Modern OBSW verification with Rust and data-oriented design patterns

ADCSS 2023

15th November 2023

Michaël Melchiore
ESTEC

AIRBUS

Export control Information

This document contains EU or / and Export Controlled technology (data) :

☐ YES ■ NO

If YES :

1/ European / French regulation controlled content

☐ Technology contained in this document is controlled by the European Union in accordance with dual-use regulation 2021/821 under Export Control Classification Number [x**E**xx]. ***(1)***

☐ Technology contained in this document is controlled by Export Control regulations of French Munitions List under Export Control Classification Number [MLXX or AMAXX]. ***(1)***

2/ US regulation controlled content

☐ Technology contained in this document is controlled under Export Control Classification Number [x**E**xxx] by the U.S. Department of Commerce - Export Administration Regulations (EAR). ***(1)***

☐ Technology contained in this document is controlled by the U.S. Department of State - Directorate of Defense Trade Controls - International Traffic in Arms Regulations (ITAR). ***(1)***

(1) See applicable export control license/authorization/exception in Delivery Dispatch Note.
Dissemination is only allowed to legal or natural persons with right to know who are covered by an appropriate export license/authorization/exception.

© Airbus Defense & Space SAS 2023

Study Context

Study “Using game engine techniques and Rust to modernize On Board software” (OXYDE)

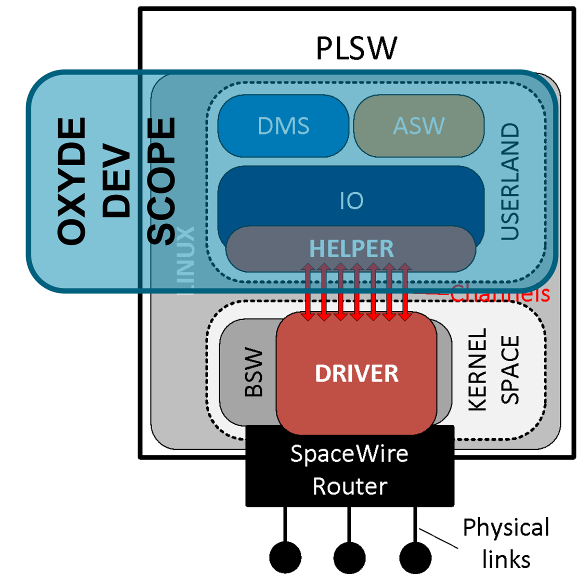
ESA Contract No.140066/22/NL/GLC/ov

Objectives

- Demonstrate space worthiness of Rust for OBSW development
- Evaluate cost & time saving due to potential simplification of ADS SDE & Core Products
- Update reference architecture guidelines to cope with highly-concurrent, heterogeneous OBSW

Plan

- Port one of our telecom payload to **Rust**, using selected **ECS design principles**
- Target ARM HW with custom Yocto-based Linux distribution
- Demonstrate Rust development across the **complete** OBSW development process
- Use current payload SW validation suite as reference



