OSGi: The Best Tool in Your Embedded Systems Toolbox

OSGi DevCon Europe 2009

Brett Hackleman & James Branigan

www.bandxi.com

Introduction

- **OSGi has been an open standard for over a decade**
- **Original domain (home gateways) has yet to take off**
- **Instead, OSGi has gained significant traction as a well-designed software componentization and deployment framework**
  - *Eclipse, as the base platform*
  - *J2EE server providers (Server-side OSGi)*
  - *Telecommunications (Sprint Titan)*
  - *Embedded (MicroDoc & Band XI)*

Our Perspective: A Decade of Embedded OSGi

- **Automotive, Fleet Management, Remote Monitoring**
  - *OnStar scenarios, diagnostics, infotainment, hands-free, navigation*
- **Industrial Controls, RFID**
  - *Sensors, scaling, configuration management, deployment*
- **National Guard: WMD Sensors, Situational Awareness**
  - *Provisioning new apps, biometric sensors, P2P comms*
- **Army: Vehicle Diagnostics & Prognostics Architecture**
  - *Provisioning new algorithms, flexible UI, provisioning*
- **Mining Equipment: Diagnostics, Machine Control**
  - *Rapid turnaround on new applications, flexible architecture*

Challenges of Embedded Systems

- **Platforms**
  - *Availability, variability, scarcity, stability, cost (future of Java?)*
- **Peripheral hardware: sensors and actuators**
  - *Availability, scarcity, size, cost*
- **User interfaces**
  - *Input methods, desktop vs. appliance*
- **Constraints**
  - *"Realtime", memory/CPU limitations, development vs runtime costs*
- **Applications**
  - *Yes, you still have to write the application(s) too!*

Accepting the Challenge

- **Lots of Risk == More Fun!**
- **Tackle risks head-on**
  - *Build simulators, identify alternate paths and representative systems*
- **Embrace change**
  - *Agile development methodologies build change into the process*
- **Use agile methods, frameworks, and tools**
  - *Testing frameworks (JUnit, FitNesse)*
  - *Continuous integration build systems (Jazz)*
  - *Software Configuration Management (Jazz)*
  - *And, of course…*

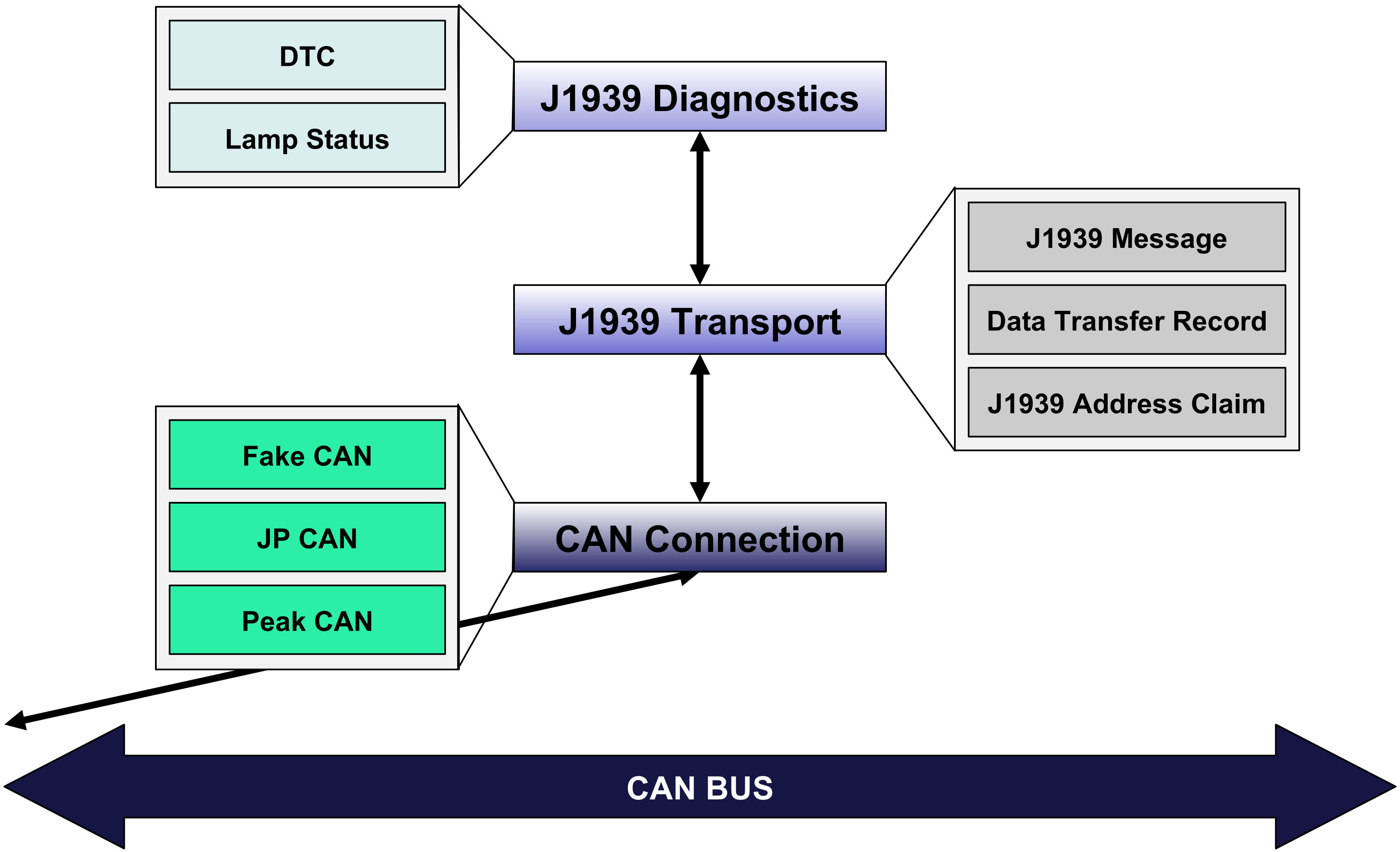
OSGi – Highly Cohesive, Loosely Coupled Component Model

