 56, no. 8, pp. 1012–1032, 2014.
- [12] T. Herald, D. Verma, C. Lubert and R. Cloutier, 'An obsolescence management framework for system baseline evolution—perspectives through the system life cycle', *Systems engineering*, vol. 12, no. 1, pp. 1–20, 2009.
- [13] J. Wilbrink, 'Facilitating the migration from 8-bit to 32-bit microcontrollers', Atmel Corporation, Tech. Rep., 2004.

Bibliography

- [14] D. C. Schmidt, 'Model-driven engineering', *Computer-ieee computer society-*, vol. 39, no. 2, p. 25, 2006.
- [15] J. Bézivin, 'In search of a basic principle for model driven engineering', *Novatica journal, special issue*, vol. 5, no. 2, pp. 21–24, 2004.
- [16] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. H. Cheng, P. Collet, B. Combemale, R. B. France, R. Heldal, J. Hill *et al.*, 'The relevance of model-driven engineering thirty years from now', in *International conference on model driven engineering languages and systems*, Springer, 2014, pp. 183–200.
- [17] J. Bézivin and O. Gerbé, 'Towards a precise definition of the omg/mda framework', in *Automated software engineering, 2001.(ase 2001). proceedings. 16th annual international conference on*, IEEE, 2001, pp. 273–280.
- [18] K. Czarnecki and S. Helsen, 'Classification of model transformation approaches', in *Proceedings of the 2nd oopsla workshop on generative techniques in the context of the model driven architecture*, USA, vol. 45, 2003, pp. 1–17.
- [19] D. Kolovos, L. Rose, R. Paige and A. Garcia-Dominguez, 'The epsilon book', *Structure*, vol. 178, pp. 1–10, 2010.
- [20] [Online]. Available: <https://www.eclipse.org/epsilon/>.
- [21] [Online]. Available: <http://www.bbc.co.uk/schools/gcsebitesize/design/electronics/logicrev6.shtml>.
- [22] [Online]. Available: <https://www.parallax.com/sites/default/files/downloads/32910-Propeller-Activity-Board-Guide-v1.1.pdf>.
- [23] [Online]. Available: <https://0x0sec.org/t/the-hackers-lab-arduino/1708>.
- [24] [Online]. Available: <https://www.farnell.com/datasheets/1682209.pdf>.
- [25] W. W. Royce, 'Managing the development of large software systems: Concepts and techniques', in *Proceedings of the 9th international conference on software engineering*, IEEE Computer Society Press, 1987, pp. 328–338.
- [26] D. J. Fernandez and J. D. Fernandez, 'Agile project management—agilism versus traditional approaches', *Journal of computer information systems*, vol. 49, no. 2, pp. 10–17, 2008.
- [27] [Online]. Available: <https://www.agilebusiness.org/content/moscow-prioritisation-0>.
- [28] G. Salton and M. J. McGill, 'Introduction to modern information retrieval', 1986.
- [29] [Online]. Available: <https://www.eclipse.org/epsilon/doc/articles/call-java-from-epsilon/>.
- [30] [Online]. Available: <https://www.eclipse.org/cdt/>.
- [31] [Online]. Available: <https://github.com/gerasimou/EMC-CDT>.
- [32] [Online]. Available: https://wiki.eclipse.org/CDT/designs/Overview_of_Parsing.
- [33] [Online]. Available: <http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.cdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fcdt%2Fcore%2Fdom%2Fast%2Fpackage-summary.html>.

Bibliography

- [34] [Online]. Available: https://en.wikipedia.org/wiki/Servo_control.
- [35] [Online]. Available: <https://softwareengineering.stackexchange.com/questions/318055/how-much-do-function-calls-impact-performance>.

