# Three Way Merge for Feature Model Evolution Plans

Eirik Halvard Sæther

Thesis submitted for the degree of
Master in Informatics: Programming and Systems Architecture

60 credits

Department of Informatics

Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2021

# Three Way Merge for Feature Model Evolution Plans

Eirik Halvard Sæther

# Acknowledgements

Thanks to Ida

Thanks to Ingrid and Crystal

Thanks to Germans, everyone on the LTEP project for input etc.

# Abstract

[[TODO: write abstract]]

Feature Model Evolution Plans is intended to help ease the development of software product lines (SPLs). Feature Models allow software engineers to explicitly encode the similarities and differences of an SPL. However, due to the changing nature of an SPL, Evolution Plans allows for representing the *evolution* of a feature model, not just the feature model as a single point in time.

Evolution planning of an SPL is often a dynamic, changing process, due to changing demands of the focus of development. The evolution planning is often not just done by a single engineer, but multiple engineers, working separately and independent of each other. Due to these factors, the need to unify and synchronize the changes the evolution plan emerges.

In this thesis, we develop a merge tool for Feature Model Evolution Plans. The core of the tool is a three-way merge algorithm. Given two different versions of an evolution plan, together with the common evolution plan they were derived from, the merge algorithm will attempt to merge all the different changes from both versions. If the merges are unifiable, the algorithm will succeed and yield the merged result containing the changes from both versions. However, if the changes are conflicting in any way, breaking the structure or semantics of evolution plans, the algorithm will stop, telling the user the reason of failure.

# Contents

# List of Figures

# List of Tables

# Preface

[[TODO: write better and more]] something about the LTEP project

something about summer project?

# Chapter 1

# Introduction

## 1.1 Chapter Overview

## 1.2 Project Source Code

# Chapter 2

# Background

## 2.1 Software Product Lines

## 2.2 Evolution Planning

## 2.3 Haskell and Algebraic Data Types

# Chapter 3

# Formal Semantics of Feature Model Evolution Plans

# Chapter 4

# Three Way Merge Algorithm

## 4.1 Algorithm Overview

### 4.1.1 Three-Way Merging of Evolution Plans

The three-way merge algorithm for feature model evolution plans will take two different versions of an evolution plan, *version 1* and *version 2*, and attempt to merge the evolution plans into a single plan. In order to do so, a third evolution plan has to be provided, which is the common evolution plan they were derived from. The common evolution plan, called *base*, will implicitly provide information about what things were added, removed and changed in each of the derived evolution plans.

### 4.1.2 Soundness Assumption

The three-way merge algorithm will assume that the three evolution plans provided are sound. By assuming the soundness of the plans, the algorithm can leverage this to create a better merge result. But more importantly, the assumption is based around the fact that there is no point in merging an evolution plan you know violates soundness in some way.

### 4.1.3 Algorithm Phases

In order to merge the different versions of the evolution plan, the algorithm is separated into several distinct phases. The different steps and phases of the algorithm can be seen in Figure 4.1.

