Common Vehicle Interfaces Working Session IFEX to/from uServices

COVESA

Accelerating the future of connected vehicles

Interface Exchange framework (IFEX)

- Yet another Interface Description Language (IDL) ?
- A generic model for *all* Interface Descriptions !
- A set of tools to convert between existing IDL formats !
- A more comprehensive view of interfaces through layers !
- A stronger Error-handling description than before !
- Bridge between “standard” software technologies (non-automotive) and significant *automotive specific technologies (*AUTOSAR)

- ALL OF THE ABOVE

Background

Too many interface description languages and IPC/RPC technologies!

Create another “one to rule them all”? (yes it is ironic… "XKCD standards")

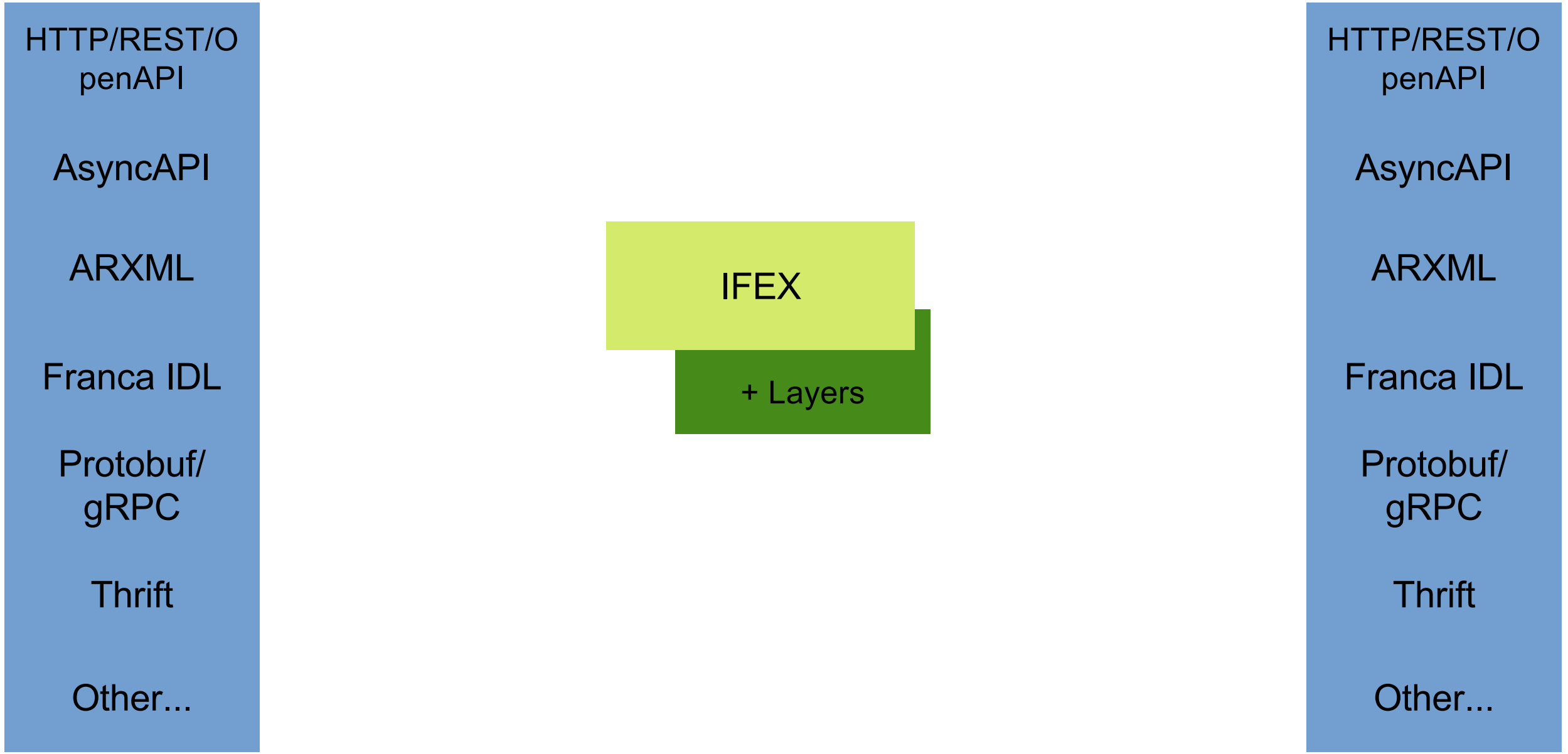
STOP! The main point is not should we create, or avoid, yet another Automotive-specific technology.