Appendices

A Fritzing Netlist Analysis Test Inputs

A Fritzing Netlist Analysis Test Inputs

A.1 Test Case 1 Netlist

```
<?xml version='1.0' encoding='UTF-8'?>
<!!-- Created with Fritzing (http://www.fritzing.org/) -->
<netlist sketch="LCD.fzz" date="Fri Aug 24 22:20:50 2018">
<net>
  <connector id="connector63" name="D2">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
  <connector id="connector13" name="DB7">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
<net>
  <connector id="connector64" name="D3_PWM">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
  <connector id="connector12" name="DB6">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
<net>
  <connector id="connector65" name="D4">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
  <connector id="connector11" name="DB5">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
<net>
  <connector id="connector66" name="D5_PWM">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
  <connector id="connector10" name="DB4">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
<net>
  <connector id="connector54" name="D11_PWM/MOSI">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
  <connector id="connector5" name="E">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
<net>
  <connector id="connector55" name="D12/MISO">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
  <connector id="connector3" name="RS">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
<net>
  <connector id="connector15" name="LED->">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
  <connector id="connector57" name="GND">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
  <connector id="connector88" name="GND">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
```

```

</net>
<net>
  <connector id="connector64" name="D3_PWM">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector63" name="D2">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector62" name="D1/TX">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector87" name="5V">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector61" name="D0/RX">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector86" name="3V3">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector60" name="A5/SCL">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
  <connector id="connector5" name="A5/SCL">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector85" name="RESET">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector84" name="ioref">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector59" name="A4/SDA">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
  <connector id="connector4" name="A4/SDA">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector58" name="AREF">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>

```

A Fritzing Netlist Analysis Test Inputs

```
</connector>
</net>
<net>
  <connector id="connector55" name="D12/MISO">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector54" name="D11 PWM/MOSI">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector53" name="D10 PWM/SS">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector52" name="D9 PWM">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector51" name="D8">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector91" name="N/C">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector3" name="A3">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector2" name="A2">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector90" name="VIN">
    <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
  </connector>
</net>
</netlist>
```

A.2 Test Case 2 Netlist

```

<?xml version='1.0' encoding='UTF-8'?>
<!-- Created with Fritzing (http://www.fritzing.org/) -->
<netlist sketch="Blink.fzz" date="Sat Aug 25 16:14:25 2018">
  <net>
    <connector id="connector1" name="anode">
      <part title="Red (633nm) LED" id="18713940" label="LED2"/>
    </connector>
    <connector id="connector56" name="D13/SCK">
      <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
    </connector>
    <connector id="connector1" name="anode">
      <part title="Red (633nm) LED" id="18713140" label="LED1"/>
    </connector>
  </net>
  <net>
    <connector id="connector0" name="cathode">
      <part title="Red (633nm) LED" id="18713140" label="LED1"/>
    </connector>
    <connector id="connector57" name="GND">
      <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
    </connector>
    <connector id="connector88" name="GND">
      <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
    </connector>
    <connector id="connector89" name="GND">
      <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
    </connector>
    <connector id="connector0" name="cathode">
      <part title="Red (633nm) LED" id="18713940" label="LED2"/>
    </connector>
  </net>
  <net>
    <connector id="connector1" name="A1">
      <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
    </connector>
  </net>
  <net>
    <connector id="connector0" name="A0">
      <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
    </connector>
  </net>
  <net>
    <connector id="connector68" name="D7">
      <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
    </connector>
  </net>
  <net>
    <connector id="connector67" name="D6_PWM">
      <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
    </connector>
  </net>
  <net>
    <connector id="connector66" name="D5_PWM">
      <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
    </connector>
  </net>
  <net>
    <connector id="connector65" name="D4">
      <part title="Arduino Uno (Rev3)" id="18460860" label="Arduino1"/>
    </connector>
  </net>

```

A Fritzing Netlist Analysis Test Inputs

```
</connector>
<connector id="connector89" name="GND">
  <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
</connector>
<connector id="connector0" name="VSS">
  <part title="LCD screen" id="17208270" label="LCD1"/>
</connector>
<connector id="connector0" name="leg1">
  <part title="Rotary Potentiometer (Small)" id="18203450"
        label="Contrast1"/>
</connector>
<connector id="connector4" name="R/W">
  <part title="LCD screen" id="17208270" label="LCD1"/>
</connector>
</net>
<net>
  <connector id="connector1" name="Pin 1">
    <part title="220Ω Resistor" id="17208290" label="R2"/>
  </connector>
  <connector id="connector1" name="VDD">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
  <connector id="connector87" name="5V">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
  <connector id="connector2" name="leg2">
    <part title="Rotary Potentiometer (Small)" id="18203450"
          label="Contrast1"/>
  </connector>
</net>
<net>
  <connector id="connector2" name="V0">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
  <connector id="connector1" name="wiper">
    <part title="Rotary Potentiometer (Small)" id="18203450"
          label="Contrast1"/>
  </connector>
</net>
<net>
  <connector id="connector0" name="Pin 0">
    <part title="220Ω Resistor" id="17208290" label="R2"/>
  </connector>
  <connector id="connector14" name="LED+">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
<net>
  <connector id="connector9" name="DB3">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
<net>
  <connector id="connector8" name="DB2">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
<net>
  <connector id="connector7" name="DB1">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
```