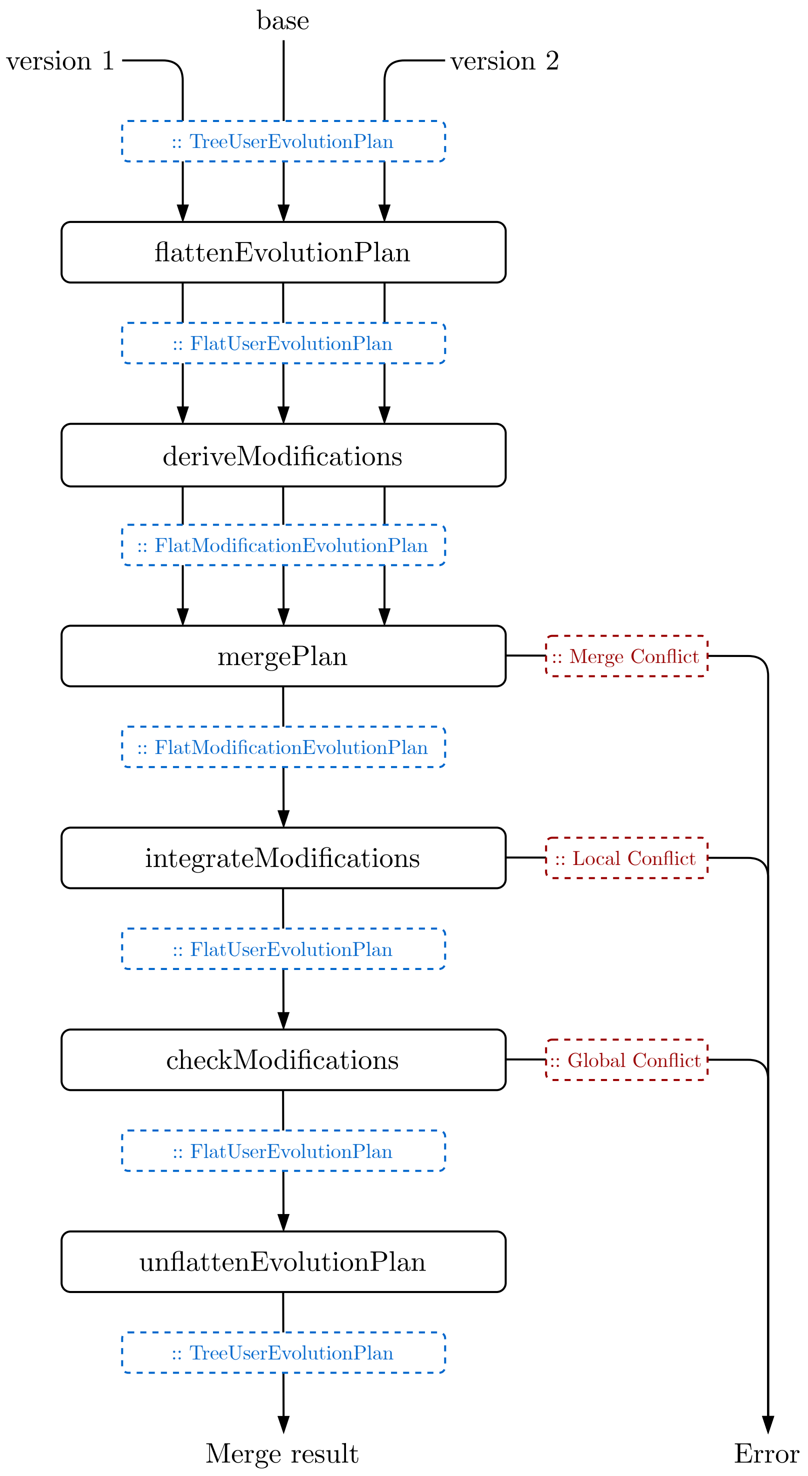

Figure 4.1: Outline of the three-way merge algorithm

The first phase is transforming the three different evolution plans into representations that is more suitable for merging. This includes converting both the way feature models are represented as well as the way the entire evolution plan is represented. This phase includes the `flattenEvolutionPlan` and `deriveModifications`, which is described in further detail in 4.2 on the following page

After changing the way evolution plans are represented, the second phase of the algorithm will calculate the differences between the *base* evolution plan and both derived evolution plans, *version 1*, and *version 2*. This will let us know what were added, changed and removed in each of the derived evolution plans. This phase is part of the `mergePlan` function, which is described in further detail in 4.3 on the next page

The information from the previous phase will be used to create a single merged evolution plan. This evolution plan is simply just the *base* evolution plan integrated with all the changes from *version 1* and *version 2*. This phase is part of the `mergePlan` function, which is described in further detail in 4.4 on the following page

Now that a single merged evolution plan is provided, the last step is to ensure that the plan is following the structural and semantic requirements of an evolution plan. Merging all changes from both versions might yield various inconsistencies. This includes structural conflicts such as orphan features, entire subtrees forming cycles, removing non-empty features, etc. The last phase includes converting back to the original representation, as well as ensuring soundness while doing so. This phase is part of the `integrateModifications`, `checkModifications` and `unflattenEvolutionPlan` functions, which is explained further in 4.5 on the next page

### 4.1.4 Conflicts

During the different phases of the merge algorithm, different kind of conflicts or errors could occur. Depending on what part of the algorithm a conflict occurred, the conflicts might be either a *merge*, *local* or *global* conflict. For example, if both versions perform different operations on the same feature at the same time point, the

## 4.2 Converting To a Suitable Representation

### 4.2.1 Representing Feature Models

### 4.2.2 Representing Evolution Plans

## 4.3 Detecting the Changes Between Versions

## 4.4 Merging Intended Changes

## 4.5 Ensuring structural and semantic soundness of the merge result

# Chapter 5

# Conclusion and Future Work