The IDL is not the (main) point - it is **determining the semantic equivalences and differences between existing technologies.**

→ To efficiently connect them, and flexibly swap one for another.

Conversions: N-to-N

Conversions: N-1-N



Why do IFEX?

The IDL is not the point - it is **determining the semantic equivalences and differences between existing technologies.**

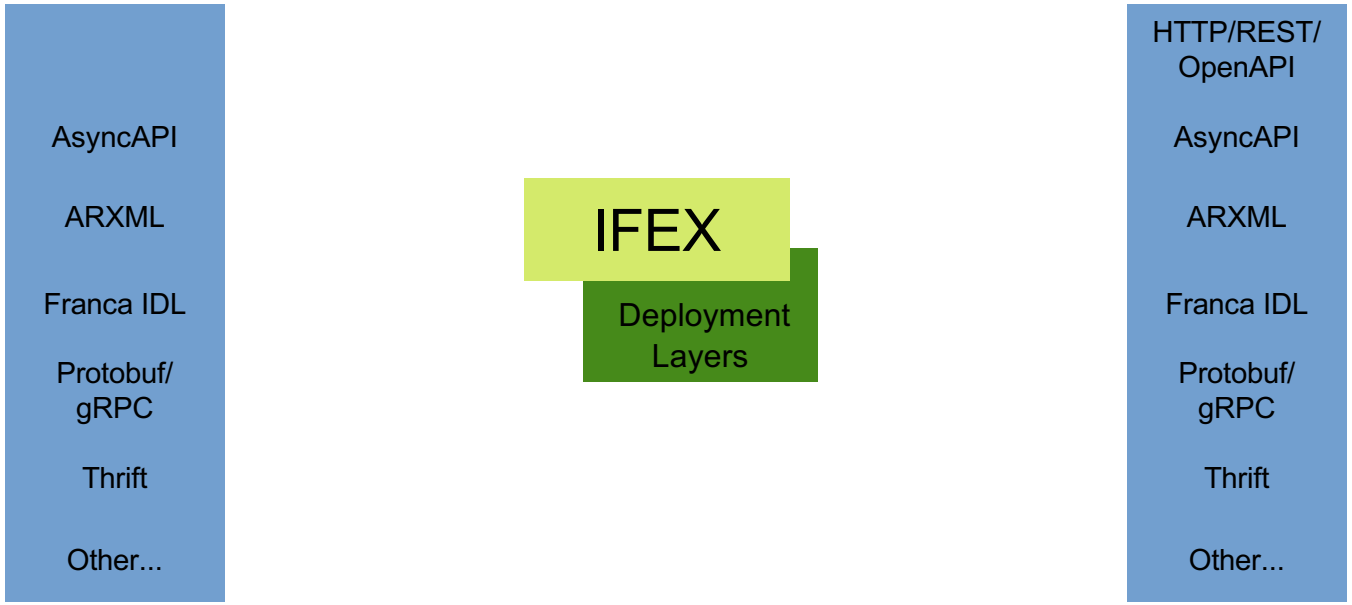
→ To efficiently connect them, and flexibly swap one for another.

The IFEX Project is a place to do the challenging *semantic*-mapping work

* While doing so, it creates translation tools between formats

* … and it results in a simple but powerful interface description format
(because it is forced to include “all” features of the other alternatives)
(more importantly because it uses Layers, to separate individual concerns)

Refined view



Not just an IDL.

- A common interface-description “***model.***
- A project to investigate, connect and unify interface & communication technologies.

A generic interface model (aka IFEX core IDL)

```
namespaces:
  name: ...
  version: ...

  typedefs:
    ...

  enumerations:
    ...

  structs:
    ...

  methods:
    ...

  properties:
    ...
  events:
```

Namespaces
- Can be nested
- Major & minor versions enables versioned APIs

Typedefs
- Type defines native, defined, struct, or enumeration types
- Supports array definitions

