Proposed 2022 Development Roadmap

Introduction

Discussion at AGL workshops and meetings during 2021 led to the proposal of dropping the existing demonstration application framework due to a combination of:

- a lack of member interest in assisting in maintenance or using it as the starting point for evolving a common platform in upstream AGL.
- its reliance on the SMACK Linux Security Module (LSM) framework, at least in the version upstreamed into AGL.
- recognition that its WebSocket plus custom JSON API scheme is somewhat limited versus the current state of the art available via other FOSS projects, and that future evolution into a cloud-friendly stack would benefit from using technologies known to that ecosystem.

While there was potential for approaches such as attempting migration to a new version of the application framework components from IoT.bzh's upstream or doing parallel development in AGL's version to remove/replace the usage of SMACK with another LSM, it was decided that complete removal would instead be the starting point of future development. Overall, starting from a clean slate seemed a better approach with respect to fostering reuse of external FOSS projects.

Current Status

As things stand, the Marlin 13.0 release will contain the following new development as an initial/interim replacement for the previous application framework:

- A minimal application launcher daemon that exposes a D-Bus API for application discovery and start/stop of applications. Application configuration is via .destkop files as with a session manager / launcher in a Linux desktop distribution. Application start up is done either by direct spawn or D-Bus activation (the latter is currently not used in the AGL demo platform).
- The launcher and homescreen Qt demo applications have been reworked to use the new application launcher D-Bus API instead of the previous app-framework-main one.
- The launcher and homescreen web app demo applications have been reworked to use a wrapping of the new application launcher D-Bus API inside a newly developed Web Application Framework (WAF) extension.
- The settings, homescreen, and mediaplayer Qt demo applications have had their usage of the agl-service-network, agl-service-bluetooth, and agl-service-mediaplayer bindings replaced with a shift of the ConnMan, BlueZ, and media playing API abstractions into the existing libqtappfw library via the use of two new of libraries (bluez-glib and connman-glib) refactored out of the agl-service-network and agl-service-bluetooth bindings.
- The other Qt demo applications that were feasible to re-enable building via stubbing out their usage of now removed bindings have been ported over and are included in builds to provide for a more comparable look to previous releases in the demo image.

Perceived Outstanding Requirements

Application Sandboxing

From EG discussions in 2021 it seems clear that some degree of enabling application sandboxing in a replacement demo application framework is desirable. One proposed scheme for this was attempting to adapt Flatpak into AGL, which has some appeal given it is achieving a degree of traction in the desktop Linux distribution space. However, investigation into this approach seems to indicate that the research and development effort required is perhaps beyond the level of commitment that AGL is able to invest in. Given that, leveraging systemd's sandboxing controls somewhat in the fashion of the previous application framework seems a lot more feasible, as does attempting some extension beyond that with some of the options available in current systemd. The additional benefit to such an approach would be potential synergy with Toyota's announced desire to work with upstream AGL on integrating a systemd based replacement for the resource/task management scheme in their base system for the Production Readiness EG.

Service API Infrastructure

With the application framework removal for Marlin 13.0, it was accepted that the APIs previously provided by the various demo service bindings would be reevaluated with respect to:

- what minimal set of services it makes sense for AGL to provide as technology demonstration and member demo enablement.
- what IPC mechanism it makes sense to use for the services AGL does develop, both in regards to effort and potential interest to members as a useful technology demonstration.

On the effort side, a lot of initial focus has been on attempting re-use of available FOSS services and their IPC mechanisms. In practice, that has led to using D-Bus for e.g. the new application launcher, and that has perhaps proven to be less than ideal for consumption in the web applications and with respect to potential for sandboxing application access. It seems clear that there are several forms of service such as audio mixer, radio, vehicle signaling, etc. that AGL likely needs to do some development on to enable member demos and serve as a potential starting point for member interest in upstream development. From discussion in the 2021 workshops and EG meetings into 2022, it seems that there is a rough consensus that grpc (grpc.io) seems a reasonable framework to use as a basis for such service development. A non-exhaustive list of the rationale for this is as follows:

- grpc as well as the protobuf tooling it is based on have a large and active development community.
- grpc tooling and programming language support is extensive, and well beyond AGL could hope to easily provide itself. For potential future Flutter app development, this includes a well-maintained Dart grpc library and protobuf compiler support.
- both grpc and protobufs are extensively used in cloud services, and there is potential for synergy with the Cloud EG with respect to their plans for service mesh enablement.

There are however, a couple of known potential drawbacks:

- Using grpc from web applications currently requires the use of a proxy mechanism as web engine HTTP/2 support is not yet complete enough to allow native implementation. In practice this means using the grpc-web library with either Envoy (envoyproxy.io) or Improbable's standalone proxy. Recent discussion in the EG indicates that this is perhaps acceptable in the interim, and that wrapping grpc would be likely be preferable to e.g. D-Bus if development on WAF API extensions ends up still being required. Additionally, the Cloud EG has indicated that Envoy is potentially something they would be interested in seeing available to ease service mesh demonstrations.
- Toyota have indicated that they considered using grpc for APIs in their own internal Flutter application development, but decided to move to using custom Flutter platform channel API wrappers (similar to what is done with WAF + web apps) for currently unquantified performance reasons. However, they agree that using grpc for AGL developed demo services seems a reasonable approach.

