Safe (Language) Interoperability

Combining multiple Programming Languages with Functional Safety

Christof Petig

27th of May 2025

16th AUTOSAR Open Conference

Bruges

Structure of the presentation

- **Recap of Functional Safety Requirements**
  - Spatial and temporal freedom from interference requirements
  - Software and Hardware solutions
  - Service Oriented Architecture
- Overview of Foreign Function Interface solutions
  - CORBA, COM, SWIG, bi-langual, SOAP, RESTful, gRPC, WIT
- Strengths and Limitations of WebAssembly Interface Types
- Advantages in a Software Defined Vehicle

Which “FFI” are we talking about?

This abbreviation is commonly used on both sides combined in this presentation

- Freedom From Interference is a key term in Functional Safety
- Foreign Function Interface is a common term in Computer Science when combining languages
- Add a style to disambiguate:

vs

Safety requirements

General

- Goal:
  Prevent harm by ensuring correct operation and handling of failures

- In Software:
  - Ensure correct operation:
    - Test against requirements
    - Define and test reaction to foreseeable failures
  - Prevent incorrect operation:
    - Minimize effect of failures … or attacks
    - Freedom from interference (FFI ✋)
      Spatial (memory) and temporal

Safety introduction

Memory safety clarification

Memory sounds like it is naturally spatial, but …

- Spatial memory safety
  - Buffer Overflow: An overflow on one element affects other elements
  - A wrong pointer value results in undefined behavior
- Temporal memory safety
  - Race Condition: An element is accessed while it is in an inconsistent state
  - Use after free: An element is accessed while it is no longer valid

Safety introduction

Spatial interference (one part of the code affects another one)

Many technologies exist to prevent this:

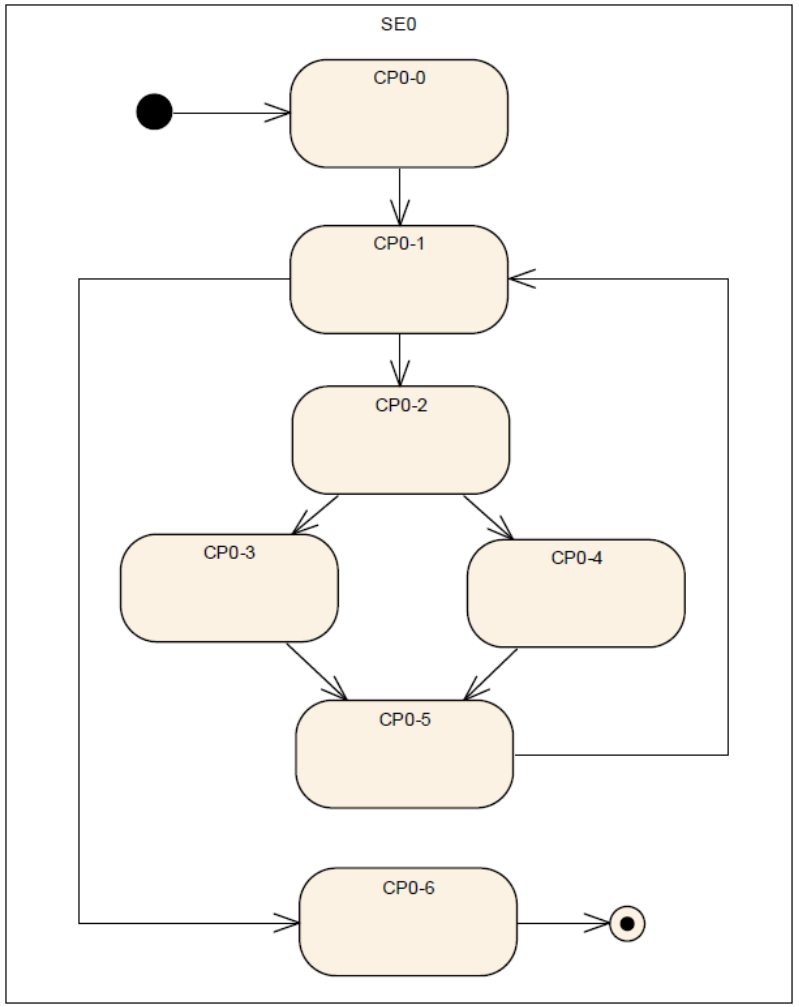
- Capability Hardware Enhanced RISC Instructions, CHERI 💰
- Containers (sandbox) 🐌
- Static or dynamic code analysis (valgrind)
- Memory safe languages
- End to end protection
- Instrumented code (SW fault isolation, SFI or HFI)