- Strong component model, common language
- Forces best practices from day one
- Core specification is pure and simple
- Logical and clear mapping with whiteboard drawings
- Dynamic life-cycle allows install/unintall/update without restart
- Integrated and sophisticated deployment mechanism
- Isolation of external libraries to monitor and consume capabilities
- Discourages spaghetti code: bundle partitions are like firebreaks
- Discourages code rot: containment lowers barrier to refactoring

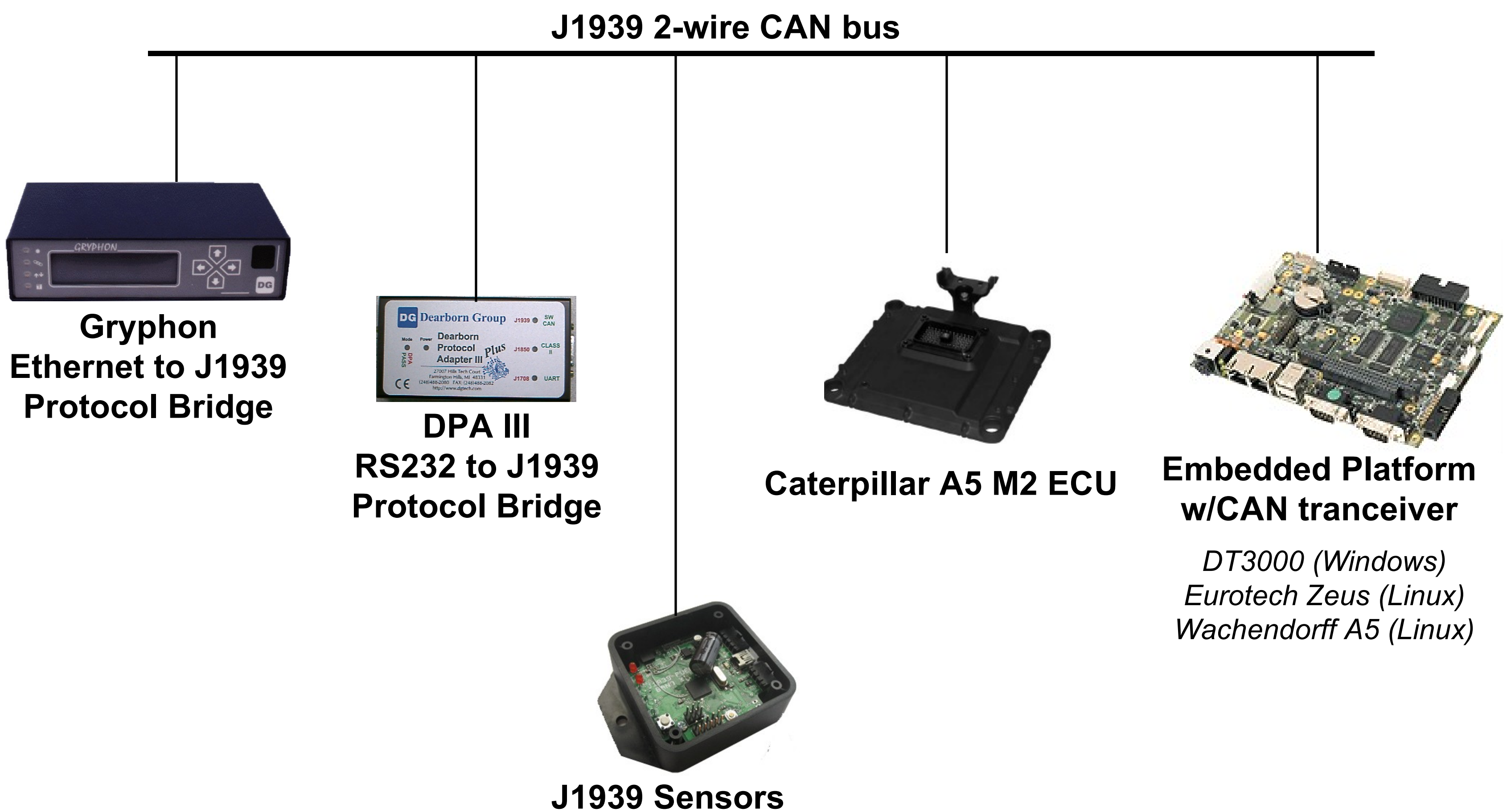
Vehicle Bus Control: DeviceKit Support of J1939 Protocols

