Self-hosting the Envision Visual Programming Environment

Master Thesis Project Description

Patrick Lüthi

Supervised by Dimitar Asenov, Prof. Dr. Peter Müller
ETH Zürich

May 7, 2015

1 Introduction

In this project we are going to work on Envision [1], a visual programming system for object-oriented programs. Visual programming environments like Envision use a variety of visual elements to represent programs. Their goal is to visualize the code in ways that help the programmer work more efficiently than with just text.

The goal of this project is to make Envision a self-hosted programming environment. The term self-hosting describes a software's ability to create variants of itself (for example a compiler compiling itself). Therefore the goal is to enable development of Envision in itself. Envision is written in `C++` and up to the present it was developed using Eclipse and Qt Creator.

Reaching self-hosting stage is an important milestone for programming tools. The main purpose of programming tools is to create or edit software and being able to self-host shows their applicability.

We are not aware of any other visual programming system that is self-hosting. Visual programming tools have been mostly successful in educational, small scale or domain specific applications. Envision's `C++` codebase on the other hand is significant in size and not written in any domain specific way making self-hosting Envision a challenging task.

2 Challenges and Core Goals

In this section we are going to outline the main challenges and core goals of the project.

- **Code generation**
  Envision's source code makes use of the `C++` preprocessor. We have to find a way to bridge the gap between Envision's internal tree representation of programs and the textual nature of `C++` macros while retaining all semantics during transformation which is generally a hard problem as previous work [2, 3] has shown.
  The solution has to support at least all usages of preprocessor directives used in Envision's source code.

- **Import from `C++` source files**
  All `C++` features used in Envision's source code have to be correctly interpreted in Envision. The current import feature of Envision has to be extended because it does not yet support processing of macros, comments and some newer `C++` features.

- **Export to `C++` source files**
  Envision uses a tree representation for programs whereas `C++` code is split into header and source text files.
  We have to decide how to generate both header and `C++` source files from a single Envision source tree by analysing the class dependencies. Using the results from the analysis we will determine the minimal amount of information required to put into the header files in order for the program to work.

- **Non-standard C++ features**
  Envision's current C++ code base uses some features provided by the Qt meta-object compiler (MOC):

  - The MOC does an additional pass over the code performing code generation before the C++ preprocessor. This is used for embedding resources and setting up the Qt meta-object property system.
  - The MOC also provides a signals and slots concept that enables an event driven program flow. The signals and slots mechanism is a variation of the publish-subscribe pattern.

  These non-standard C++ features have to be interpreted correctly when importing and made use of when exporting to retain compatibility with other development tools.

3 Extensions

The following list contains possible features to work on after the core goals have been completed:

- **Preserve Documentation**
  Envision supports advanced documentation features (images, tables, etc.). There exists no trivial mapping between C++ text files and this type of documentation. This extension's goal is to preserve such data during an export-import cycle (for example by referencing secondary files in C++ source files).

- **Improve library support**
  To enable efficient working in an IDE, auto completion, jump to definition, find all references and other related features are essential. In order to provide such features for external libraries we would have to find a way to generate or import the headers of the libraries alongside a project when it is imported.

- **Support QMake**
  Envision's current C++ code base uses the QMake build system.

  - In order to output meaningful error messages during compilation and other related services, Envision has to support the usage of QMake's options while self-hosting.
  - QMake files store relevant information regarding the build process of the project (i.e. what classes are part of it etc.). If a user modifies a program that uses the QMake build system in Envision by adding or removing a class from said project then the related QMake files have to be adjusted as well.

4 Approach

To make Envision self-hosted we are going to provide functionalities for importing code written in `C++` into Envision and for exporting Envision projects to `C++` code. One export-import cycle should result in the same program (round-trip requirement) as illustrated by figure 4.1. This enables the usage of other development tools alongside Envision until all features are working properly.

The system should be running on simple inputs (using few language features) early on and kept in a running state during all phases of development while it is gradually extended to handle more complex code. The final goal is to be able to handle at least all the features needed to process Envision in itself.

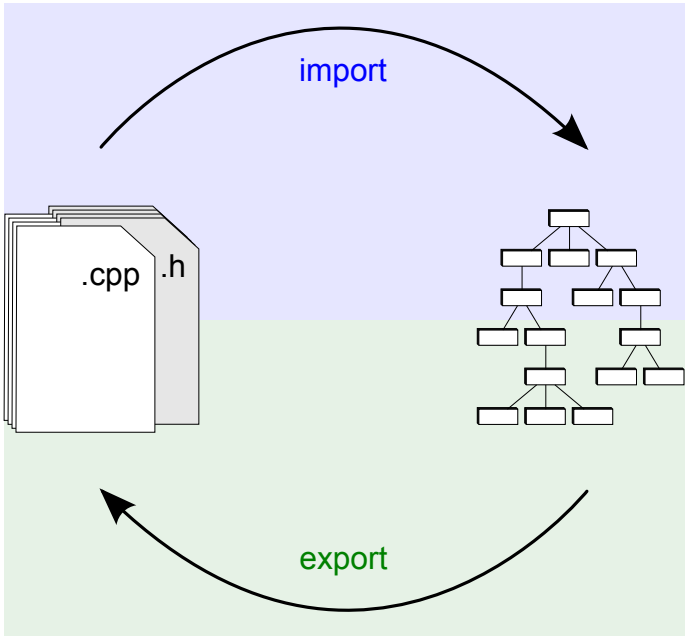

Figure 4.1: A visualization of the import-export round-trip. One full cycle should result in the original program.

5 Schedule

We are going to work on this project for exactly 6 months. The following table shows an approximate schedule:

Date	Time	Task
11.05.2015	2 weeks	Explore existing work on code generators
25.05.2015	7 weeks	Design a code generator framework for Envision that can represent `C++` preprocessor directives
13.07.2015	1 week	Familiarize with Envision's current import/export systems and their limitations
20.07.2015	2 weeks	Design and implement correct interpretation of macros and comments from `C++` code in Envision
03.08.2015	3 weeks	Find a way to interpret non-standard `C++` features when importing from `C++` code
24.08.2015	3 weeks	Find a way to use non-standard `C++` features when exporting from Envision
14.09.2015	2 weeks	Design a dependency analysis to generate `C++` header files from Envision source trees
28.09.2015	4 weeks	Extensions
26.10.2015	2 weeks	Report
11.11.2015	-	End

References

[1] D. Asenov and P. Müller. Envision: A fast and flexible visual code editor with fluid interactions (overview). In *Visual Languages and Human-Centric Computing (VL/HCC)*, pages 9–12, 2014.

[2] Alejandra Garrido and Ralph Johnson. Challenges of refactoring c programs. In *Proceedings of the International Workshop on Principles of Software Evolution*, IWPSE '02, pages 6–14, New York, NY, USA, 2002. ACM.

[3] Bjarne Stroustrup, Aditya Kumar, and Andrew Sutton. Rejuvenating c++ programs through demacrofication. In *Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM)*, ICSM '12, pages 98–107, Washington, DC, USA, 2012. IEEE Computer Society.