A.2 Test Case 2 Netlist

```
</connector>
</net>
<net>
  <connector id="connector6" name="DB0">
    <part title="LCD screen" id="17208270" label="LCD1"/>
  </connector>
</net>
<net>
  <connector id="connector1" name="A1">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector0" name="A0">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector68" name="D7">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector67" name="D6_PWM">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector62" name="D1/TX">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector61" name="D0/RX">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector86" name="3V3">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector60" name="A5/SCL">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
  <connector id="connector5" name="A5/SCL">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector85" name="RESET">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector84" name="ioref">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
```

A Fritzing Netlist Analysis Test Inputs

```
<net>
  <connector id="connector59" name="A4/SDA">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
  <connector id="connector4" name="A4/SDA">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector58" name="AREF">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector56" name="D13/SCK">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector53" name="D10 PWM/SS">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector52" name="D9 PWM">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector51" name="D8">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector91" name="N/C">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector3" name="A3">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector2" name="A2">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
<net>
  <connector id="connector90" name="VIN">
    <part title="Arduino Uno (Rev3)" id="18329900" label="Arduino1"/>
  </connector>
</net>
</netlist>
```

A.3 Test Case 3 Netlist

```

<?xml version='1.0' encoding='UTF-8'?>
<!-- Created with Fritzing (http://www.fritzing.org/) -->
<netlist sketch="case_study_arduino.fzz" date="Thu Aug 23 17:04:02 2018">
<net>
  <connector id="connector0" name="gnd">
    <part title="Basic Servo" id="18709310" label="J3"/>
  </connector>
  <connector id="connector89" name="GND">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
  <connector id="connector0" name="gnd">
    <part title="Basic Servo" id="18707750" label="J2"/>
  </connector>
  <connector id="connector57" name="GND">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
  <connector id="connector88" name="GND">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
  <connector id="connector0" name="cathode">
    <part title="Red (633nm) LED" id="18687990" label="LED3"/>
  </connector>
  <connector id="connector0" name="cathode">
    <part title="Infrared (850nm) LED" id="18684330" label="LED2"/>
  </connector>
  <connector id="connector11" name="GND 2">
    <part title="Humidity and Temperature Sensor RHT03" id="18686670" label=
      "RHT1"/>
  </connector>
  <connector id="connector0" name="GND">
    <part title="TSOP312.., 314.." id="18684100" label="IR2"/>
  </connector>
  <connector id="connector0" name="cathode">
    <part title="Infrared (850nm) LED" id="18683090" label="LED1"/>
  </connector>
  <connector id="connector0" name="GND">
    <part title="TSOP312.., 314.." id="18683180" label="IR1"/>
  </connector>
  <connector id="connector1" name="Vs">
    <part title="TSOP312.., 314.." id="18684100" label="IR2"/>
  </connector>
</net>
<net>
  <connector id="connector1" name="vcc">
    <part title="Basic Servo" id="18709310" label="J3"/>
  </connector>
  <connector id="connector87" name="5V">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
  <connector id="connector1" name="vcc">
    <part title="Basic Servo" id="18707750" label="J2"/>
  </connector>
  <connector id="connector1" name="Vs">
    <part title="TSOP312.., 314.." id="18683180" label="IR1"/>
  </connector>
  <connector id="connector1" name="Pin 1">
    <part title="10kΩ Resistor" id="18686700" label="R5"/>
  </connector>
  <connector id="connector8" name="Vcc">
    <part title="Humidity and Temperature Sensor RHT03" id="18686670" label=
      "RHT1"/>
  </connector>
</net>

