RustProof

Software Architecture Design Document

Drew Gohman
Matt O'Brien
Bradley Rasmussen
Sami Sahli
Michael Salter
Vincent Schuster
Matthew Slocum

Version 1.0 **05/30/16**

Table of Contents

1. Introduction

1.1. Purpose

The Software Architecture Document provides a comprehensive architectural overview of RustProof. It presents a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made about the system. This document describes the aspects of RustProof design that are considered to be architecturally significant; that is, those elements and behaviors that are most fundamental for guiding the construction of RustProof and for understanding this project as a whole.

1.2. Definitions, Acronyms, and Abbreviations

Attribute - A system of annotated declarations utilized by Rust.

Compiler - The compiler for Rust code, rustc, for which RustProof shall be a plugin.

Prover - The specific automated theorem proving program to be used by the system. Automated theorem proving software takes Verification Conditions and tests their validity, returning an example where a Verification Condition is invalid if one can be found.

Rust - The Rust programming language with which the Compiler and RustProof will deal.

RustProof - A Rust Verification Condition generator, the software this document describes the architecture of.

User - A human user who chooses to utilize RustProof. They will insert Attributes into their Rust code to be used by RustProof during compilation.

Verification Condition - A logical statement derived from a precondition Attribute and a Weakest Precondition, to be checked in a Prover.

Weakest Precondition - A logical statement derived from a postcondition Attribute and a body of Rust code, used to create a Verification Condition.

1.3. References

1) RustProof Software Requirements Document: https://github.com/RustVCG/RustVCG/blob/master/documents/RequirementsSpecificationDocument_v1.1.pdf

2. Use-Case View

2.1. Actors

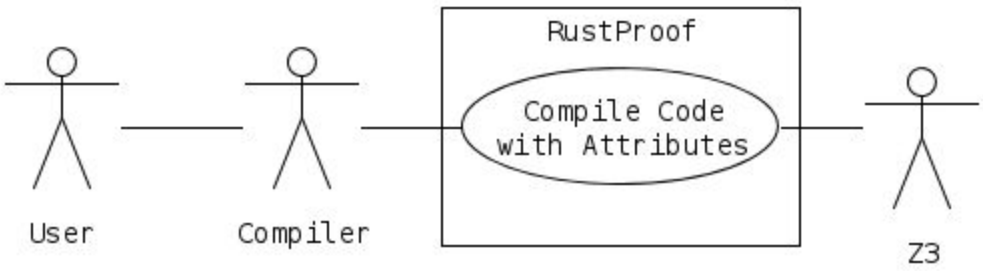
The following are the actors relevant to the RustProof plugin:

- **User** - A human user of the system. The User inserts RustProof Attributes to specify a precondition and/or postcondition for a Rust function, and then calls the Compiler to compile Rust code. The User also receives output from RustProof via the Compiler.

- **Compiler**- The Compiler for Rust, through which the User and RustProof communicate.
- **Z3** - The Prover, to which RustProof provides Verification Conditions generated from User code, and from which RustProof gets output related to the existence or nonexistence of counterexampes to said Verification Conditions.

2.2. Use-Case Realizations

RustProof is called by the Compiler for each Attribute and function it must process, and therefore RustProof really only has one use case, as specified in the Software Requirements Document [1]. Although the User initiates the process of compilation, the User does not directly interface with RustProof; all input and output of RustProof is to or from the Compiler and Z3. This functionality is further decomposed in other sections.



3. Logical View

The software architecture seperates functionality into logical modules to produce a system with high cohesion and low coupling.

The Parser module verifies the correctness of provided Attribute syntax and returns pre- and postconditions to the Control module.

