A Software Strategy for Flexibility and Reliability

Kiran Shila

September 13, 2022

1 Author Biography

Kiran Shila is a Ph.D. in Electrical Engineering at Caltech, under Sandy Weinreb's and Gregg Hallinan's supervision. His educational background includes RF circuit design, phased array and mm-wave antennas, and computer engineering. Kiran graduated with B.S. and M.S. degrees in electrical engineering from the University of South Florida in Tampa, where he received the King O'Neal Award, Outstanding EE Graduate, and the Rudy Henning Award for Excellence in Microwave Studies. Before Caltech, Kiran worked at the NASA Goddard Spaceflight Center on mm-wave radiometer systems for earth science missions and the MITRE Corporation on beyond-line-of-sight communication systems.

Outside of studies, Kiran enjoys playing percussion with the Caltech Jazz Improv group and the Caltech Wind and Symphonic Orchestras. You will also find Kiran biking around the hills in Pasadena or at home, contributing to open-source software.

2 Abstract

The Galactic Radio Explorer (GReX) Telescope is fundamentally a software instrument. The capability to detect radio transients at such high resolution is not due to novel analog hardware or FPGA code but rather due to the implementation of a streamlined data collection and processing pipeline. Additionally, GReX is unique in that we plan to deploy stations worldwide, requiring substantial forethought in software design and deployment. Instead of the "traditional strategy" of throwing together scripts and collections of virtual environments, GReX utilizes a fully deterministic build and deployment system coupled with formatted, documented, and linted code. We strategized this to maximize uptime, allow easy customization and configuration, and quickly onboard those who want to contribute to the code base.

This talk will consist of two parts. First, we will discuss the role of the Guix package manager in the GReX system. Guix is GNU's deterministic build system in which we create package definitions that fully describe the entire dependency graph of a given application or library. Once defined, one can recreate these packages bit-for-bit. In the context of GReX, we package all pipeline software, associated scripts, and transitive dependencies in Guix. We also package the server software within Guix, where the kernel, networking configuration, etc., is just another package. This strategy allows us to have minimal configuration files and collect the full description of the software system into a single source of reproducible truth.

The second portion of the talk will cover the decision to use the Rust programming language for large chunks of the software. The Rust language is a modern systems language designed for correctness. As the GReX systems will be primarily remote, we want to develop software with compile-time guarantees of certain classes of runtime behavior. In addition, we want confidence that once compiled, our software will not crash or, at the very least, does so in a predictable manner. We will discuss the language features in Rust that make this possible and how the current software fits into the greater pipeline context.