EDB: Debugger for Ethereum's Programming Languages

Report #3: Requirements Specification
Advised by Dr. Jackowitz
University of Scranton

Andrew Plaza

November 4, 2018

Contents

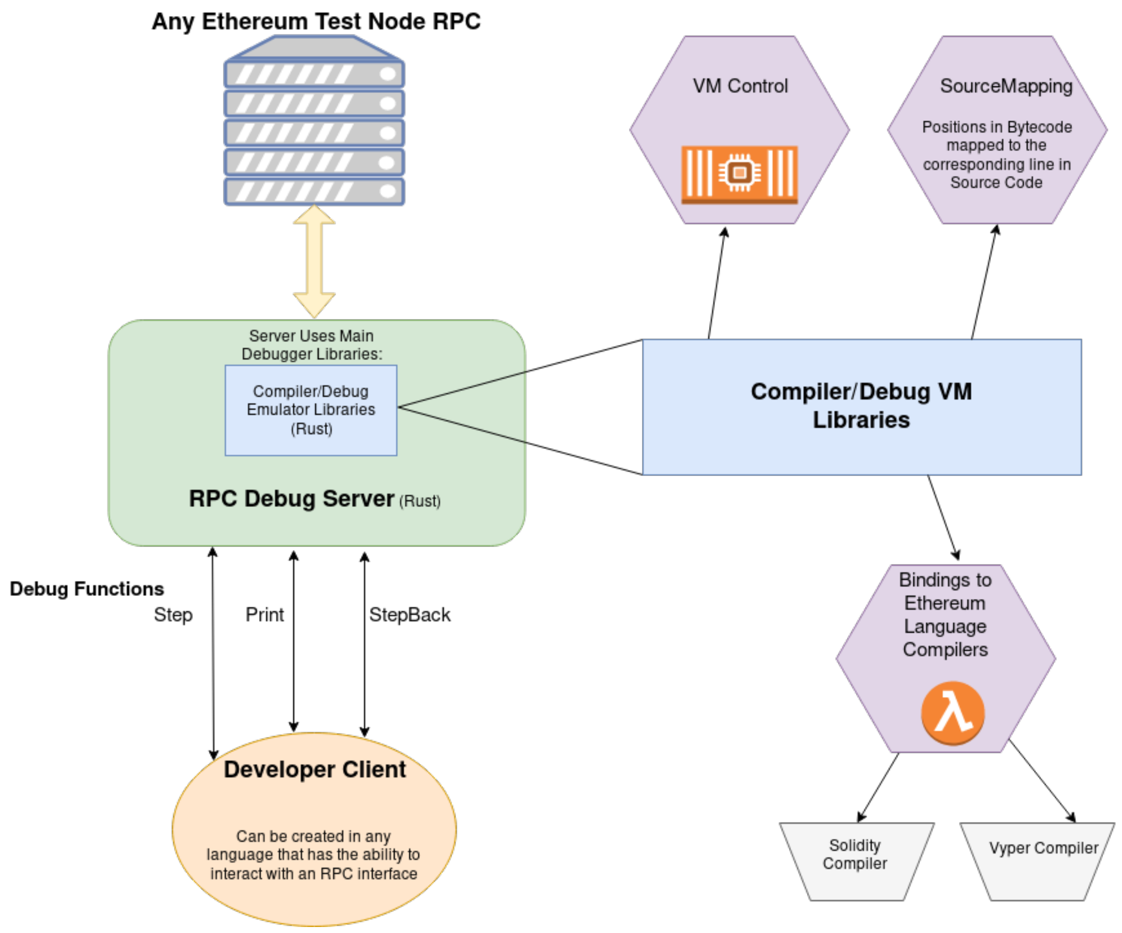
0.1 Introduction

Ethereum and Blockchain technology have the potential to produce an entirely new era of technology more disruptive and transformative than the Internet. This project works with the core of what Ethereum is, providing developers who work with the core Ethereum Blockchain a more intuitive experience. Ultimately, this means faster development and the proliferation of Decentralized Applications