- **J1939-21: Datalink**
  - *Basic Protocol Data Unit (PDU), PGNs, priorities, etc.*
  - *Transport Protocol (multi-packet, broadcast and point to point)*
- **J1939-81: Network Management**
  - *Address Claim (fixed, arbitrary address)*
  - *Discovery*
- **J1939-73: Diagnostic Messages**
  - *DM1: Active Diagnostic Trouble Codes (DTCs)*
  - *DM2: Logged DTCs*
  - *DM14/15/16: Memory Configuration*
  - *Additional information for active/logged DTCs*
- **Proprietary J1939 Messages**
  - *For solution-specific ECU's*

In Vehicle J1939 Device Software Stack



J1939 (CAN) Test Bench



J1939 Interfaces Supported

- **Dearborn Group Gryphon (Ethernet to CAN)**

- **Peak CAN (USB to CAN)**

- **Wachendorff on-board PCAN transceiver**

- **Eurotech Zeus on-board CAN transceiver**

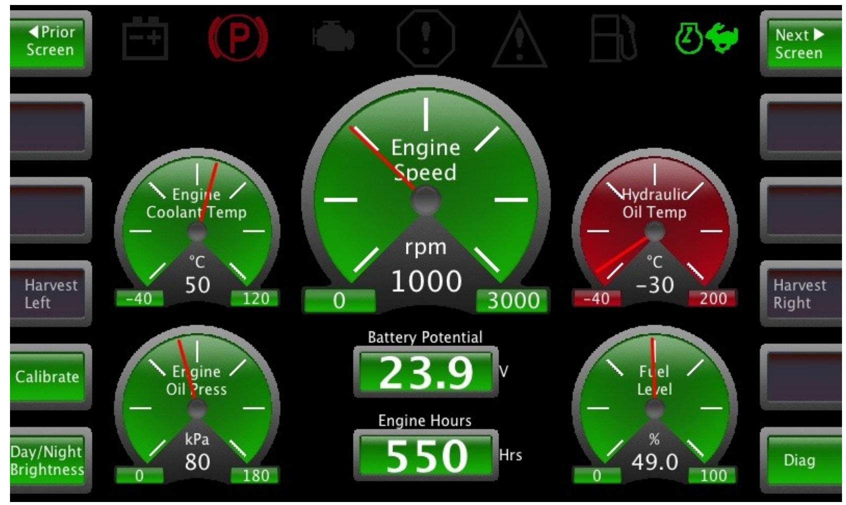
- **DriverTech DT-3000A**

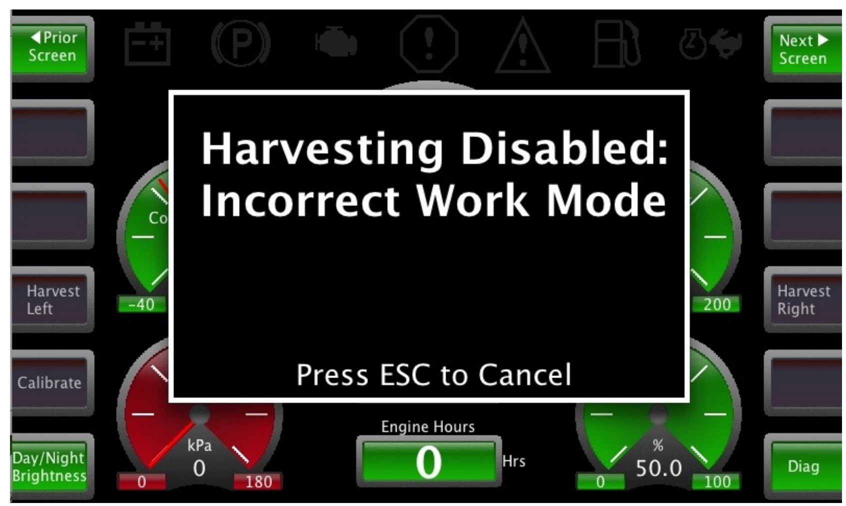
- **DRS TEM BlueRing Platform**

- **RP1210A library (CAN/J1939 on Windows, Cat CMPD)**

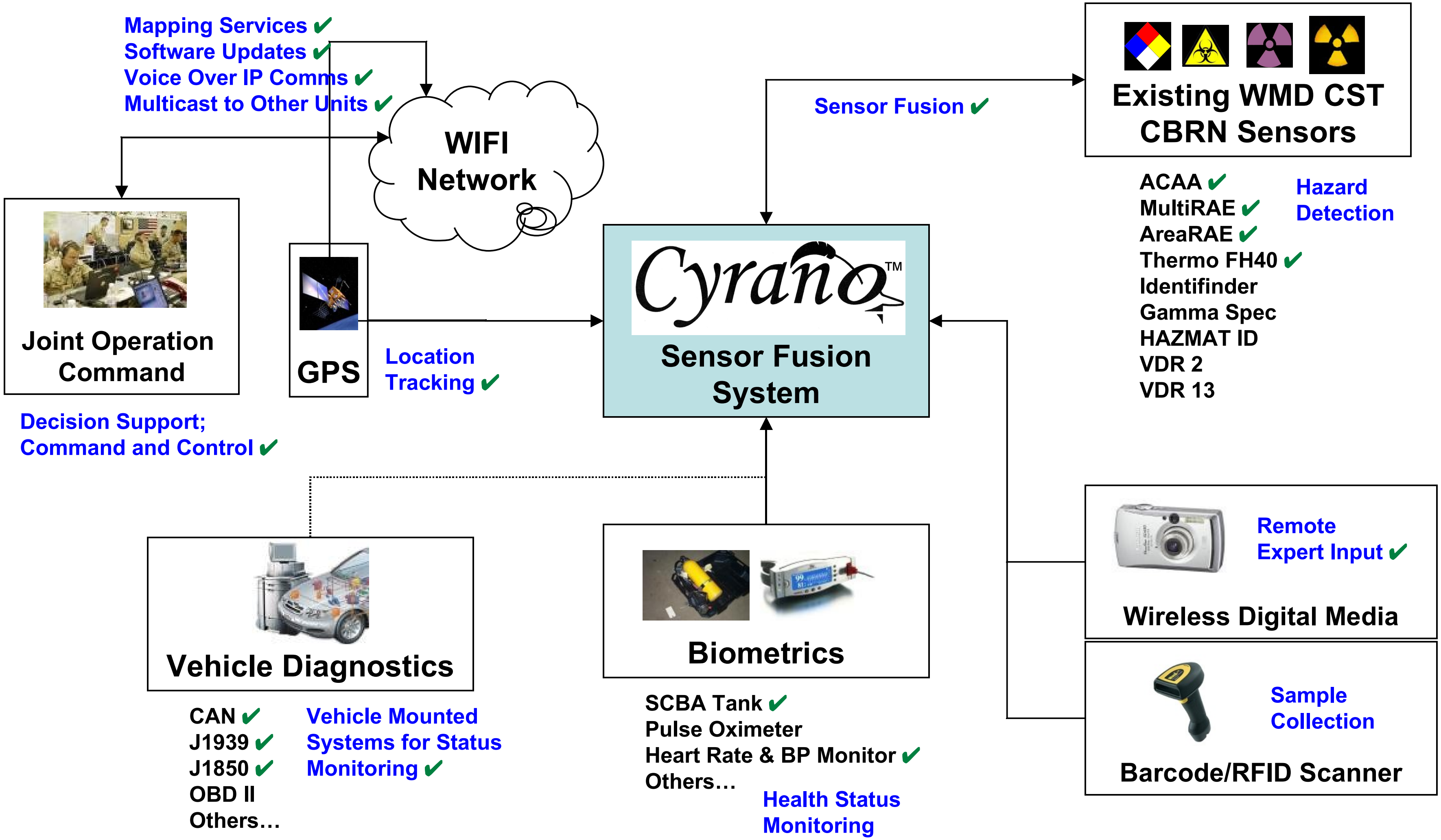
Industrial Graphics Framework

- Desktop metaphors do not work in industrial control settings
- Funded by US Army under Small Business Innovation Research (SBIR) Grant
- Designed for touchscreen and hard-/soft-button navigation
- Set of custom SWT widgets and exemplary sample applications
- Includes application shell, menu bar, graphics oriented widgets, and pop-ups





Cyrano™ Overview



Industrial, Mining, Construction, & Military Applications

- **Building field production applications for mining and construction machines**
  - *Salt Harvester*
  - *Foundation Drill*