Enumerations
- Supports optional value specification for each element

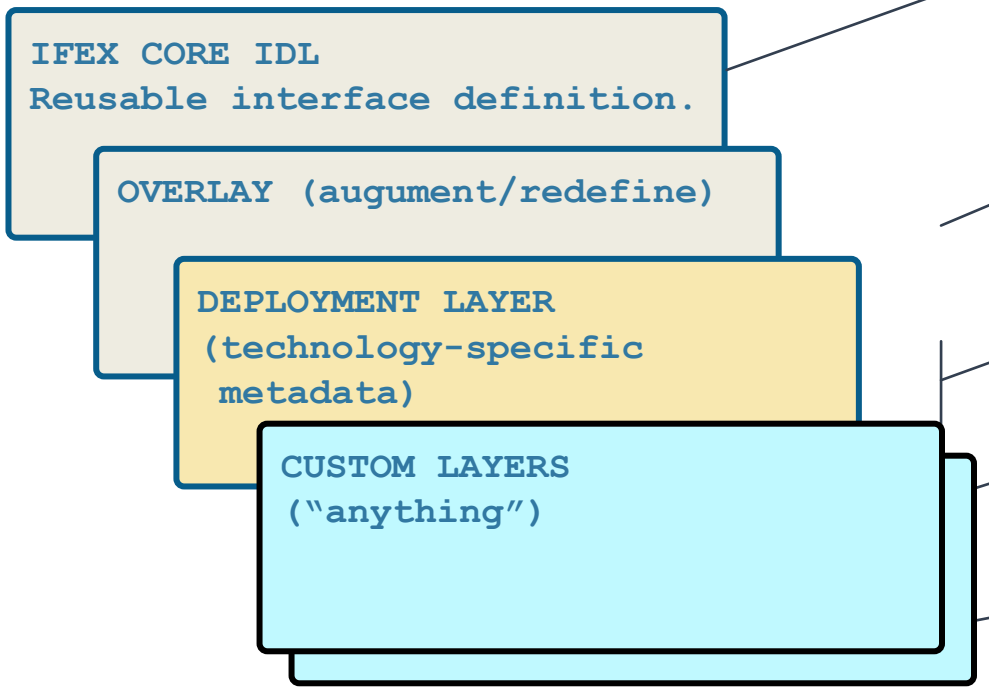
Structs
- Can be nested
- Elements can be of any native or defined datatype

Methods
- Arbitrary number of input and output parameters
- Can return stream of output parameters
- Comprehensive Error definitions (composable)

Properties
- Observable data item. Get/Set/Publish/Subscribe

Events
- Events can contain arbitrary number of elements

Composable Layer Philosophy



IFEX Core IDL
- Defines names, types and generalized behavior
- "Pure interface" (functional view)
  – no technical specifics
- Reusable definition across all technologies

Overlays
- Optionally augument/redefine of an existing interface def.
- E.g. COVESA standard interface + my own small change
- Protocol-specific errors layered on top of business-logic errors.

Deployment Layer
- Required details to deploy the general interface with a specific *target environment*.
- *Target environment* = programming language, communication protocol, deployment environment, etc.

Custom Layers
- Defined by community ("standard") or local (proprietary)
- Anything:
  - Access control logic/rules
  - Interface sensitivity (privacy / personal data?)
  - Safety/Security specific handling?
  - Classification according to OEM-internal nomenclature
  - … whatever is needed

Details and F-A-Q

Q: Why not just select an existing IDL and put that in the middle of N-1-N?

A1: None of them have ALL the features of the others

A2: Few care about overall picture, compatibility with other choices

A3: None (*_except Franca_) have a strongly layered approach required to manage complexity and IDL scope-creep.

- IFEX adopts this important concept and extends it
- Avoid deployment details and related meta-data to pollute the core IDL. Put those details in composable layers
- => keeps the fundamental “interface-description” reusable

Details and F-A-Q

Q: Isn't it a lot of work to create code generators for the IFEX IDL?

A: Some work, yes. But we only write new what is necessary!

A: Reuse: Translating to an existing IDL means we can often use "*their*" code generators. IFEX source → <IDL A> → [reuse existing tools for IDL A]!
In some areas, it is more a *requirement* (approved AUTOSAR tools need ARXML)

Status

Core IDL/model specification “v 1.0” complete (mid-2023)

→ Mostly stable. Minor updates expected from now on (versioned!)

Implementations of several to/from technologies

Layer Type definition is continuous, as support for translations grow

Status (2)

Implementations and principles for IFEX tooling exists.

Python implementations – lightweight and easy to get into.

New tools can be developed following the existing patterns.

Existing support:

Translation into formats like DTDL, SDS-BAMM, Protobuf (gRPC), AUTOSAR XML* (early stage) exists *not published yet

IFEX -> D-Bus -> C++ code generation chain

Translation gRPC -> IFEX implemented (a few features pending)
Analysis of OpenAPI
Others, prioritized on a need basis.

