Technical Specifications

Jason GROSSO

January, 2023

Abstract

This is the Technical Specifications for the project group 8 during the year 2022-23 at ALGOSUP. This document describes the practical aspects of the project such as its functionalities, which technologies were used and where, as well as the architectures used.

Contents

1 Introduction

1.1 Objective

The objective of the project is for us to implement Rust to FABGen, a binding generator developed by Harfang.

1.2 Project Team

This project will be made over the course of 7 weeks, our Project Manager is David and is in charge of organizing the work and dividing the tasks to the team.. Max is the Program Manager, his job is to define the high level functionalities. Jason is the Tech Lead, his job is to chose how the functionalities will be implemented and which technologies will be used. Louis is in charge of the Quality Assurance, he will be testing every functionality of the project to find and point out any bug found, finally, Leo is our software engineer, and he will be the one responsible of implementing most of the functionalities.

2 Overview

2.1 Current State

Currently SWIG is the main tool used to do bindings, but it has many problems, such as missing features, old database, the use of a meta C like language making it really hard to support and the extreme difficulty of debugging due to the hiding of the type of variable by the code-base.

Harfang wanted to expand their user base, a way to do that is to add new languages and a simple way to add languages is to implement bindings, but as previously said SWIG is not usable, so Harfang decided to create their own binding generator: FABGen. However there are only 3 languages available in FABGen :

- Python
- Lua
- GOlang

In order to expand their pool of languages Harfang via ALGOSUP commissioned us to add Rust to FABGen

2.2 Scope

Item	In Scope	Out Of Scope
Rust Binding	X	
Unit Tests	X	
Reliability	X	
No Polymorphism At Run Time		X
Working Harfang3D-Rust Example		X
Ruby Binding		X

Table 1: Scope And Out Of Scope

2.3 Approach

To complete the project, we decided to use bindgen for the bindings and to translate the GOlang units tests to Rust as both of the languages are statically typed.

2.4 Risks and Assumptions

2.4.1 Assumption

We assume that our solution will be :

- Easy to support;
- Easy to extend/work on;
- With no missing features;
- Able to bind many C++ constructs;

2.4.2 Risk

Problems our project can encounter:

- Delays → use a planner and work overtime;
- OS related problems → use Docker or ask the school for a different OS;
- Difficult to use → follow what was done on the others languages;
- Difficult to work on → comment the code and keep the code as clear as possible;

- Low customers satisfaction → communicate regularly with the customers to update him on the improvements of the project and ask him to validate the direction of the project;

3 Proposition

3.1 Process

We use the Python description of the C++ library, pass them into FABGen, which will return a .c, .h, and a .rc file; by using the clang command we will transform the .c file into a .o file usable by every language. This file will be then called by the users via the .rc file to access the C++ library(see 1).

Here we will demonstrate using the GO version of the generator; we launch FABGen we use the command line in 2. This will start the process in 3 to create the bindings.

The first part consists of setting the generator to the chosen language, to do this bind.py we will go into the language file of the chosen language, and send the info to gen.py in order to activate the GO version. Once it is done it will send the info to bind.py.

The second part is a verification of the passed file in this case, harfang.py: it will execute lots of small functions between go.py gen.py and himself and it will also check some functionalities.

The third part mainly takes place in the std.py file of GOlang it consists of transforming the GO variable types to C variable types, after that bind.py will execute the bind generator, apply the clang command and print the files in the directory passed in the starting command.

3.2 Success Evaluation

To determine the success of the project we have two ways at our disposition:

- Harfang gave us a file with 30 unit tests, that we will use to quantify the percentage of work done;

- We also have access to harfang.py witch is an example of a .py file passed into FABGen we can use it to make sure our bindings are working.

Glossary

Bind Allow a programming language to use a library from another programming language.

Bindgen Tool used to automatically generate Rust bindings to C and C++ libraries

Binding generator Automatically creates a binding for any given language.

C++ Generalist programming language.

C Generalist programming language.

Clang command Transform a file into an object file.

Docker Docker packages software into standardized units called containers that have everything the software needs to run.

Harfang A French company based in Orleans that developed a 3D visualisation tool (Harfang3D) and a binding generator (FABGen).

GOlang / GO Programming language statically typed.

Library Reusable collection of prewritten code that can be used for optimisation.

Polymorphism In programming language theory polymorphism is defined as a function that can accept multiple types.

OS Stand for Operating System, is the software managing the hardware of the computer (exemple: Apple, Windows).

Unit tests Tests on a specific part of the code.

Ruby Programming language focusing on simplicity and productivity.

Rust Programming language statically typed.

Statically Typed Is said of a language where the variables types are known at run time.

SWIG Is a old binding generator encompassing multiple languages.