© Airbus Defense & Space SAS 2023
AIRBUS

OBSW Verification Stakes

Outstanding verification constraints

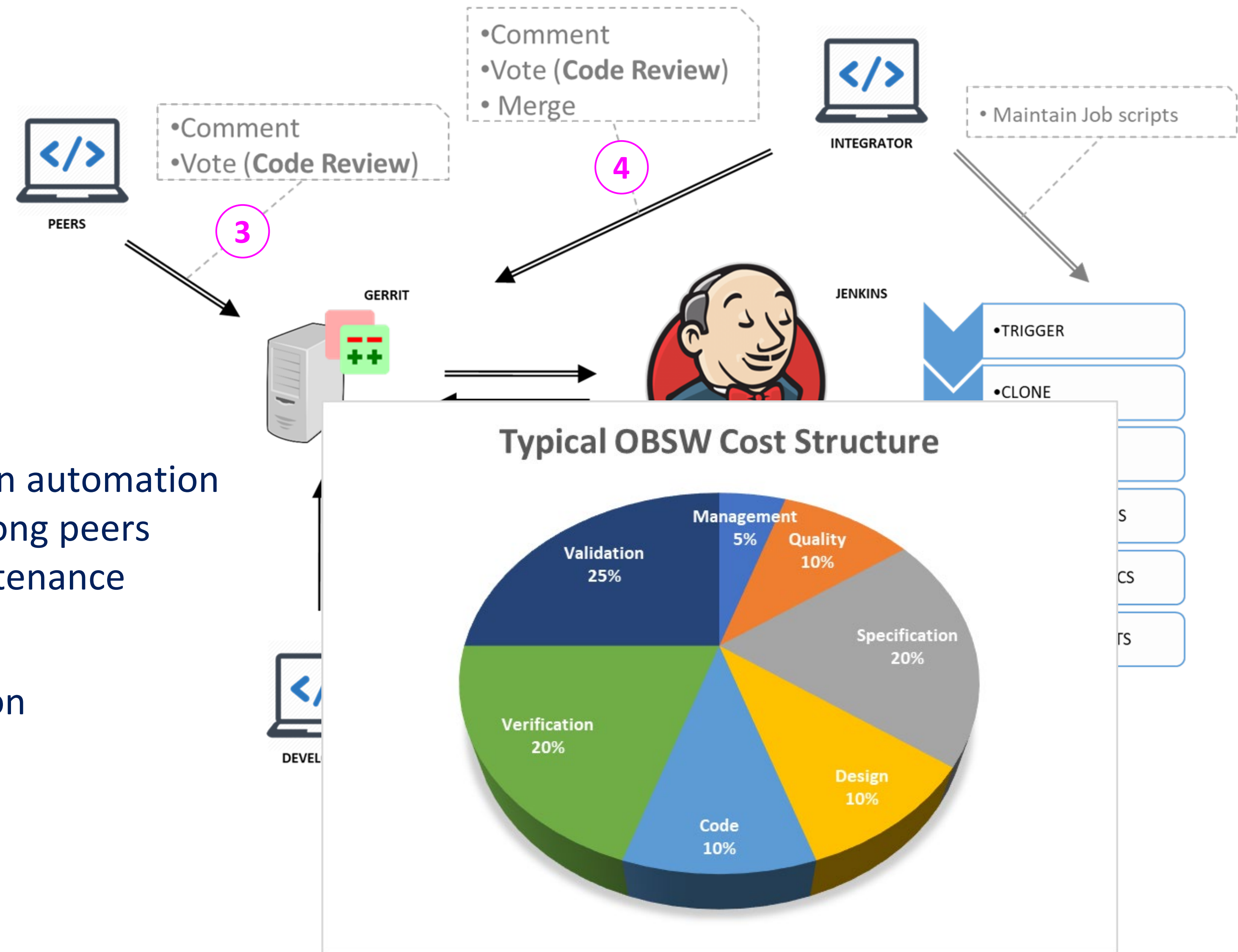
- Real-time SW
- Specialized, multi-modal equipments
- Remote operations
- Cover **all** possibilities

Recurring, cost & time investment

- Despite significant investment in verification automation
- Long reviews with numerous iterations among peers
- Complex SW Development Env. (SDE) maintenance

Expected benefits of a (more) modern verification

- Optimized cost & planning
- Risk mitigation through quicker iterations
- Unlock advanced features verifiability



ESA Contract No.140066/22/NL/GLC/ov
© Airbus Defense & Space SAS 2023
AIRBUS

SW Development Environment

Replicate C SDE for Rust development

- Rust community is aligned with our priorities
- Heavy focus on code quality
- Automated checks via (elaborated) tooling

Example: `cargo-deny`

- Check libraries for known advisories (CVEs…)
- Enforce licensing policy
- Deny specific libraries and version duplicates

Rust SDE is **significantly** leaner

- Code maintained by community
- Open, integrated ecosystem

Few limitations to be monitored

- MC/DC coverage not available (yet)
- Formal coding guidelines to be proposed

Coverage Report

Created: 2023-04-24 16:18

Click here for information about interpreting this report.