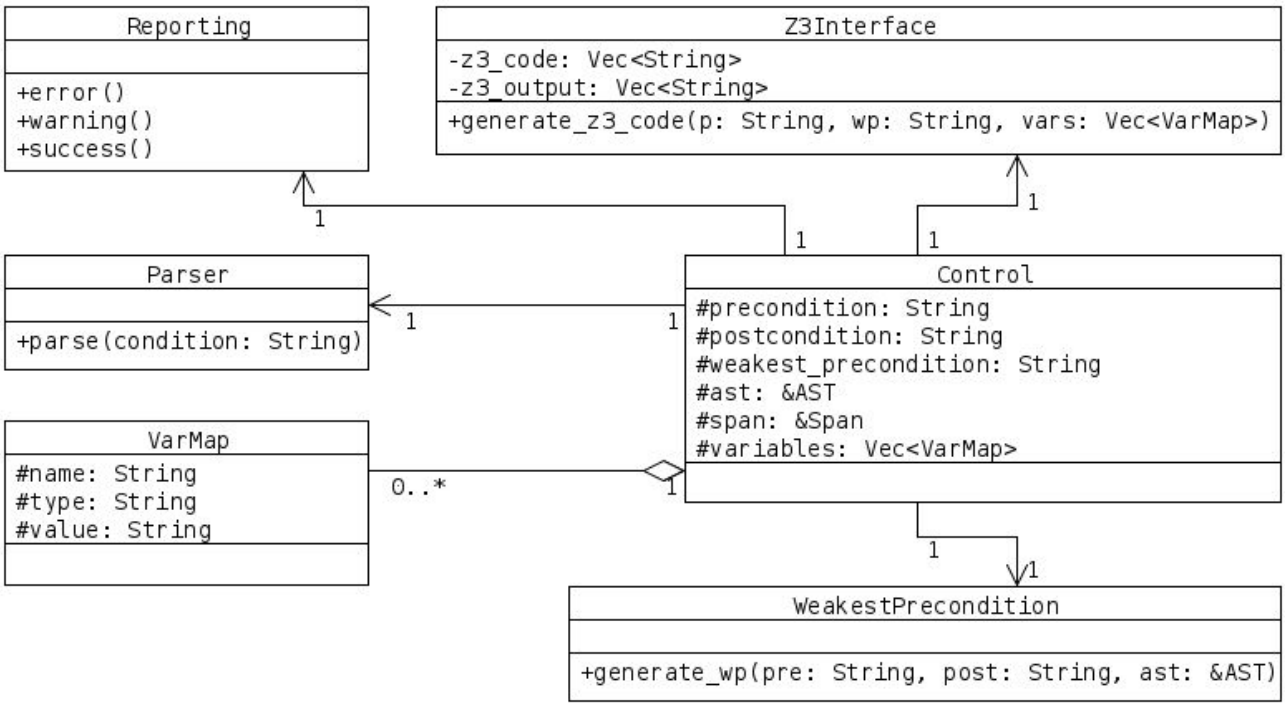
The Contol module is reponsible for control flow. It also stores the following data:

- Precondition - A String respresentation of attribute data generated by the Parser module.
- Postcondition - A String representation of attribute data generated by the Parser module.
- Weakest Precondition - A String representation of the weakest precondition, generated by the Weakest Precondition module.
- AST - An Abstract Syntax Tree representation of the attributed code.
- Span - A mapping from an AST node to source code.
- A vector of VarMaps

The VarMap entity contains strings representations of the name, type, and value of variables occuring in conditions.

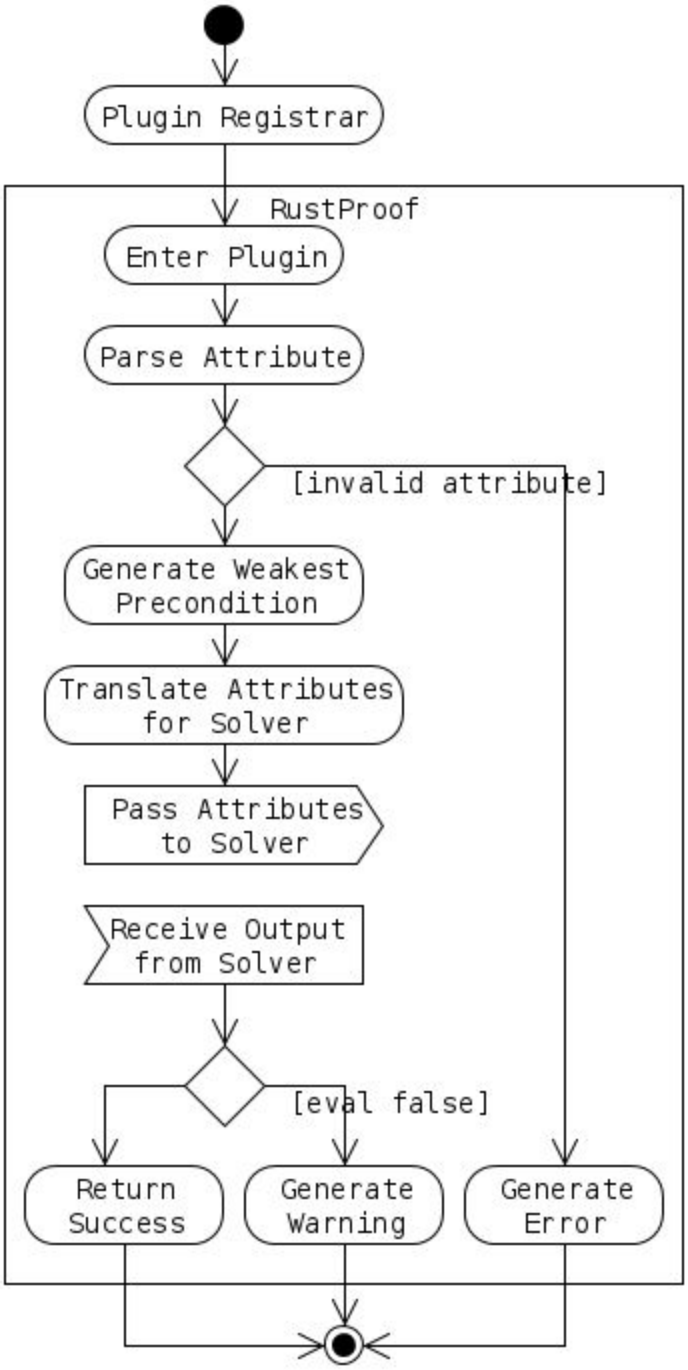
The Weakest Precondition module computes the Weakest Precondition and returns to the control module.