```

A Fritzing Netlist Analysis Test Inputs

```
        "RHT1"/>
    </connector>
</net>
<net>
    <connector id="connector2" name="pulse">
        <part title="Basic Servo" id="18709310" label="J3"/>
    </connector>
    <connector id="connector56" name="D13/SCK">
        <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
    </connector>
</net>
<net>
    <connector id="connector2" name="pulse">
        <part title="Basic Servo" id="18707750" label="J2"/>
    </connector>
    <connector id="connector55" name="D12/MISO">
        <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
    </connector>
</net>
<net>
    <connector id="connector68" name="D7">
        <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
    </connector>
    <connector id="connector1" name="Pin 1">
        <part title="220Ω Resistor" id="18688300" label="R6"/>
    </connector>
</net>
<net>
    <connector id="connector67" name="D6 PWM">
        <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
    </connector>
    <connector id="connector9" name="Data-signal">
        <part title="Humidity and Temperature Sensor RHT03" id="18686670" label=
            "RHT1"/>
    </connector>
    <connector id="connector0" name="Pin 0">
        <part title="10kΩ Resistor" id="18686700" label="R5"/>
    </connector>
</net>
<net>
    <connector id="connector63" name="D2">
        <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
    </connector>
    <connector id="connector0" name="Pin 0">
        <part title="2kΩ Resistor" id="18684550" label="R2"/>
    </connector>
</net>
<net>
    <connector id="connector64" name="D3 PWM">
        <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
    </connector>
    <connector id="connector0" name="Pin 0">
        <part title="220Ω Resistor" id="18685040" label="R4"/>
    </connector>
</net>
<net>
    <connector id="connector52" name="D9 PWM">
        <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
    </connector>
    <connector id="connector0" name="Pin 0">
```

A.3 Test Case 3 Netlist

```
<part title="2kΩ Resistor" id="18683560" label="R1"/>
</connector>
</net>
<net>
<connector id="connector53" name="D10 PWM/SS">
<part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
</connector>
<connector id="connector0" name="Pin 0">
<part title="220Ω Resistor" id="18684890" label="R3"/>
</connector>
</net>
<net>
<connector id="connector2" name="Data">
<part title="TSOP312.., 314.." id="18684100" label="IR2"/>
</connector>
<connector id="connector1" name="Pin 1">
<part title="220Ω Resistor" id="18685040" label="R4"/>
</connector>
</net>
<net>
<connector id="connector2" name="Data">
<part title="TSOP312.., 314.." id="18683180" label="IR1"/>
</connector>
<connector id="connector1" name="Pin 1">
<part title="220Ω Resistor" id="18684890" label="R3"/>
</connector>
</net>
<net>
<connector id="connector0" name="Pin 0">
<part title="220Ω Resistor" id="18688300" label="R6"/>
</connector>
<connector id="connector1" name="anode">
<part title="Red (633nm) LED" id="18687990" label="LED3"/>
</connector>
</net>
<net>
<connector id="connector10" name="GND 1">
<part title="Humidity and Temperature Sensor RHT03" id="18686670" label=
    "RHT1"/>
</connector>
</net>
<net>
<connector id="connector1" name="Pin 1">
<part title="2kΩ Resistor" id="18683560" label="R1"/>
</connector>
<connector id="connector1" name="anode">
<part title="Infrared (850nm) LED" id="18683090" label="LED1"/>
</connector>
</net>
<net>
<connector id="connector1" name="Pin 1">
<part title="2kΩ Resistor" id="18684550" label="R2"/>
</connector>
<connector id="connector1" name="anode">
<part title="Infrared (850nm) LED" id="18684330" label="LED2"/>
</connector>
</net>
<net>
<connector id="connector1" name="A1">
<part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
```

A Fritzing Netlist Analysis Test Inputs

```
</connector>
</net>
<net>
  <connector id="connector0" name="A0">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
</net>
<net>
  <connector id="connector66" name="D5_PWM">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
</net>
<net>
  <connector id="connector65" name="D4">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
</net>
<net>
  <connector id="connector62" name="D1/TX">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
</net>
<net>
  <connector id="connector61" name="D0/RX">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
</net>
<net>
  <connector id="connector86" name="3V3">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
</net>
<net>
  <connector id="connector60" name="A5/SCL">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
  <connector id="connector5" name="A5/SCL">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
</net>
<net>
  <connector id="connector85" name="RESET">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
</net>
<net>
  <connector id="connector84" name="ioref">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
</net>
<net>
  <connector id="connector59" name="A4/SDA">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
  <connector id="connector4" name="A4/SDA">
    <part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
  </connector>
</net>
<net>
  <connector id="connector58" name="AREF">
```

```
<part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
</connector>
</net>
<net>
<connector id="connector54" name="D11_PWM/MOSI">
<part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
</connector>
</net>
<net>
<connector id="connector51" name="D8">
<part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
</connector>
</net>
<net>
<connector id="connector91" name="N/C">
<part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
</connector>
</net>
<net>
<connector id="connector3" name="A3">
<part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
</connector>
</net>
<net>
<connector id="connector2" name="A2">
<part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
</connector>
</net>
<net>
<connector id="connector90" name="VIN">
<part title="Arduino Uno (Rev3)" id="18683060" label="Part1"/>
</connector>
</net>
</netlist>
```

