GDP principles

• Right combination of long-term architecture thinking and **standard** open-source solutions
• APIs formally described in Franca IDL
  -> Enables to **flexibly** generate bindings to **fast changing** technologies
GDP – Next

Increase leverage of GENIVI-unique features, while improving code reuse
GDP – Next

A system for engineers looking for a development platform for fast innovation
GDP – Next

1. Leverage unique features
2. Continue investigation into model-based / formal system definition and code generation
3. Reuse fitting AGL code
4. Improve standard-Linux technology adoption
GENIVI – AGL reuse

• Some things don’t fit. Some do.
• Alignment of BSP and hardware layers to close perceived hardware support gap (std. Yocto BSP)
• Initiate shared functional Yocto layer
• Reuse of appropriately fitting code/recipes to improve platform
• Strengthen alignment through 3rd party collaborations (W3C, OCF, ...)
GENIVI – Standard Linux Reuse

• GDP Master project to use Flatpak
  – Application packaging and containment

Good example of increased Standard Linux Reuse
GENIVI – Standard Linux Reuse

• GDP Master project to use Flatpak
  – Application packaging and containment

Good example of increased Standard Linux Reuse
It’s all about APIs!

• GENIVI principle is formal API description of all Inter-Process Communication using 2 main ways:

1. **Franca IDL** for all command-and-control (i.e. RPC)
2. **Signal model** for all vehicle and internal signals/events (Potential alignment/convergence possible)

Generative approach produces tremendous flexibility.
Messaging

- Formalize a decoupled IoT-style embedded message router (**WAMP**, MQTT/other) – under discussion
- Web API, inter-node, and device connectivity all in one?*
- Bridges (= translate GENIVI API descriptions to/from any other desired technology, for example **REST**ful APIs)
- * **SOME-IP** binding remains option for inter-node comms.
GDP - Next

• Everything and Anything is Securely Connected (e.g. RVI)
• Fine grained least-possible-privilege security model for every component (not only apps)
• Enabling Component and System model-based development --> generates security configuration, among other things.
GDP – Next-Next, ideas...

• GDP Core
  (compact, headless, cluster, telematics, AD, and more)

• Third generation SOTA/update/code-integrity solution using content addressable filesystems, dm-verity, ...
GDP – Next-Next, ideas… (2)

• Component and System modeling yields results:
  • Generated full fledged fine-grained security implementation?
  • Enable experiments in totally new system approaches
    - Full function deployment transparency across networks
    - Massive Virtualization/containerization (CoreOS\textsuperscript{(R)} or similar).
    - Anomaly detection systems on component level