Find out more

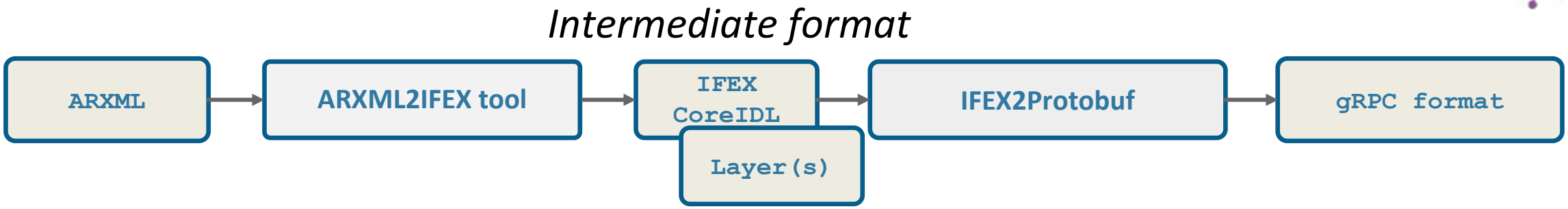
This presentation does *not* cover many details about IFEX

The project has been active for a few years so many of *your concerns* are known – but please ask and we will clarify/discuss

Read the specification of the IFEX Core IDL

Ask IFEX developers for deeper discussions

Converting to change technology
Converting to, or via, IFEX



- IFEX Project is where translations are analyzed/defined
- IFEX IDL/abstract-model can be used as an intermediate model

- Converting to IFEX (one single “rule them all” interface description format)
- **OR**:
  Converting *via* IFEX, to what you need

Reuse the ecosystems that exist (code generation)

- Input could be something other than IFEX -> converting *via* IFEX
- Output could be any other supported format
- Leverage a huge amount of existing technologies
- **WRITE** “simple” translations between IDLs
  REUSE “complex” output/code-generation/etc.
- Writing custom generators for IFEX only if and where it provides new value

Reuse the ecosystems that exist (code generation)

- Input could be something other than IFEX -> converting *via* IFEX
- Output could be any other supported format
- Leverage a huge amount of existing technologies
- **WRITE** “simple” translations between IDLs
  REUSE “complex” output/code-generation/etc.
- Writing custom generators for IFEX only if and where it provides new value

uServices – Example of standard interface

uServices = Open-source interface proposals for common features
Described and published by General Motors.

Original described using Protobuf language (gRPC)

Investigation: Convert to IFEX for proof of concept and analysis

IFEX features and approach *might* be a better way* to describe the fundamental interface (discussion)

*Reasons
- IFEX Core IDL is a richer language.
  - Methods take multiple arguments instead of one single protobuf message (struct))
  - Stronger Error description capabilities
  - Layers with clearly named additional metadata instead of overloading protobuf “option” feature)
  - Layering generally gives more extension capability

Ref: Example Layer definitions for E2E, etc. -> LINK

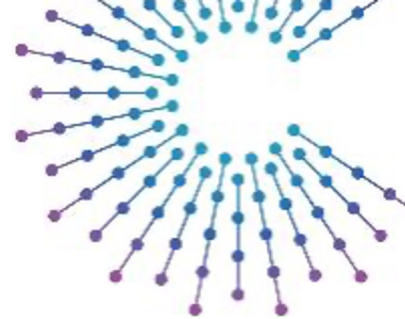
uServices in IFEX

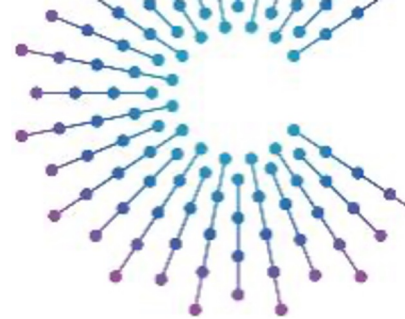
Better explained using live Demo
… and viewing/discussing the result

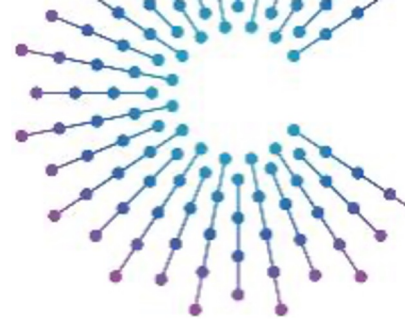
Challenges and discussion areas:

- What are the protobuf **options** used for?
  What is the underlying semantic “feature” we actually strive to describe with it?

- What are protobuf features such as “extends” or “reserved” good for in a RPC-style interface description?
  I think they are a results of protobuf’s original purpose: to describe an extensible data serialization format. Do they serve a purpose in an RPC-style interface description?
  Should IFEX (using a Layer) include similar features?







COVESA

Accelerating the future of connected vehicles