  - *Train Engines*

- **Proves viability of the architecture and implementation in fielded environments**

Mining & Construction Systems: Goals & Challenges

- **Goals**
  - *Shorten Application Development Time from 6-9 months to 2 weeks!*
  - *Tackle this in stages by building and refining the foundation*
    - 1st application: Duplicate existing reference application (6 months)
    - 2nd application: Salt Harvesting Machine (10 weeks)
    - 3rd application: Foundation Drill Machine (5 weeks)
- **Challenges**
  - *2 platforms to be supported: one legacy, one newly selected*
    - Legacy platform: 400MHz ARM, 32MB RAM/ROM, WindowsCE
    - New Platform: 400MHz PPC, 1GB RAM, 64MB ROM, Linux
  - *Machinery does not exist yet; we will never touch it*
  - *Incumbent native application to be displaced*

Challenge #1: New Platform Integration

- **Risks at Development Start**
  - *Platform not available until late in project schedule*
  - *SWT not running on the platform*
  - *CAN native JNI libraries did not exist*
- **Coping Strategy to Get Started**
  - *Started development with similar Linux platform*
  - *Used Gryphon (Ethernet-to-CAN) transport bundles*
  - *Found a CAN transport supported by same driver (PCAN-USB)*
  - *Built native JNI libraries and bundles to access CAN bus*
- **When Platform Became Available**
  - *Validated JNI libraries and bundles against platform CAN transceiver*
  - *Ran application headless while SWT in development*
  - *Ran entire application once SWT had been ported*

BANDXI INTERNATIONAL

Challenge #2: Salt Harvesting Controls (Sensors & Actuators)

