Truly portable Vehicle Applications using Webassembly & WASI
Mathias Danzeisen, Research Engineer, ETAS GmbH
27 April 2023
Do we have Portable Vehicle Applications?

We want Application Software...

- develop for
- develop with
- write in
- communicate
- deploy on

- AUTOSAR
- Velocitas
- gRPC
- SOME/IP
- x86
- ARM64
- μC
- Android
- Java
- DDS Foundation
- 2
- ...
Do we have Portable Vehicle Applications?

We want Application Software...

Vehicle Signal Specification

develop for

AUTOSAR

develop with

Velocitas

write in

Android

communicate

Java

SOME/IP

deploy on

gRPC

x86

DDS Foundation

ARM64

μC

...
What is WebAssembly(Wasm)?

- Wasm defines an Instructions Set and an Execution Model
- Design Goals:
  - Portable code (Interpreted, Ahead-of-time compiled, JIT)
  - Performance near native-code performance
  - Safe and secure sandboxed
  - Streamable
  - Language-independent (C/C++, Rust, Java,...)
- Developed by W3C with support from major browsers, but with increasing support to run outside the browser
- Use cases span Cloud, Edge and embedded Devices (e.g. ARM M3/4)

Wasm designed to run code “fast, safe and efficient”
written in any language on any platform*

*) https://webassembly.github.io/spec/core/intro/introduction.html#design-goals
## Wasm: The Next Step Virtualization?

### Diagram
- **App**
- **Lib**
- **OS**
- **Hardware**

### Abbreviations
- WASM Host
- Libs
- OS
- Hardware

### Environment

<table>
<thead>
<tr>
<th>Environment</th>
<th>PC (Datacenter)</th>
<th>Cloud (public)</th>
<th>Cluster (e.g. K8S)</th>
<th>Wasm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format</strong></td>
<td>Image</td>
<td>VM</td>
<td>Container</td>
<td>WASM Module</td>
</tr>
<tr>
<td><strong>Developer Responsibility</strong></td>
<td>App &amp; Libs &amp; OS &amp; Hardware</td>
<td>App &amp; Libs &amp; OS</td>
<td>App &amp; Libs</td>
<td>Business Logic</td>
</tr>
<tr>
<td><strong>Abstraction</strong></td>
<td>Hardware</td>
<td>CPU</td>
<td>OCI/Linux kernel</td>
<td>Sandbox, WASI</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>System</td>
<td>OS</td>
<td>Process</td>
<td>Capability-based</td>
</tr>
</tbody>
</table>

*Based on: https://cosmonic.com/blog/*
WASI – The Standardized WebAssembly System Interface

- a specification to run WebAssembly outside the web
- family of APIs for WebAssembly
- currently is a subset of POSIX APIs (insecure and thread-unsafe APIs are dropped)
- focus on system-oriented APIs (files, networking, messaging, machine-learning,...)
- WASI vision is “capability based” system interface
- Interface definition language (component proposal)
Wasm Interface Type (WIT) format

```wasm
world my-world {
    import host: interface {
        use pkg.types.{errno}
        record ldata {
            ino: u64,
            size: u64,    // ...
        }
        log: func(param: ldata) -> result<errno>
    }
    export run: func()
}
```

- **world**
  - Top level definition of wasm component

- **Import & export**
  - Gives interfaces direction

- **interface**
  - Collection of function and types

- **use**
  - references

- **record/data types**
  - Complex data types can defined e.g.: record, enum, flag, union, variant

- **func**
  - Named functions with parameters and return value

See full specification at https://github.com/WebAssembly/component-model/blob/main/design/mvp/WIT.md
We want Application Software...

Idea: Portable Vehicle Applications with Wasm & WASI

- develop for
- develop with
- write in
- communicate
- deploy on

WEBASSEMBLY

We need:

- Some/IP
- ARM64
- µC
- x86
- ...

- AUTOSAR
- Velocitas
- android
- ...

- C
- ...

- gRPC
- MQTT
- DDS
- ...

- IDEA: Portable Vehicle Applications with WASM & WASI
Mapping: vss -> wit

Constrains from wit format:
- nesting of interfaces not possible (As of now )
- Future & and stream types for interfaces not available (yet)
- Notation: Only small letters and no dots are allowed in the interface name

Convert hierarchical structure of VSS
- Map tree to concatenated string
  - e.g.: vehicle-body-windshield-front-wiping-system-ispositionreached
- Map datatypes between VSS and wit
  - e.g.: boolean (Vss ) -> bool (wit);
  - float (vss) -> float32 (wit)
- Keep naming wit naming conventions: VSS snake case name

```c
// State of the supply voltage of the ECU
interface vehicle-lowvoltagesystemstate {
    enum vehicle-lowvoltagesystemstate-values {
        UNDEFINED,
        LOCK,
        ...
        START,
    }
    subscribe: func() -> bool
    unsubscribe: func() -> bool
    get: func() -> vehicle-lowvoltagesystemstate-values
}
```
vsstooling enhancement: vspec2wit (draft)

Usage:
```
$vspec2wit.py ../spec/VehicleSignalSpecification.vspec _result/veh.wit
```

Input: vspec  
Output: wasm interface
Proof of Concept (PoC)

Wasm component interfaces

default world wiper-poc {
    import vehicle-body-<>-ispositionreached: body.<>-ispositionreached
    export vehicle-body-<>-ispositionreached: body.<>-ispositionreached-cb

    export vehicle-app: interface{
        init: func() -> bool
        run: func() -> bool
    }
}

V-App (Rust) eg. [wiper-poc]

VSS + vehicles application interfaces

Wasm Runtime

grpc / mqtt

Linux/ RTOS
Proof of Concept (PoC)

Wasm component interfaces

```rust
default world wiper-poc {
    import vehicle-body-<>-ispositionreached: body.<>..>.ispositionreached
    export vehicle-body-<>-ispositionreached: body.<>..>.ispositionreached-cb
}

export vehicle-app: interface{
    init: func() -> bool
    run: func() -> bool
}
```
PoC Toolchain

**vspec2wit**

- **Vss.vspec**
- **Manual definition**
- **my-world.wit**
- **vss.wit**
PoC Toolchain

**Vss.vspec** → **vspec2wit** → **vss.wit**

**Wasm runtime**
- Implement binding
  - Manual coding

**compiler**

**runtime**

**Choose runtime**

**wasmtime**

Uses

**my-world.wit**

**Choose runtime**

**wit-bindgen**

**Choose language**

**rust**

**lib.rs**
- Implement binding
  - Manual coding

**compiler**

**wit-component**

**wiper.comp.wasm**

Runs on
PoC Integration

V-App (Rust) eg. [wiper-poc]

Wiper Motor

Wiper Motor

manuag lever

Safety guard

Safety guard

LIN-driver

LIN-driver

AUTOSAR Classic

AUTOSAR Classic

Domain-Crt

Domain-Crt

V-App (Rust) eg. [wiper-poc]

Wasm Runtime

Kuksa

Kuksa.val

grpc / mqtt

Linux/ RTOS

Linux/ RTOS

grpc / mqtt

Kuksa

Pub sub

Pub sub

Grpc Lib

Mqtt Lib

Mqtt Lib

Kuksa.val
Thank you!

Visit: etas.com

Mathias.Danzeisen@etas.com