Developers hoping to write new Decentralized Applications in Ethereum-specific programming languages such as Solidity, Vyper and LLL will be able to meticulously examine the state of their code at different points of execution taken by the Ethereum Virtual Machine. This includes being able to see the internal Ethereum Virtual Machine stack, and memory. In addition, the developer will be able to control the VM's execution; stopping it at a line in their code, continuing from a specific position, or stepping backward. Lastly, the developer may choose to inspect variables and their values that are in use in their code. All these functions will be available to be used with any Ethereum Node the developer may use for development. Additionally, The un-opinionated design of the libraries is such that it may be adapted to multiple different developer environments.

0.2 System Model

(figure on next page)


Any Ethereum Test Node RPC
VM Control
SourceMapping
Positions in Bytecode mapped to the corresponding line in Source Code
Server Uses Main Debugger Libraries:
Compiler/Debug Emulator Libraries (Rust)
RPC Debug Server (Rust)
Compiler/Debug VM Libraries
Debug Functions
Step
Print
StepBack
Bindings to Ethereum Language Compilers
Developer Client
Can be created in any language that has the ability to interact with an RPC interface
Solidity Compiler
Vyper Compiler

0.3 Functional Requirements

The developer will be able to conduct all debug functions from a client application. There may be multiple possible client applications, but they will all provide the same debug functions. Differences exist only in how they are presented to the user. For example, Visual Studio Code may have a button representing 'step forward', while a Command-Line Client would instead take the command 'step'.

A client will be able to enter into a debug session for a specific source file. The client will then provide some functions available to the user in order to debug the source file that has been selected. The user will see this basic functionality at their disposal:

- **help — ?**: Display a help message containing all available commands
- **version**: Display the current version of ‘edb‘ that is being used
- **step**: step forward to the next source-line in execution
- **stepback**: step back one source-line in execution
- **print**: print stack, memory, or variables in the program
- **printline**: print the current line in the source code that execution is at
- **printlines**: print a number of lines relative to the current line code execution is at in source code (two subcommands are available, each which accept a number argument. This would be used as such: ‘printlines previous 10‘ to print the 10 previous lines).
  - **previous**: print lines previous to current line
  - **next**: print lines after the current line
- **exit — quit**: exit the client
- **breakpoint**: set a breakpoint at a line number in the source code where execution will stop
- **restart**: restart execution entirely
- **run**: run a function in the source code (accepts arguments according to function parameters)
- **continue**: Continue execution to the next breakpoint
- **previous**: Take execution back to the previous breakpoint
- **stepinto**: Step into a function (if the current line contains a function call)

0.4 Non-Functional Requirements

In order to use EDB, the user will need to set up a local Ethereum Node that exposes an RPC interface and may be used in 'testnet' mode. Many variants of these applications exist, and the user is not constrained to a particular implementation. For example the user may choose between Parity, Geth, Cpp-ethereum, EthereumJ or EthereumJS. A node running in 'testnet mode' is not connected to the Ethereum Mainnet, and as such does not have the same storage requirements or require an internet connection. The user will need at least 1GB of storage for the test node.

In addition to this, the user is required to run at least Linux, MacOSX, or Windows. Only these operating systems are supported. Embedded versions of these operating systems are not supported. Depending on the language the user is planning to debug, they will need to posses the official compiler available for their system. For instance, debugging Solidity code requires that the 'solc' compiler is installed.

Code design and layout will follow official Rust Language guidelines[1]. In addition, Every public method, struct, and function in the resulting libraries from this project will be doc-commented to allow for document-generation. The project itself and all of it's libraries will be licensed under the GPLv3 GNU General Public License.

0.5 User Interface Specification

To initiate a debug session, the user will type the ‘edb‘ command into their terminal-emulator of choice. The ‘edb‘ command accepts one argument when debugging: the path to the source file to debug. For example, a user may type edb ./my_program.sol. The user is then greeted with the edb welcome screen and a command prompt. Typing in a command which is not recognized by the client will result in an error message denoting that the client does not recognize the command, and that the user may type the command 'help' or '?' to display a list of available commands. At any point in time, the user may use the 'exit' or 'quit' commands to close ‘edb‘.