B Parallax Source Code Analysis Test Inputs/Outputs

B.1 Test Case 1 Parallax Source Input

```
#include <stdio.h>
#include "abdrive.h"
#include "dht22.h"
#include "simpletools.h"

int irLeft, irRight;

int humPin = 6;
int ledPin = 8;
float humThresh = 25.0;

int main()
{
    low(26);
    low(27);

    drive_setRampStep(12);

    while(1)
    {

        dht22_read(humPin);
        float humidity = dht22_getHumidity();
        humidity = humidity / 10.0;
        float temp = dht22_getTemp(CELSIUS) / 10.0;
        myPrintf(humidity);
        putchar('\n');
        myPrintf(temp);
        putchar('\n');

        if(temp > humThresh) {
            high(ledPin);
        } else {
            low(ledPin);
        }

        freqout(11, 1, 38000);
        irLeft = input(10);

        freqout(1, 1, 38000);
        irRight = input(2);

        if(irRight == 1 && irLeft == 1) {
            drive_rampStep(128, 128);
        } else if(irLeft == 0 && irRight == 0)
            drive_rampStep(-128, -128);
        else if(irRight == 0)
            drive_rampStep(-128, 128);
        else if(irLeft == 0)
            drive_rampStep(128, -128);
    }
}

// https://stackoverflow.com/questions/23191203/convert-float-to-string-without-sprintf
void myPrintf(float fVal)
{
    char result[100];
    int dVal, dec, i;
```

B Parallax Source Code Analysis Test Inputs/Outputs

```
fVal += 0.005;

dVal = fVal;
dec = (int)(fVal * 100) % 100;

memset(result, 0, 100);
result[0] = (dec % 10) + '0';
result[1] = (dec / 10) + '0';
result[2] = ' ';

i = 3;
while (dVal > 0)
{
    result[i] = (dVal % 10) + '0';
    dVal /= 10;
    i++;
}

for (i=strlen(result)-1; i>=0; i--) {
    putc(result[i], stdout);
}
}
```

B.2 Test Case 1 Parallax Source Output

```
#include <stdio.h>

int irLeft, irRight;
int humPin = 6;
int ledPin = 8;
float humThresh = 25.0;

void setup() {
    low(26);
    low(27);
    drive_setRampStep(12);
}

void loop() {
    dht22_read(humPin);
    float humidity = dht22_getHumidity();
    humidity = humidity / 10.0;
    float temp = dht22_getTemp(CELSIUS) / 10.0;
    if(temp > humThresh) {
        high(ledPin);
    } else {
        low(ledPin);
    }
    freqout(11, 1, 38000);
    irLeft = input(10);
    freqout(1, 1, 38000);
    irRight = input(2);
    if(irRight == 1 && irLeft == 1) {
        drive_rampStep(128, 128);
    } else if(irLeft == 0 && irRight == 0)
        drive_rampStep(-128, -128);
    else if(irRight == 0)
        drive_rampStep(-128, 128);
    else if(irLeft == 0)
        drive_rampStep(128, -128);
    }

    void myPrintf(float fVal)
    {
        char result[100];
        int dVal, dec, i;

        fVal += 0.005;

        dVal = fVal;
        dec = (int)(fVal * 100) % 100;

        memset(result, 0, 100);
        result[0] = (dec % 10) + '0';
        result[1] = (dec / 10) + '0';
        result[2] = '.';

        i = 3;
        while (dVal > 0)
        {
            result[i] = (dVal % 10) + '0';
            dVal /= 10;
            i++;
        }
    }
}
```

