The University of York Department of Computer Science

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

Model-driven software migration between microprocessors

Sophie Wood

Version 0.1, 2018-May-16

Supervisor: Simos Gerasimou

Number of words = 0, as counted by `wc -w`.
This includes the body of the report, and Appendices TODO, but not TODO.

Abstract

TODO

Acknowledgements

TODO

Contents

1 Introduction

1.1 Background and Motivation

Bartels et al. define obsolescence as "materials, parts, devices, software, services and processes that become non-procurable from their original manufacturer or supplier" [**bartels2012strategies**]. Both software and hardware can be subject to obsolescence problems.

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 [**bartels2012strategies**]. This is a particular issue in the defence/aerospace sectors as the typical life cycle of a system is 20-30 years or longer [**rojo20100bsolescence**]. As such, parts will 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" [**singh20060bsolescence**].

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." [**bartels2012strategies**]

There are three main causes of software obsolescence [**sandborn20070bsolescence**]:

(1) Logistical: digital media obsolescence, formatting, or degradation limits or terminates access to software

(2) Functional: hardware, requirements, or other software changes to the system obsolete the functionality of the software

(3) Technological: the sales and/or support for commercial off the shelf (COTS) software terminates

A particular issue is that hardware obsolescence can drive software obsolescence and vice versa. Software obsolescence can also be caused by 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 [**rajagopal2014software**]. 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 [**singh2006obsolescence**].

Obsolescence mitigation can be costly, particularly in the case of systems with a long life cycle. The US Navy estimates that obsolescence problems can cost up to $750 million annually [**adams2005getting**]. 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 [**sandborn2008designing**].

Strategies for obsolescence mitigation/management have generally focused on hardware obsolescence problems. This is despite software obsolescence costs often equalling or exceeding that of hardware. Proactive measures to mitigate obsolescence (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 include software license downgrades (i.e. customers can adapt legacy versions of the software by purchasing a licence for the latest version and applying that to the older product); purchasing the source code and third party support (i.e. a third party will take over maintenance of the legacy software). If these strategies are insufficient, the legacy system may have to be redeveloped or rehosted [**sandborn2007obsolescence**]. However, many such software modernisation projects are abandoned or not completed within the planned timescale/budget [**kowalczyk2009model**].

Model-driven software modernisation (MDSM) is a set of strategies that use model-driven engineering (MDE) to address the above issues by (partially) automating the process of modernisation. These approaches have already proved feasible by Kowalczyk and Kwiecińska who demonstrated the modernisation of a project written in Java 1.4 and the Hibernate 2.x framework, which uses XML mappings, into Java 1.5 and Hibernate 3.x which uses Java Persistence API (JPA) annotations [**kowalczyk2009model**]. Additionally, a model-driven approach is employed in industry by the Sodifrance company who have migrated a large-scale banking system (at a size of around one million lines of code) to J2EE [**fleurey2007model**].

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 from a legacy hardware platform to a modern more powerful platform" [**gerasimou2017technical**]. 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

TODO

1.3 Project Scope

TODO

1.4 Report Structure

TODO

2 Literature Review

3 Methodology

4 Requirements

5 Design and Implementation

6 Evaluation

7 Conclusion