Safety Introduction

Temporal freedom from interference

Common solutions

- Watchdog (coarse)
- Health monitoring system (PHM)
- Timeout handling
- Fuel



FO ASWS Health Monitoring Figure 6.9

Safety requirements

Requirements on FFI 🤝

When combining different ASIL levels

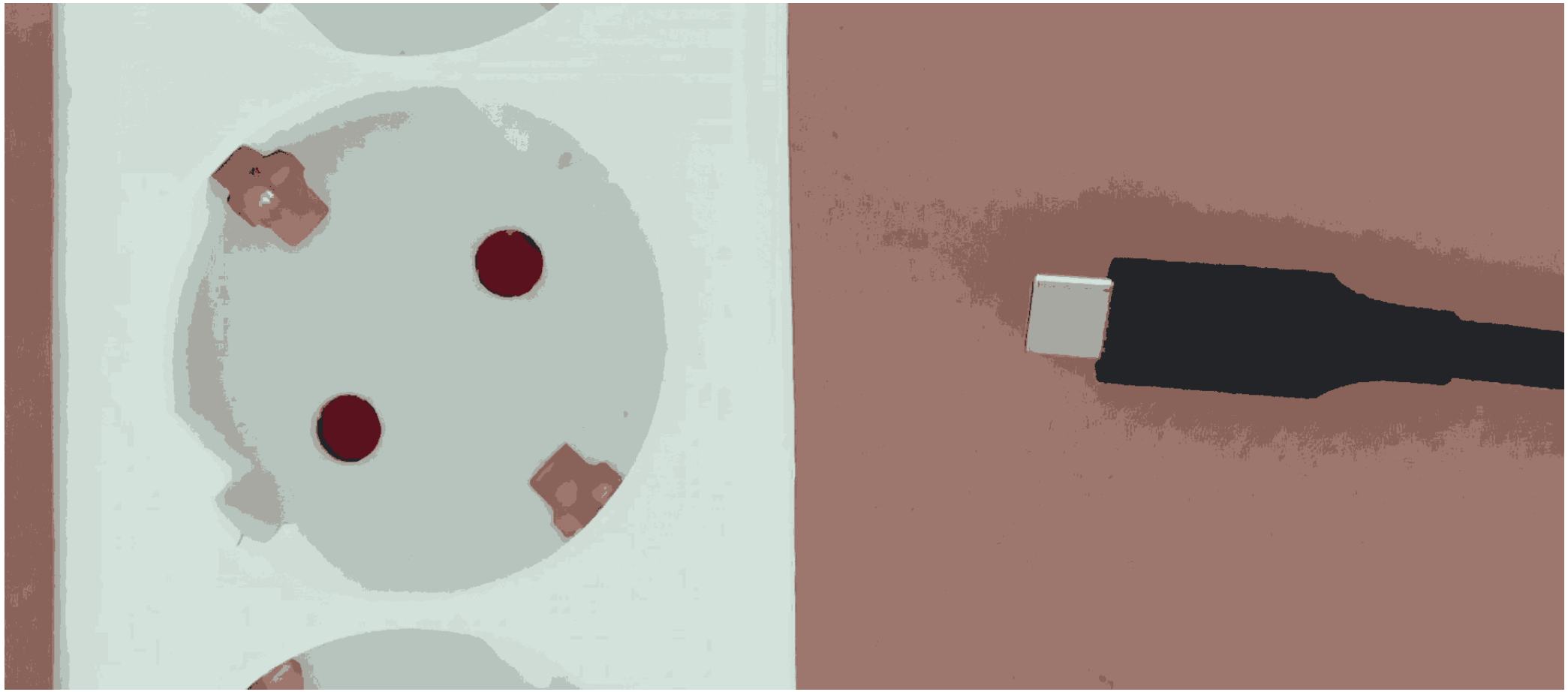
- Define and check
  - Validity of values at the border
  - Exception handling and restarting of sub-systems
  - Error conditions and behavior
  - Timing constraints
- Freedom from interference
  (memory protection/sandbox, timeout handling/fuel)
- Version control of all pieces
  Including interfaces ⇒ Semantic versioning