- **Risks at Development Start**
  - *Custom machine that was still in design and development*
  - *Assembled on-site in Australia, half way around the world, remote*
  - *Limited testbench testing time available*
- **Coping Strategy to Get Started**
  - *Agreed on logical messages and application screen layour and flow*
  - *Built HTTP based simulator (self hosted)*
  - *Agreed on J1939 message specs: mix of standard and proprietary*
  - *Encoded as DeviceKit ML, built Device bundles*
  - *Ran against real ECU via Gryphon gateway from development box*
- **When Platform Became Available**
  - *Ran same stack on target platform*
  - *Swapped in target platform CAN transceiver bundles*
  - *Created HTTP based publisher*
    - Simulated ECU and advanced scenarios over real CAN link

Best Practices for Embedded OSGi (or any OSGi effort)

- Use Eclipse PDE and p2 tooling - *bundles are first class citizens!*
- Identify target platform early, put under version control
- Use only `Import-Package` and `Export-Package` for visibility
- Use Declarative Services to manage optional and required services
- Separate interfaces from implementation
- Separate application logic implementation from OSGi specific code
- Build fake, simulated, and real devices to keep moving
- Keep application work off OSGi threads (activation or call-backs)
- Assume nothing about start order
- Test shutdown behavior obsessively to expose loose ends
- Have at least one OSGi expert on your team
- Develop using OSGi – *even if it will not be used in final product*

Open for Discussion

- **Questions?**
- **Comments?**