The Z3 Interface module sends variable mappings and a Verification Condition generated from the precondition and weakest precondition to Z3 and awaits output. It then returns any output from Z3 to the Control module.



4. Process View

Rustproof follows a procedural flow. The Control module delegates tasks to other modules that return results to the Control module. It then calls Z3, generating a return value that is reported to the compiler.



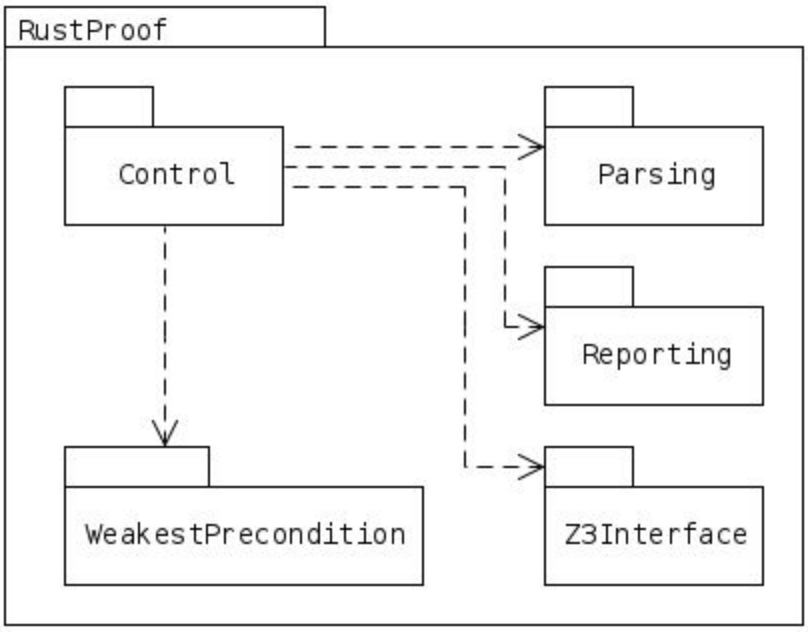
5. Module Decomposition View

Rustproof is composed of 5 modules:

- Control:

  This is the entry point to Rustproof. It interacts with the other 4 modules.

- Parser:
  Reads and checks the attributes.
- Weakest Precondition Generator:
  Generates the Weakest Precondition.
- Reporting:
  Reports to the compiler any warnings, errors, and successes.
- Z3 Interface:
  Interfaces with Z3.



6. Deployment View

Deployment of RustProof consists of its general presentation on Github with complete documentation, and its deployment on Cargo. The presentation on Github will include: code that follows a consistent style, concise comments at the beginning of files and functions within, and the documentation generated from Cargo describing the design and usage of RustProof and its attribute(s). The deployment of RustProof to Cargo is to allow users to easily download and use RustProof with their own programs.

7. Issues and concerns

- AST representation of user code may not be sufficient to generate a weakest precondition, in this case MIR (Rustc's intermediate representation) or tokentree representations will need to be explored.
- If Z3's interface does not seem adequate, the Z3 module will instead produce Verification Conditions in smtlib format.