Structure of the presentation

- Recap of Functional Safety Requirements
  - Spatial and temporal freedom from interference requirements
  - Software and Hardware solutions
  - Service Oriented Architecture
- **Overview of Foreign Function Interface solutions**
  - CORBA, COM, SWIG, bi-langual, SOAP, RESTful, gRPC, WIT
- Strengths and Limitations of WebAssembly Interface Types
- Advantages in a Software Defined Vehicle

Why do we need Foreign Function Interfaces?

… because we often want to combine different technologies …



Foreign Function Interfaces: Introducing IDL

Interface Definition Language

- Specification of an **interface** in a **language neutral** way

- Defines types (struct, enum)
- Defines functions with arguments and return types

- Typically includes a standardized binary representation (ABI)

Foreign Function Interfaces: Grouping

- C compatible interfaces on both sides
- Combine two languages
- Multi-language, single process
- Local Inter Process Communication (IPC, shared memory)
- Network based solutions

Foreign Function Interface

C compatible interfaces

- Operating system and system libraries are written in C (1970-2020)
- The platform Application Binary Interface (**ABI**) standardizes the binary representation of types and location of data

- So, every major programming language can call C functions
  … and be called from C functions

- **Independently** define the Interface on both sides as a C interface

✓ done?

Foreign Function Interfaces: Two languages

Combining Rust with …

- rust-bindgen: Reads C(++) headers and generates matching Rust code
  ⚠ Correct include paths and conditional compilation
- cbindgen: Reads Rust and generates matching C(++) headers
- cxx: Rust style IDL describing Rust and C++ types and functions
- autocxx: More complex types for Rust and (unmodified) C++

- pyo3: Bidirectional high-level interaction between Rust and Python
- wasm-bindgen: Combine Rust with browser-based JavaScript

Multi-language, single process

Typically uses shared libraries (DLL or so) at language boundary

- Microsoft Component Object Model (COM), outdated
- Simplified Wrapper and Interface Generator (SWIG):
  Starts from C(++) and supports a variety of languages (but no Rust)
- Microsoft .NET via Common Language Infrastructure (CLI)
  Requires a runtime (CLR)
- WebAssembly Interface Types (WIT)
  Defined by Web Assembly Standard Interfaces (WASI) version 0.2+

Local Inter-Process Communication

Just for completeness

- Shared Memory + Signals (previous slide)
- Sockets (next slide)

- (Memory mapped) File
- (Named) pipe
- Message queue

Local Inter-Process Communication

Via shared memory; socket based on next slide

- Eclipse iceoryx
  - Base for Robot Operating System (ROS) 1+2,
    Cyclone Data Distribution System (DDS)
    and many more automotive frameworks
  - Current Version 2.0.6 ⚠
  - Supports Rust and C++ (more via DDS)
  - Safety certification available

- iceory2
  - Re-written in Rust, no central daemon, much faster
  - Version 0.5.0

New Concepts

- Service Oriented Architecture (SOA)
  - Multiple independent services
  - Self-contained functionality
  - Black box from out-side
  - Hierarchical Composability: Can call other services
- Container (e.g. “docker”)
  - Wraps a service in a binary format
  - Minimal assumptions about the host
- Orchestrator
  - Manages service instantiation via containers

Network based FFI

Needs serialization and deserialization

- Common Object Request Broker Architecture (CORBA), outdated
- Distributed COM (DCOM), outdated

