Creating Services for AGL

AGL Training Class
October 20, 2022
Scott Murray (scott.murray@konsulko.com)
About me

• Linux user/developer since 1994
• Embedded Linux developer since 2000
• Principal Software Engineer at Konsulko Group since 2014
• Working on AGL on contract since 2016
  • Yocto Project maintenance
  • Demo development, integration, and maintenance
Agenda

- AGL Services?
- AGL Services pre-Marlin
- AGL Services Today?
- protobufs and gRPC
- Implementing a AGL Service?
- Example: applaunchd
- Summary
- Future plans
What do we mean by AGL Services?

- Demo and/or example services in upstream AGL tree
  - e.g. for HVAC, radio, media playback, navigation
- Used for AGL's own demonstration images
- Goal of serving as an example of building such services on top of AGL
- Allow demonstrations with different front ends
  - Qt and HTML5, now Flutter
AGL Services pre-Marlin

• Used now legacy application framework
• APIs implemented with JSON over WebSockets
• Linux SMACK Mandatory Access Control (MAC) used similarly to Tizen
• Framework included packaging and installation
• Services built against this AGL specific framework
  • Effectively tied a lot of code to the framework
AGL Services Today?

• The legacy application framework did not gain traction with members, and it became difficult to justify the maintenance effort
• As well, the technology choices for it became less interesting as a forward looking technology demonstrator
  • SMACK, JSON over WebSockets
• Discussion started in 2021 about a replacement
Leveraging existing FOSS

• Proposal from Collabora to replace the application framework by leveraging widely used open-source projects as much as possible
• Aim of providing a more relevant technology demonstration with lower maintenance effort
• Some AGL demonstration services would be reimplemented, but that would be avoided if a suitable FOSS replacement was available
• Collabora proposal suggested using protobufs and gRPC as basis for new APIs
protobufs

- protobufs = protocol buffers
  - [https://developers.google.com/protocol-buffers](https://developers.google.com/protocol-buffers)
- language-neutral, platform-neutral extensible mechanism for serializing structured data
- Simple data definition language with code generation for read/write of binary serialized data
  - Support C++, Java, Rust, Dart, etc.
- Google project with a large userbase
- Widely used in cloud infrastructure
gRPC

• gRPC is a modern open source high performance Remote Procedure Call (RPC) framework
  • [https://grpc.io/](https://grpc.io/)
• RPC API specification is an extension of the protobufs definition language
• Another Google project
• Like protobufs, large userbase and widely used in cloud infrastructure
Vehicle Signaling

- The legacy application framework included an API for CAN signals and a "signal-composer" API for abstracting signal sources for applications
- Replacement for these using existing FOSS projects?
- Investigation in 2021 found emerging Vehicle Signal Specification (VSS) and Vehicle Information Service (VIS) Server standards
- Decision to adopt KUKSA.val VIS server
  - Extends VIS with a gRPC version of the API
  - Further discussion in "Using CAN Services with AGL" next
AGL Services Today…

• applaunchd
  • gRPC API for application start/stop/status

• agl-service-audiomixer
  • Backend for VSS master volume signal
  • Addition of a gRPC version of the API from the legacy application framework planned before CES 2023

• agl-service-hvac
  • Backend for VSS HVAC signals
Implementing a AGL Service?
Implementing a AGL Service?

• If the API is something not covered by VSS
  • Define API with gRPC
  • Use that to build service daemon

• Otherwise
  • Build service daemon that implements API from VSS
  • Example will be shown in "Using CAN Services with AGL"
Implementing a gRPC API Service

1. Define API
2. Generate API stubs
3. Build implementation on top of stubs
Defining gRPC API

• RPC methods defined in .proto file: https://grpc.io/docs/what-is-grpc/core-concepts/#service-definition
• There are naming conventions: https://cloud.google.com/apis/design/naming_convention
• And a style guide: https://developers.google.com/protocol-buffers/docs/style
Defining gRPC API (continued)

• Keep compatibility concerns in mind
  • Adding message fields or RPC calls is okay, removal should be avoided without a clear deprecation plan
  • Be consistent with message field tags, and avoid changing them

• More information:
  • https://earthly.dev/blog/backward-and-forward-compatibility/
  • https://www.beautifululcode.co/blog/88-backward-and-forward-compatibility-protobuf-versioning-serialization
Generating API Stubs

• Manually with "protoc" protobufs compiler
  • Example at https://grpc.io/docs/languages/cpp/basics/#generating-client-and-server-code
• Preferably with meson or CMake rules
  • meson easier and greatly preferred for any new AGL development
  • Example at: https://git.automotivelinux.org/src/applaunchd/tree/src/meson.build?h=needlefish#n36
API Implementation

• gRPC has synchronous, asynchronous, and callback server and client APIs in the C++ implementation
  • Synchronous API simple but blocking unless manual thread processing is used
  • Asynchronous API more complicated, but more flexible, and handling some error cases is more straightforward
  • Newer callback API seems likely to replace the existing asynchronous API over time
    • Should be considered for new development
Example: applaunchd
applaunchd?

- Qt based demo homescreen and launcher start external applications
  - e.g. mediaplayer, navigation, etc.
- Had been using API provided by af-main binding in the legacy application framework
- A replacement was required -> applaunchd
- [https://git.automotivelnux.org/src/applaunchd/](https://git.automotivelnux.org/src/applaunchd/)
applaunchd (Marlin)

- Initial prototype implementation
- D-Bus activated daemon
- D-Bus API
- Applications enumerated via .desktop files
- Applications directly spawned by daemon
applaunchd (Needlefish)

- Daemon substantially reworked
- Applications started with systemd template units
  - Sandboxing configuration examples via optional systemd override units
- Application enumeration based on systemd unit presence
  - agl-app*@*.service pattern matching
- gRPC API
applaunchd API

applauncher.proto - RPC definition:

```plaintext
service AppLauncher {
  rpc StartApplication(StartRequest) returns (StartResponse) {} 
  rpc ListApplications(ListRequest) returns (ListResponse) {} 
  rpc GetStatusEvents(StatusRequest) returns (stream StatusResponse) {} 
}
```
applaunchd API (continued)

applauncher.proto - example messages:

```protobuf
message StartRequest {
  string id = 1;
}

message StartResponse {
  bool status = 1;
  string message = 2;
}
```
applaunchd gRPC Implementation

- .proto file -> generated stubs
  - meson.build rules for generation
- Uses gRPC synchronous server API on top of generated stubs to implement service
- Synchronous server API used in applaunchd for now
  - Seems sufficient for low volume of API calls
  - Simplicity of implementation
  - Plan to reimplement with the callback API in the future as an improved demo
applaunchd Source Walkthrough
Future Development
Plans for 2023

• Finish minimal set of services for demos
  • Audio mixer
  • Radio
  • Network configuration
  • Bluetooth configuration
  • Others?
• Switch to using gRPC API in KUKSA.val
• Set up a global repo for AGL API .proto files
  • Single source for server and client implementations
• Implement a demonstration of service authorization
  • systemd-creds, OAuth, ?