Diagrams

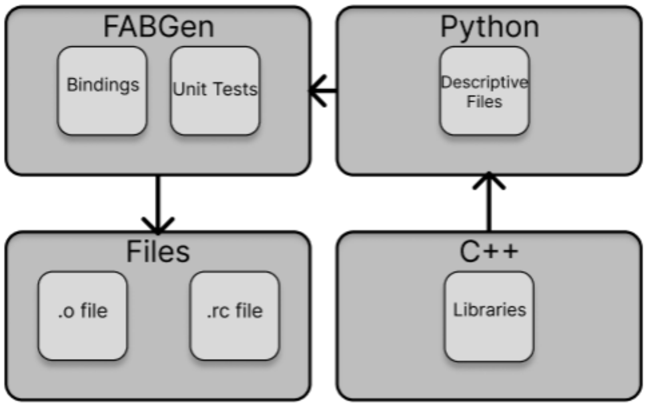
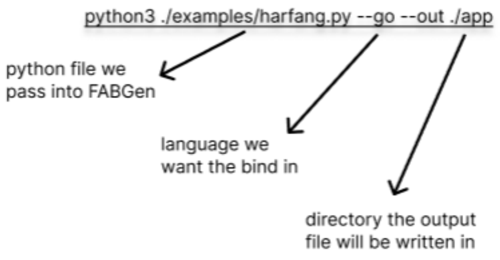


Figure 1: Project Architecture



```
python3 ./examples/harfang.py --go --out ./app
```

python file we pass into FABGen
language we want the bind in
directory the output file will be written in

Figure 2: Command example

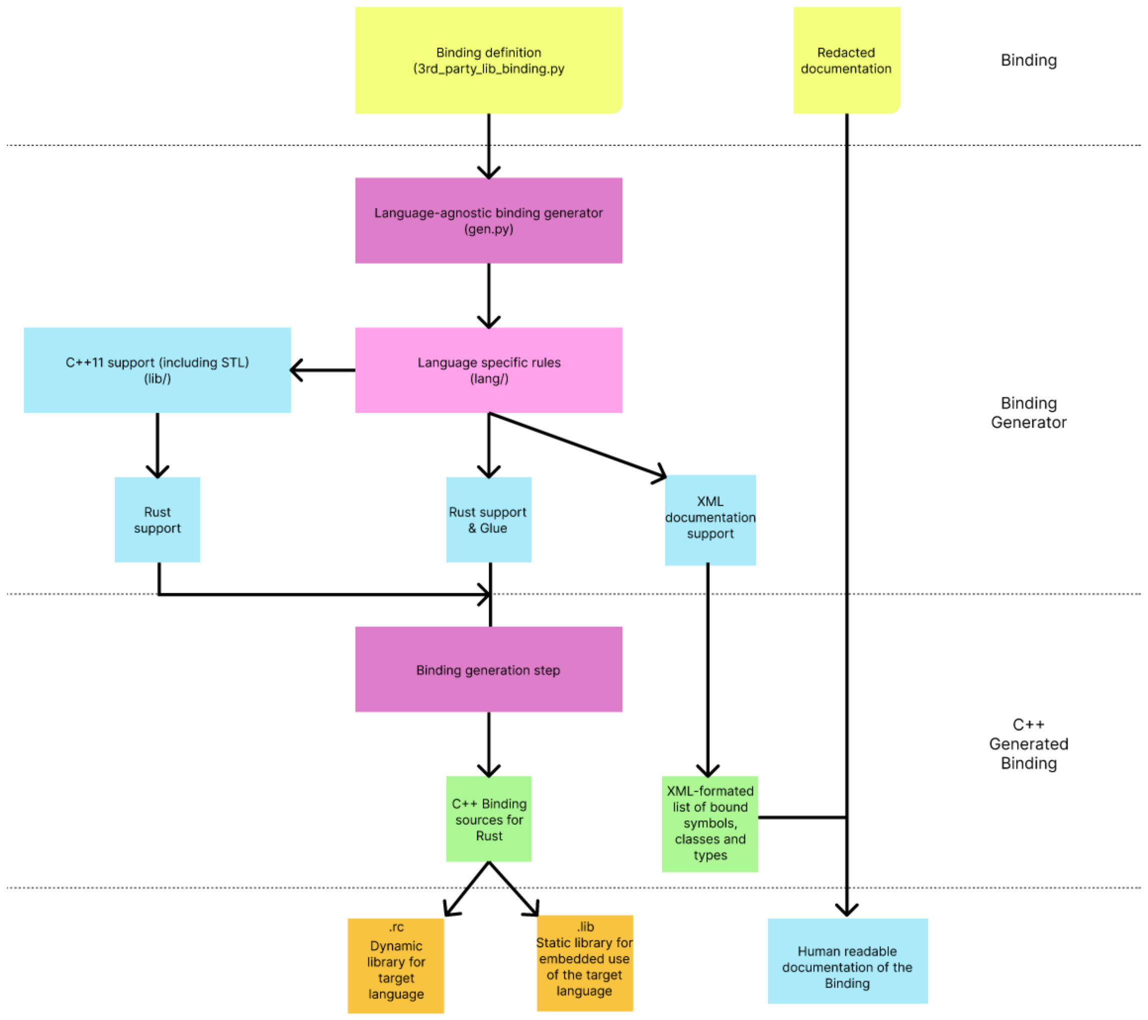

Binding definition
(3rd_party_lib_binding.py
Redacted documentation
Binding
Language-agnostic binding generator
(gen.py)
C++11 support (including STL)
(lib/)
Language specific rules
(lang/)
Binding Generator
Rust support
Rust support & Glue
XML documentation support
Binding generation step
C++ Generated Binding
C++ Binding sources for Rust
XML-formated list of bound symbols, classes and types
.rc
Dynamic library for target language
.lib
Static library for embedded use of the target language
Human readable documentation of the Binding

Figure 3: FABGen Architecture(GO example)