Software Operating Environment for Phase 4 Ground

The document presents requirements and best practices for complex amateur radio projects, and suggests a platform choice for Phase 4 Ground.

What do projects look like now and in the future?

Amateur Radio projects are increasingly complex, and consist of an engineered system of components which operate together to achieve the end result. Components may be built from hardware, software, or both. As more functionality moves into software and as those functions become more modular, the nature of these projects must take on more of the principles of Systems Engineering, and address the complexity inherent in their interdisciplinary nature. Examples of some complex amateur radio projects are HPSDR, AMSAT Phase 4 ground station, the Algoram and Whitebox.

What are the characteristics and requirements of projects like these?

Different components are often developed by different teams, working in relative isolation from each other, and globally distributed.

An incomplete list of modular components for Phase 4 ground station might include:

- Radio transmit/receive chain control
- Antenna pointing control
- Contact (QSO) logging
- GPS interface, location and time service
- Authentication services
- Log uploading (when attached to the internet)
- Codecs
- Remote operation interface (when attached to the internet)

Components may not be delivered on time or in order, so it becomes necessary to design components to documented interfaces in order enable independent testing. As components are finished and tested, they are integrated into the product.

What do the interfaces look like?

Current best practices from cloud and web application computing are described below. These practices are supported by a wide range of tools.

For hardware which is accessed from a software application, the interfaces may be USB, I2C, Ethernet, or other standard interfaces, with documented APIs. Different hardware platforms may need to address these individually using drivers.

For software-only components, interfaces should be RESTful, network socket-based connections using documented APIs wherever possible. HTTP and JSON are almost universally supported. For interfaces or data streams that cannot be handled through a RESTful API, appropriate standards should be used. An example is using the VITA49 standard for radio IF data.

When possible, designs should strive to use a web browser as the client through which user interaction takes place, using standards such as HTML5.

Using this modular approach allows developers of each component to use the most appropriate tools to accomplish the task. For example, let's consider a software codec. It could be implemented in C++ or another language, or may already have an implementation available. It may also have a dependency on certain hardware which must exist in order to run, perhaps a separate Digital Signal Processor that's only available on evaluation hardware from the processor vendor. There may be multiple sets of code for the implementation - some which executes on the embedded DSP, and some which runs on the host. By making all interfaces conformant with documented network APIs, the team developing the codec can manage this complexity and use any development hardware and tools they require within their component project, and still have the codec component be available for use by the other components in development for the ground station.

This approach also divides the design discussion into more appropriate subsets of the entire project. Decisions about which APIs to use should be made at a higher level and applied across the project. Decisions about which is the best language for a particular component, by contrast, can be made at lower levels, closer to the needs of the implementation.

Further, it leads to a more extensible approach which enables addition of future features like additional codecs or modes. If we define a means for these components to advertise their availability to other components, we encourage experimentation and extension of the functionality of the ground station.

What do components (apps) developed under this paradigm look like?

This is another area where best practices have emerged. Developers should view their component as a service, available to other components. While the “web app” model doesn’t precisely fit all of what we’ll need on the ground station, the “Twelve-Factor” methodology is very informative: http://12factor.net/

A lot of the reasoning behind this is too long to include here, but factors like “Execute the app as one or more stateless processes” enable restarting crashed components without having to restart the entire device. Good design with a microservice architecture can go a long way toward being bulletproof.

What should the platform for the entire project look like?

Looking at this from the level of a “platform integrator,” we can start defining some system-level requirements.

Security and reliability of the final product is a primary concern. Isolation of components from one another and from the underlying platform is absolutely necessary. In order to assure platform performance, it should be possible to impose limits and quotas for system resources upon individual components. Applications should not have access to “secret” or local data for other applications, and should only be available to other components through documented interfaces. This is especially important since these systems use licensed radio spectrum. Bugs and security vulnerabilities are a regular occurance, so it must be possible to update individual components without updating them all. This also helps with integration and testing, since teams are able to update their component to the latest unreleased build while keeping the rest of the platform components at known-stable versions. Ideally, development teams could make different releases available for update subscriptions, such as ‘daily unstable’, and ‘released’.

Portability is important. Components are likely to be developed on a number of different platforms. It’s also likely that a unified “ground station box” will eventually be designed. It should be possible to run the components in as many environments as possible. Some components (like the QSO visualizer) could be cloud hosted and optionally available if the ground station is connected to the internet. Some components like codecs may need to run on special development hardware, accessible via network.

Ease of updates is important. This must support the security and development concerns above, but should also include multiple ways to distribute new releases of components and install them in the field, and to roll back component versions if needed. For Phase 4 ground, it should be possible to update components from the internet (if connected), using files contained on local media like USB sticks, or downloaded over a satellite link. Ultimately, it should be possible to auto-update the ground station components “over the air”. It should also be possible to release “master releases” for the entire ground station, with component versions which have been tested to work together. Having these known master version releases is necessary for reproducing and finding bugs, and supporting a community of users.

Here are those requirements in bullet form:

- Must be able to update single components and portions of the underlying platform.
- Must be able to update the entire platform to a released master version.
- Must be able to develop individual components using tools and application environment of choice
- Must support a variety of platforms
- Must support asynchronous updates of different components
- Must support multiple release channels for each component (unstable, released, etc)
- Must Isolate components from each other

I Propose a platform

This functionality is provided by modern operating systems and “containerized” operating environments. These have become ubiquitous, with companies like Heroku offering these operating environments for rent in the cloud (disclaimer: I work for Heroku). There are a large number of companies and open source projects competing for mind and market share in this sort of functionality, unifying both “bare metal” and virtual computers as underlying platforms.

Hardware capabilities have continued to increase in recent years, so it’s possible to purchase multicore ARM processors at very low cost. This allows us to consider using the best practices for cloud computing and process isolation for self-contained projects. In addition, storage is cheap and plentiful (using SD cards etc), so that it’s reasonable to bundle components with all the libraries and system components they depend on.

Specific to the Phase 4 project, the Ubuntu Linux platform could serve all the requirements listed above. The Snappy packaging introduced in the 16.04 release handles the isolation of components and updating them along with all their dependencies. Ubuntu is supported on a wide variety of platforms including ARM architectures. Ubuntu’s parent company Canonical (disclaimer: I previously worked for Canonical) has shown an interest in officially supporting new

platforms, and recently announced that Ubuntu and Snappy packaging will be used for LimeSDR (another great radio project). In particular, the LimeSDR project means that Ubuntu and Snappy will be used by developers for that project, which may well overlap with the set of Phase 4 ground station developers. Ubuntu has a thriving community which supports amateur radio applications on the platform, and a number of employees who are radio amateurs. They have provided a lot of help in the past with packaging and delivery of amateur radio applications, and could be very helpful to us as well.

I'm willing to coordinate and help define the platform for Phase 4 ground, and facilitate contact between phase 4 teams and developers, and the Ubuntu communities. I think it's desirable to establish an understanding of what the environment will be. This will help components develop to that from the beginning, and make integration much easier. It will also make the eventual design of a purpose-built hardware platform for the ground station much easier.

The first steps would be assembling documentation for best practices, with pointers to other documentation as needed.

Feedback?

How do other project members feel about establishing an "official" platform?

How do you feel about Ubuntu and Snappy Packaging as that platform?