Filename	Function Coverage	Line Coverage	Region Coverage	Branch Coverage
crates/db/src/data.rs	0.00% (0/1)	0.00% (0/3)	0.00% (0/1)	- (0/0)
crates/db/src/lib.rs	100.00% (1/1)	100.00% (1/1)	100.00% (1/1)	- (0/0)
crates/plsw/src/applications/asw/mod.rs	0.00% (0/3)	0.00% (0/9)	0.00% (0/11)	- (0/0)
crates/plsw/src/applications/dms/mod.rs	0.00% (0/10)	0.00% (0/82)	0.00% (0/57)	- (0/0)
crates/plsw/src/io/rtc.rs	0.00% (0/4)	0.00% (0/35)	0.00% (0/13)	- (0/0)
crates/plsw/src/io/spw.rs	0.00% (0/15)	0.00% (0/122)	0.00% (0/103)	- (0/0)
crates/plsw/src/main.rs	25.00% (1/4)	1.20% (1/83)	3.85% (1/26)	- (0/0)
crates/plsw/src/pus/app.rs	0.00% (0/13)	0.00% (0/65)	0.00% (0/34)	- (0/0)
crates/plsw/src/pus/mod.rs	14.29% (1/7)	21.43% (3/14)	12.50% (1/8)	- (0/0)
crates/plsw/src/pus/tm_ack.rs	71.43% (10/14)	69.95% (128/183)	70.21% (33/47)	- (0/0)
crates/spwapi/spwapi-sys/src/lib.rs	100.00% (1/1)	100.00% (1/1)	100.00% (1/1)	- (0/0)
crates/spwapi/src/lib.rs	60.44% (55/91)	69.79% (476/682)	61.74% (234/379)	- (0/0)

```
552  2  pub fn set_addr(mut self, addr: u64) -> Result<Self, FrameError> {
553  2      if self.is_rmap() {
554  2          let result = unsafe { frame_rmap_set_addr(&mut self.0.frame, addr) };
555  2
556  2          match result {
557  2              0 => Ok(self),
558  0              _ => Err(FrameError::InvalidRmapPacketType),
559             }
560         } else {
561  0          Err(FrameError::InvalidProtocol)
562         }
563  2  }
564
```