B Parallax Source Code Analysis Test Inputs/Outputs

```
    for (i=strlen(result)-1; i>=0; i--) {
        putc(result[i], stdout);
    }

    int input(int pin) {
        // TODO: complete method
    }

    int dht22_getHumidity() {
        // TODO: complete method
    }

    char dht22_read(int dht_pin) {
        // TODO: complete method
    }

    void low(int pin) {
        // TODO: complete method
    }

    int dht22_getTemp(char temp_units) {
        // TODO: complete method
    }

    void drive_setRampStep(int stepsize) {
        // TODO: complete method
    }

    void freqout(int pin, int msTime, int frequency) {
        // TODO: complete method
    }

    void drive_rampStep(int left, int right) {
        // TODO: complete method
    }

    void high(int pin) {
        // TODO: complete method
    }
```

C Case Studies Inputs/Outputs

C Case Studies Inputs/Outputs

C.1 CS-1 Output

```
#include <stdio.h>
#include "DHT.h"
#include "Servo.h"

const int basicServo1WritePin1 = 12;
const int humidityandTemperatureSensorRHT031ReadPin1 = 6;
const int tS2ReadPin1 = 3;
const int infrared850nmLED1WritePin1 = 2;
const int tS1ReadPin1 = 10;
const int infrared850nmLED2WritePin1 = 9;
const int basicServo2WritePin1 = 13;
const int red633nmLED1WritePin1 = 7;

int irLeft, irRight;
int humPin = 6;
int ledPin = 8;
float humThresh = 25.0;

void setup() {
    low(26);
    low(27);
    drive_setRampStep(12);
}

void loop() {
    dht22_read(humPin);
    float humidity = dht22_getHumidity();
    humidity = humidity / 10.0;
    float temp = dht22_getTemp(CELSIUS) / 10.0;
    if(temp > humThresh) {
        high(ledPin);
    } else {
        low(ledPin);
    }
    freqout(11, 1, 38000);
    irLeft = input(10);
    freqout(1, 1, 38000);
    irRight = input(2);
    if(irRight == 1 && irLeft == 1) {
        drive_rampStep(128, 128);
    } else if(irLeft == 0 && irRight == 0)
        drive_rampStep(-128, -128);
    else if(irRight == 0)
        drive_rampStep(-128, 128);
    else if(irLeft == 0)
        drive_rampStep(128, -128);
}

void myPrintf(float fVal)
{
    char result[100];
    int dVal, dec, i;

    fVal += 0.005;

    dVal = fVal;
    dec = (int)(fVal * 100) % 100;

    memset(result, 0, 100);
    result[0] = (dec % 10) + '0';
```

```

result[1] = (dec / 10) + '0';
result[2] = '.';

i = 3;
while (dVal > 0)
{
    result[i] = (dVal % 10) + '0';
    dVal /= 10;
    i++;
}

for (i=strlen(result)-1; i>=0; i--) {
    putc(result[i], stdout);
}

int input(int pin) {
    // TODO: complete method
}

int dht22_getHumidity() {
    // TODO: complete method
}

char dht22_read(int dht_pin) {
    // TODO: complete method
}

void low(int pin) {
    // TODO: complete method
}

int dht22_getTemp(char temp_units) {
    // TODO: complete method
}

void drive_setRampStep(int stepsize) {
    // TODO: complete method
}

void freqout(int pin, int msTime, int frequency) {
    // TODO: complete method
}

void drive_rampStep(int left, int right) {
    // TODO: complete method
}

void high(int pin) {
    // TODO: complete method
}

```

C Case Studies Inputs/Outputs

C.2 CS-1 Migrated Code

```
#include <stdio.h>
#include "DHT.h"
#include "Servo.h"

// Add definition
#define DHTTYPE DHT22

const int basicServo1WritePin1 = 12;
const int humidityandTemperatureSensorRHT031ReadPin1 = 6;
const int tS2ReadPin1 = 3;
const int infrared850nmLED1WritePin1 = 2;
const int tS1ReadPin1 = 10;
const int infrared850nmLED2WritePin1 = 9;
const int basicServo2WritePin1 = 13;
const int red633nmLED1WritePin1 = 7;

int irLeft;
int irRight;
// Edit: int humPin = 6;
int humPin = humidityandTemperatureSensorRHT031ReadPin1;
// Edit: int ledPin = 8;
int ledPin = red633nmLED1WritePin1;
float humThresh = 25.0;

// Add definition
char CELSIUS = 0;

// Add decl.
DHT humiditySensor(humPin, DHTTYPE);

// Add decl.
Servo servoLeft;
Servo servoRight;

void setup() {
  Serial.begin(9600);

  humiditySensor.begin();

  low(26);
  low(27);
  drive_setRampStep(12);

  // Add servo setup
  servoLeft.attach(basicServo2WritePin1);
  servoRight.attach(basicServo1WritePin1);

  // Add pinMode setup
  pinMode(infrared850nmLED1WritePin1, OUTPUT);
  pinMode(infrared850nmLED2WritePin1, OUTPUT);
  pinMode(tS2ReadPin1, INPUT);
  pinMode(tS1ReadPin1, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  dht22_read(humPin);
  float humidity = dht22_getHumidity();
  // humidity = humidity / 10.0; Delete: Parallax libraries return in tenths
  //float temp = dht22_getTemp(CELSIUS) / 10.0;
```