Replacing SMACK

The SMACK Linux Security Module (LSM) at this point is mostly only used by Samsung in their now mostly internal Tizen development. As mentioned above this was one of the motivations for removing the previous application framework. However, it seems reasonable to attempt demonstrating one of the more commonly used LSMs, e.g. SELinux or AppArmor. Multiple AGL member companies have indicated they use SELinux in their products, so investing effort into some form of enablement / technology demonstration in upstream AGL seems worthwhile if done in a modular fashion.

Proposed Development Tasks

Given the discussion above of perceived requirements from a technology demonstration and demo enabling perspective, the proposed 2022 development roadmap is as follows.

Application Launcher

Development tasks:

1. Add discovery and launching of AGL applications via the systemd D-Bus API. AGL applications would provide a systemd user unit applaunchd could discover via the systemd API and then stop/start in response to its own API mechanism. The goal would be to at first deprecate, and then obsolete (for e.g. the Octopus 15.0 release) the direct spawn and D-Bus activation schemes currently implemented. Potential standardization /templating of application systemd units and whether to still rely on the use of .desktop files is something that would require investigation and discussion in the EG. A lesser reason for using systemd units rather than D-Bus or direct launching is for an improvement in logging, a discussion on this can be found in the comments on SPEC-4211.
2. Add a grpc API that duplicates the current D-Bus one, with an eye towards potentially obsoleting the latter in a future release once grpc use is better understood. Rework of the demo homescreen and launchers to use the grpc API would likely be part of this effort unless a separate test application is developed.

Service API Infrastructure

Development tasks:

1. Develop a grpc API provider for the previous agl-service-audiomixer binding, to allow for both the Qt and web app demo mixer applications to be reworked against it and re-added to their demo images. This would act as a frontrunner for demonstrating using grpc/protobufs in AGL potentially in parallel with (2) above, though it may be more straightforward for one or the other to be chosen as an initial project to establish an example. The audio mixer binding is viewed as a good example service to reimplement in that the PipeWire and WirePlumber APIs it wraps will require non-trivial development to re-enable in demos no matter the approach taken.
2. In parallel with (1), start an investigation into the feasibility of enabling grpc-web usage in the demo web apps. This will involve evaluation of making Envoy or Improbable's proxy available inside of AGL with respect to build and configuration, and once that is in hand attempting a demonstration of using an API in a web application with grpc-web. It is possible (or perhaps likely) that this may need be decomposed into separate tasks for the build versus web app development components.
3. Once the audio mixer and/or application launcher APIs have been shown feasible, move on to reimplementing some of the other binding services that have no available FOSS API mechanisms that are easily leveraged, with likely priorities being services such as the radio and the HVAC bindings. It is likely that services such as telephony are good candidates here as well, since their APIs are relatively simple and development of some kind would be required to (re)integrate them into the demo applications in any instance. From a technology demonstration perspective, once there are some examples in hand there may be opportunity to drive member engagement on API requirements with respect to serving as useful abstraction layers for proprietary implementations. Potential rework of the Bluetooth, network, and media playing APIs should only be considered once it is clear that the grpc approach is workable and that follow on maintenance will not be a significant issue. Bluetooth is one area where there has previous been member interest in working on a reusable API, so that might make it a more worthwhile candidate for attempting later in 2022.
4. Investigate enabling and using the grpc API in the kuksa.val vehicle signaling framework. The planned use of kuksa.val in Marlin 13.0.x will be limited to the standard VISS WebSocket API due to build issues stemming from that project's use of CMake intersecting with a known upstream OpenEmbedded/Yocto Project limitation. Working with upstream of potentially both projects will be required to enable using grpc with kuksa.val, with a potential significant benefit to OE/YP/AGL ecosystem use of grpc in the future. One note with respect to this effort is that some care will need to be taken in any of AGL's own development around grpc to avoid use of the CMake grpc module until the issue with it can be resolved with upstream.
5. Outside of base grpc usage, authorization and service discovery are functionality that need further research and development to work towards a complete technology demonstration. It is likely that the two features need separate research tasks that would lead to potential development later in 2022 or in 2023 once the EG reaches some consensus on any proposed development stemming from the research. There is some potential on both fronts to work with the Cloud EG in requirements definition and development, as their micro-services and service mesh plans have overlaps in these areas. One point of concern that will likely need to factor into research in this area is that some of the available solutions in at least the service discovery side are somewhat heavyweight for an embedded/automotive system, this is likely another area where there is some good synergy with the Cloud EG plans, as it is also a concern for them.

SELinux

Development tasks:

1. Add the meta-selinux layer to the default AGL demo image builds, with an initial goal of getting builds working with the upstream targeted reference policy in permissive (i.e. non-enforcing / warning) mode. The aim would be to have the SELinux kernel configuration, tooling, and base reference policy available in the AGL infrastructure for members to leverage.
2. Start an investigation into the iterative effort to tweak the SELinux policy to allow running the AGL demo images with SELinux in enforcing mode. This should ideally involve with working with meta-selinux upstream, as currently this is not possible when using systemd in even a plain core-image-minimal Yocto image. It is not expected that this work would be completed in 2022, though there is some potential for being able to enable demos on telematics or cloud gateway demos. Demonstrating the ability to have SELinux work in enforcing mode in a non-IVI image with a container engine runtime may be an achievable goal for 2022, and has good synergy with Cloud EG requirements.