Total	226	25653	100%	7	181	100%

ESA Contract No.140066/22/NL/GLC/ov
© Airbus Defense & Space SAS 2023
AIRBUS

Development Use Cases

Simple, asynchronous SpaceWire TC development

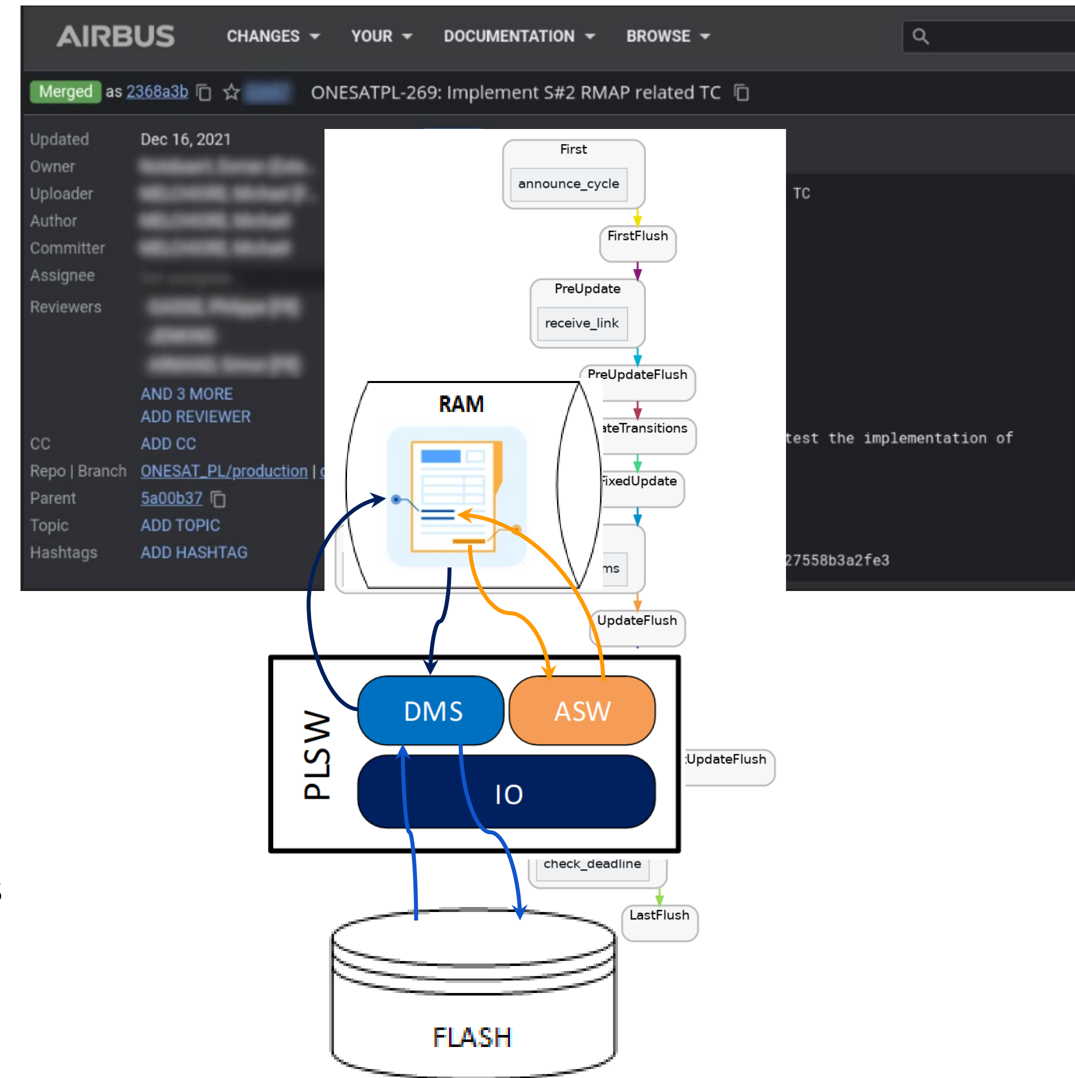
- Extracted typical C dev. pitfalls from *manual* review database
  - Memory management across API & thread boundaries
  - Buffers, dangling pointers, data copies, endianness…
- *Rust* encodes such constraints in its expressive type system
  - Copy vs. Clone, data ownership, borrow checker

Dynamic architecture of the PayLoad SoftWare (PLSW)

- Data and logical service dependencies, operational modes…
- Capture and maintain schedulability analysis hypotheses
- *ECS engine* encodes dynamic constraints through Rust type system
  - Developers implement systems as passive, stateless functions
  - Architects register systems onto execution schedules

Smart and reliable algorithm data duplication

- Prevent data corruption due to concurrent state updates during backups
- Minimize Flash device power on cycles to maximize device lifetime
- *Developers* encode business specific constraints in PLSW design



ESA Contract No.140066/22/NL/GLC/ov
© Airbus Defense & Space SAS 2023
AIRBUS

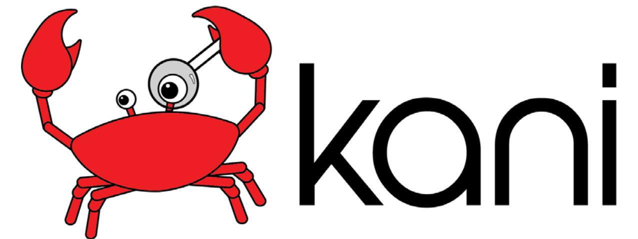
Conclusion

Rust provides a promising framework for automated, compile-time verification

- Expressive & flexible type system with unique semantics
- Leveraged by **everyone** to build efficient & reliable SW
- Emerging open, integrated tooling ecosystem
  - Property, fuzz testing: `proptest`, `cargo-fuzz`, `afl`…
  - Advanced code/model checkers: `MIRI`, `Kani`, `Loom`

Significant opportunities to optimize OBSW verification activities

- Focus manual reviews focus on problem understanding (Specification - Why?)
- ECS provide an architectural framework to statically enforce typical OBSW constraints
- Prepare separation of concerns in line with OSRA/SAVOIR principles



Currently identified enablers on which we should collaborate

- Spaceworthy OS/HW support
- OBSW-subset characterisation of the Rust ecosystem
- Ecosystem maturity growth

Ongoing industry initiatives !

ESA Contract No.140066/22/NL/GLC/ov

© Airbus Defense & Space SAS 2023

Thank you