C.2 CS-1 Migrated Code

```
float temp = dht22_getTemp(CELSIUS);

// Delete putchar
myPrintf(humidity);
myPrintf(temp);

if(temp > humThresh) {
    high(ledPin);
} else {
    low(ledPin);
}

// Replace w. generated variable
freqout(irradiated850nmLED1WritePin1, 1, 38000);
irRight = input(tS2ReadPin1);

// Replace w. generated variable
freqout(irradiated850nmLED2WritePin1, 1, 38000);
irLeft = input(tS1ReadPin1);

if(irRight == 1 && irLeft == 1) {
    drive_rampStep(128, 128);
} else if(irLeft == 0 && irRight == 0)
    drive_rampStep(-128, -128);
else if(irRight == 0)
    drive_rampStep(-128, 128);
else if(irLeft == 0)
    drive_rampStep(128, -128);
}

// Replace
void myPrintf(float fVal) {
    Serial.println(fVal);
}

void dht22_read(int dht_pin) { // Change return type char -> void
    // Do nothing
}

float dht22_getHumidity() { // Change return type int -> float
    return humiditySensor.readHumidity();
}

void high(int pin) {
    digitalWrite(pin, HIGH);
}

void freqout(int pin, int msTime, int frequency) {
    tone(pin, frequency, msTime*8);
    delay(1);
}

int input(int pin) {
    int val = digitalRead(pin);
    delay(1);
    return val;
}
```

C Case Studies Inputs/Outputs

```
void drive_rampStep(int left, int right) {
    servoLeft.writeMicroseconds(1500 + left);
    servoRight.writeMicroseconds(1500 - right);
    delay(20);
}

float dht22_getTemp(char temp_units) { // Change return type int -> float
    return humiditySensor.readTemperature();
}

void low(int pin) {
    digitalWrite(pin, LOW);
}

void drive_setRampStep(int stepsize) {
    // Do nothing
}
```

C.3 CS-2 Input

```

int main()
{
    low(26);
    low(27);

    drive_setRampStep(12);

    while(1)
    {

        dht22_read(humPin);
        float humidity = dht22_getHumidity();
        humidity = humidity / 10.0;
        float temp = dht22_getTemp(CELSIUS) / 10.0;

        if(temp > humThresh) {
            high(ledPin);
        } else {
            low(ledPin);
        }

        freqout(11, 1, 38000);
        irLeft = input(10);

        freqout(1, 1, 38000);
        irRight = input(2);

        if(irRight == 1 && irLeft == 1) {
            drive_rampStep(128, 128);
        } else if(irLeft == 0 && irRight == 0)
            drive_rampStep(-128, -128);
        else if(irRight == 0)
            drive_rampStep(-128, 128);
        else if(irLeft == 0)
            drive_rampStep(128, -128);
    }
}

```

C.4 CS-2 Output

```
void setup() {
    low(26);
    low(27);
    drive_setRampStep(12);
}

void loop() {
    dht22_read(humPin);
    float humidity = dht22_getHumidity();
    humidity = humidity / 10.0;
    float temp = dht22_getTemp(CELSIUS) / 10.0;
    if(temp > humThresh) {
        high(ledPin);
    } else {
        low(ledPin);
    }
    freqout(11, 1, 38000);
    irLeft = input(10);
    freqout(1, 1, 38000);
    irRight = input(2);
    if(irRight == 1 && irLeft == 1) {
        drive_rampStep(128, 128);
    } else if(irLeft == 0 && irRight == 0)
        drive_rampStep(-128, -128);
    else if(irRight == 0)
        drive_rampStep(-128, 128);
    else if(irLeft == 0)
        drive_rampStep(128, -128);
}
```