- Data Distribution System (**DDS**)
- D-Bus, replacement for CORBA in Gnome and KDE
- **ara::com**: Adaptive Platform, SOME/IP or DDS based, mostly C++
- Hyper Text Transport Protocol (HTTP) based
  - SOAP uses eXtensible Markup Language (XML)
  - RESTful uses Java Script Object Notation (JSON)
  - **gRPC** uses Protocol Buffers (protobuf, binary)

- Table Columns: flexibility, types (future, stream), cost, designed for sandbox
- Rows: rust-bindgen, cxx, autocxx, cbindgen, CORBA, COM, SWIG, SOAP, RESTful, gRPC, WIT

AUTOSAR Adaptive Platform and Rust

WG-SAF-Rust

- Started with cbindgen in 2022
- Investigated into combining WIT with native compilation in February 2023
  - WASI 0.2 prototype with C++ application running in browser
  - Rust stack prototype in 2024
  - Rust application porting was too much effort (async runtime) compared to switching to WASI 0.3
- Open issues
  - Multiple subscribers and zero allocation need extension to WASI Using (Stream) Buffers (planned on for 0.4) and shared memory

Practical example

C interface with cbindgen (simple)

- Fusion calling Radar’s echo function
- Shows a simple function call with a string argument

What about safety?

- Interface is weakly defined
  - Should the callee free the string? (**ownership**)
  - Can the callee save the pointer for later? (**lifetime**)
- Avoid memory allocations (non-predictable timing)
  - All arguments should be passed **by reference**, only valid during the call
  - Results should be placed into a **buffer provided** by the caller
- Timing considerations
  - Is the callee allowed to **block** (e.g. call a service) during the call?
    This can be solved by asynchronous calls, but C++-17 requires multiple threads

Introduction to WIT

- Show echo call defined in WIT

Practical example

WIT (simple)

- Same example with WIT
- Sandbox vs no sandbox
- ~~Shows missing feature of WIT (caller provided buffers)~~

Practical example

C interface with cbindgen (event)

- A more complex example using event subscription (brake event) or future (calibrate call)

Practical example

WIT (event)

- Same with WIT

Applying Functional Safety Requirements

- Should illustrate nicely the simplification and handling of allocations
- This points to
  - Multiple subscribers and zero allocation need extension to WASI Using (Stream) Buffers (planned on for 0.4) and shared memory

Structure of the presentation

- Recap of Functional Safety Requirements
  - Spatial and temporal freedom from interference requirements
  - Software and Hardware solutions
  - Service Oriented Architecture
- Overview of Foreign Function Interface solutions
  - CORBA, COM, SWIG, bi-langual, SOAP, RESTful, gRPC, WIT
- **Strengths and Limitations of WebAssembly Interface Types**
- Advantages in a Software Defined Vehicle

WIT is designed for a Sandbox environment

- Pro
  - Prepared for future portable container technology
  - High insulation aids Functional Safety argumentation
  - Types map well with ara::core
- Contra
  - Constrains the data types and data flow modelling
  - Lacks support for zero-copy and multiple subscribers

Structure of the presentation

- Recap of Functional Safety Requirements
  - Spatial and temporal freedom from interference requirements
  - Software and Hardware solutions
  - Service Oriented Architecture
- Overview of Foreign Function Interface solutions
  - CORBA, COM, SWIG, bi-langual, SOAP, RESTful, gRPC, WIT
- Strengths and Limitations of WebAssembly Interface Types
- **Advantages in a Software Defined Vehicle**

- The same WIT based wasm binary containers (components) can run on cloud, phone, PC, central vehicle computers and microcontrollers
- Add a serialization option for network support (wrpc)


AUTOSAR

- Safety
  - General requirements; Safety guidelines for FFI
  - Memory safety: Spatial and Temporal
  - Solutions: Sandboxing, CHERI, SFI, HFI, valgrind
  - How these work with C and WIT in comparison
    - C: both sides need to agree, WIT: needs hashing to ensure matches?
- FFI solutions (rust-bindgen, cxx, autocxx, cbindgen, wit-bindgen)
- WIT concepts and limitations: resources, streams, shared-nothing
- Future WIT: zero copy, zero allocation via mutually attached shm buffers