A typical debug session consists of a user first using the ‘breakpoint‘ command to set breakpoints at locations in the source code. The user can then start program execution with the 'run' command. Since Ethereum does not have the concept of a 'main' function, the 'run' command accepts arguments pertaining to the function and parameters the user intends to run. EDB will execute the users program up until the first breakpoint it hits. At that point, the user may choose to 'step' to the next line of code, 'stepback' to the previous line, 'print' the current values of variables, the stack, and memory, or 'restart' execution entirely. In addition, the user has the option of printing the current line, or an amount of next or previous lines of source code with the 'printlines' command.

[1]Rust Style Guidelines *https://github.com/rust-lang-nursery/fmt-rfcs/blob/master/guide/guide.md*

Finally, the user may use the 'continue' command to continue execution to the next breakpoint they set, or the 'previous' command to go back to the last breakpoint they set.

0.6 System Evolution

The fundamentals of the project is written in Rust. While this is a slightly risky choice since Rust is a relatively new language, it has a promising community and is backed by Mozilla. Rust is used in a few well-known production applications; mainly the Firefox web-browser. Therefore, support for Rust will be available for the foreseeable future. Since the language is still new, however, maintenance will be needed for language constructs which are changing or being added, as well as refactoring done when older code is phased out and no longer idiomatic.

The next part of my project that will need to be maintained is the Ethereum Ethereum Virtual Machine and compilers. Ethereum is still a new application, so it's languages and protocols are never set-in-stone. Part of the maintenance of EDB will include keeping up-to-date with Ethereum Improvement Proposals and implementing them if any are accepted and effect the execution of Decentralized Applications. The libraries being used by EDB are open-source, and created by Parity Technologies. Parity is currently one of the main and most popular Ethereum Node Clients, so the maintenance of the libraries may be ensured for as long as Parity and Ethereum exist. It is, however, problematic if Parity Technologies happens to stop development of these libraries. In this worst-case scenario, since the libraries are under the GPLv3 GNU General Public License, the code may be forked and maintained as part of the debugger.

Glossary — Index

Blockchain A growing list of records stored using some datastructure (usually a variation of a tree) and linked through cryptography. 2

cpp-ethereum An Ethereum Node Client written in the C++ programming language. 5

Decentralized Application An application made specifically for Ethereum, which runs on the Ethereum Main Network. 2, 6

Ethereum Improvement Proposal Community-reviewed proposals anyone may submit to the Ethereum Foundation for review and discussion. If they are merged, they become part of the official Ethereum Protocol. 6

Ethereum Virtual Machine A stack-based virtual machine created specifically for executing Ethereum Bytecode. It is part of the main ethereum protocol and is present in any Ethereum Node. 2

EthereumJ An Ethereum Node Client written in the Java programming language. 5

EthereumJS An Ethereum Node Client written in the Javascript programming language, using the NodeJS runtime. 5

Geth An Ethereum Node Client written in the Go programming language. 5

GPLv3 GNU General Public License The GNU General Public License is a widely used free software license, which guarantees end users the freedom to run, study, share and modify the software. 5, 6

LLL A Lisp-like programming language that compiles down to EVM Bytecode. 2

Parity An Ethereum Node Client written in Rust. Many of EDB's libraries rely on libraries that were created by the Parity team. 5, 6

RPC 'Remote Procedure Call' is a protocol specification that outlines how a server should react to data sent to it. 5

Rust a systems programming language with performance comparable to C/C++, but which prevents segfaults, and guarantees thread safety.. 5, 6

Solidity A Javascript-like Programming Language that compiles down to EVM Bytecode. 2, 5

Visual Studio Code A popular code editor created by Microsoft. 4

Vyper A Python-like programming language that compiles down to EVM Bytecode. 2