OPC DAY
FINLAND 2023
30.11.2023

RUST Programming Language and OPC UA Status

Veli-Pekka Salo, Wapice Oy

FINNISH SOCIETY OF AUTOMATION
SUOMEN AUTOMAATIOSEURA RY

SPONSORS:

advenica BECKHOFF brightly NOKIA OPC FOUNDATION PROSYS OPC TGS prediktor Unified Automation Valmet FORWARD Wapice

Wapice. Unique approach, unique competence.

Digital transformation is a complex challenge. When you choose Wapice, you're choosing a partner with end-to-end technological competence. While others focus on one key element of the transformation, we have specialized in the entire digitalization chain for years.

Tailored to your needs:

Analytics, AI and Big Data
- Profiling and forecasting
- Anomaly detection & preventive maintenance
- Natural language, sound, video and image analysis
- Edge computing

Cloud development & management
- Optimization
- Architectures
- Security
- Migrations to cloud
- Hybrid & On-premises

Design services
- UI / UX design
- Conceptualisation
- Business consultation
- Testing and acceptance
- Web & mobile solutions

Connecting efficiently, reliably and securely
- Embedded and mobile devices
- Sensors and automation systems
- Edge gateways and Clouds
- Protocol engineering
- Fieldbus wired/wireless

Custom embedded solutions
- Electronics Design (Atex) and Manufacturing as a Service
- Embedded System Software Development
- Functional Safety Expertise
- Realtime system design

DevOps • Cyber Security • Consulting

DevOps
- DevOps Maturity Assessment
- Agile & DevOps Training
- Test Automation
- Pipeline Engineering

Cyber Security
- Code and Security Review
- Full stack Security Design
- Hardware Security Design
- Hardening Services
- Risk assessments

Technology Consulting
- Specifications and preliminary investigations
- Audits and reviews, Technology evaluations

Solution Consulting
- Consultation of processes and operating models
- Supporting new business models with IT solutions
- Specifications, investigations and analyses of business premises

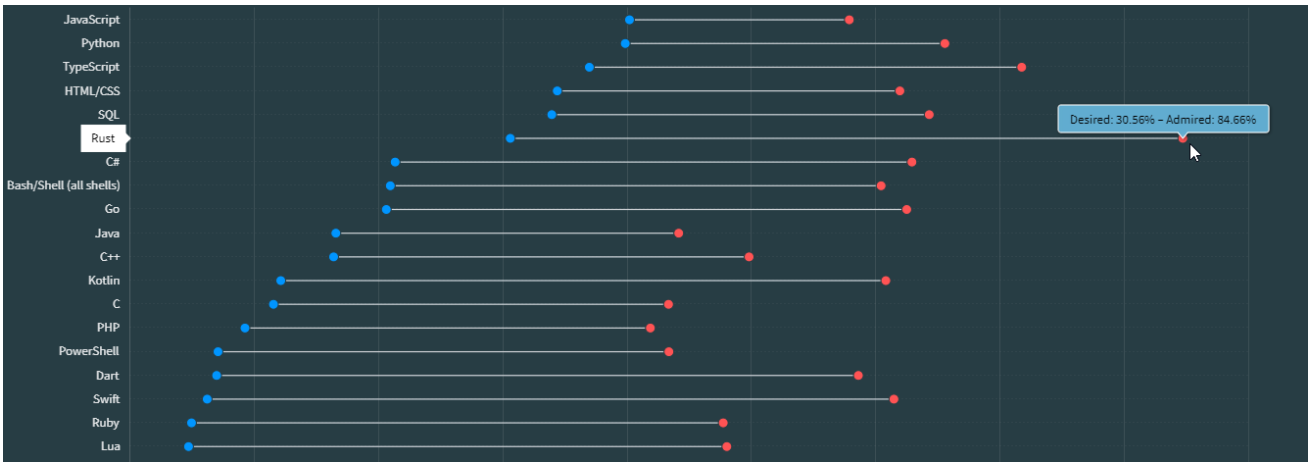
Ready-to-use products:

IoT-TICKET®
- No-code / low-code IoT platform
- Create IoT apps in minutes

Summium® CPQ
- Digitalise your sales process
- Visual sales configurator

EcoReaction
- Energy monitoring and reporting tool
- Customers able to manage contracts and invoices through 24 h self service

Rust - The most admired programming language!



https://survey.stackoverflow.co/2023/#section-admired-and-desired-programming-scripting-and-markup-languages

Rust background

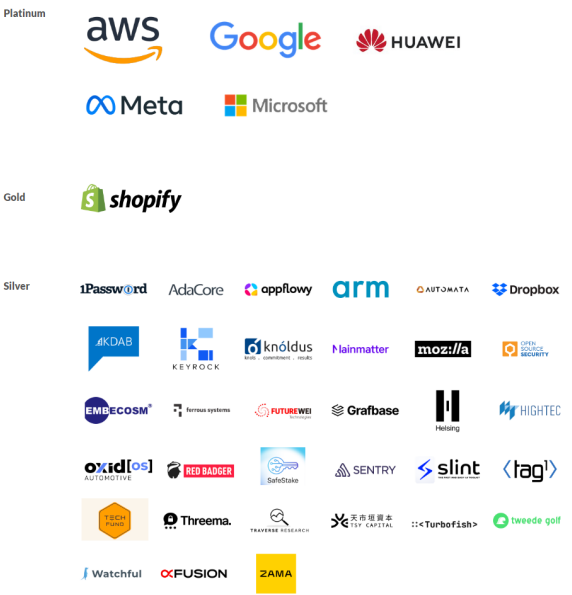
- Rust was developed as a **fast and memory safe alternative to languages like C and C++**
- Started 2006 as a personal project by Graydon Hoare in Mozilla research
- We at Wapice think that **Rust and OPC UA could be a good combination** in future
- While OPC UA solves security problems at protocol level, Rust does it **at source code level**

What is Rust?

- General purpose, systems programming language
  - **Cross-platform**
  - **Fast** – can be run even on bare metal.
  - **Control over** how memory is used
  - … but still completely **memory safe**
  - **Thread safe**
  - Rich **type system**
  - **Debugging** at compile time
  - Integrated **package management**
  - Good IDEs, Documentation, Community, etc…

Where is Rust used?

- Amazon Web Services
  - Some high-performance, secure infrastructure networking, and other systems software (e.g. Firecracker)
- Facebook
  - Source control backend was rewritten in Rust
- Dropbox
  - File-syncing engine is partially built with Rust code
- Cloudflare, Coursera, Discord, …
- Rust for Linux
  - “Series of patches to the Linux kernel that adds Rust as a second programming language to C for writing kernel components”



Why Rust is needed?

- Even though security and safety is more important than ever, the number CVEs reported by year is **growing**
- Microsoft estimate: 70% of vulnerabilities reported to MSRC caused by a **memory issue!**
- A method to prevent CVEs other than traditional ways is needed.

https://msrc.microsoft.com/blog/2019/07/why-rust-for-safe-systems-programming/

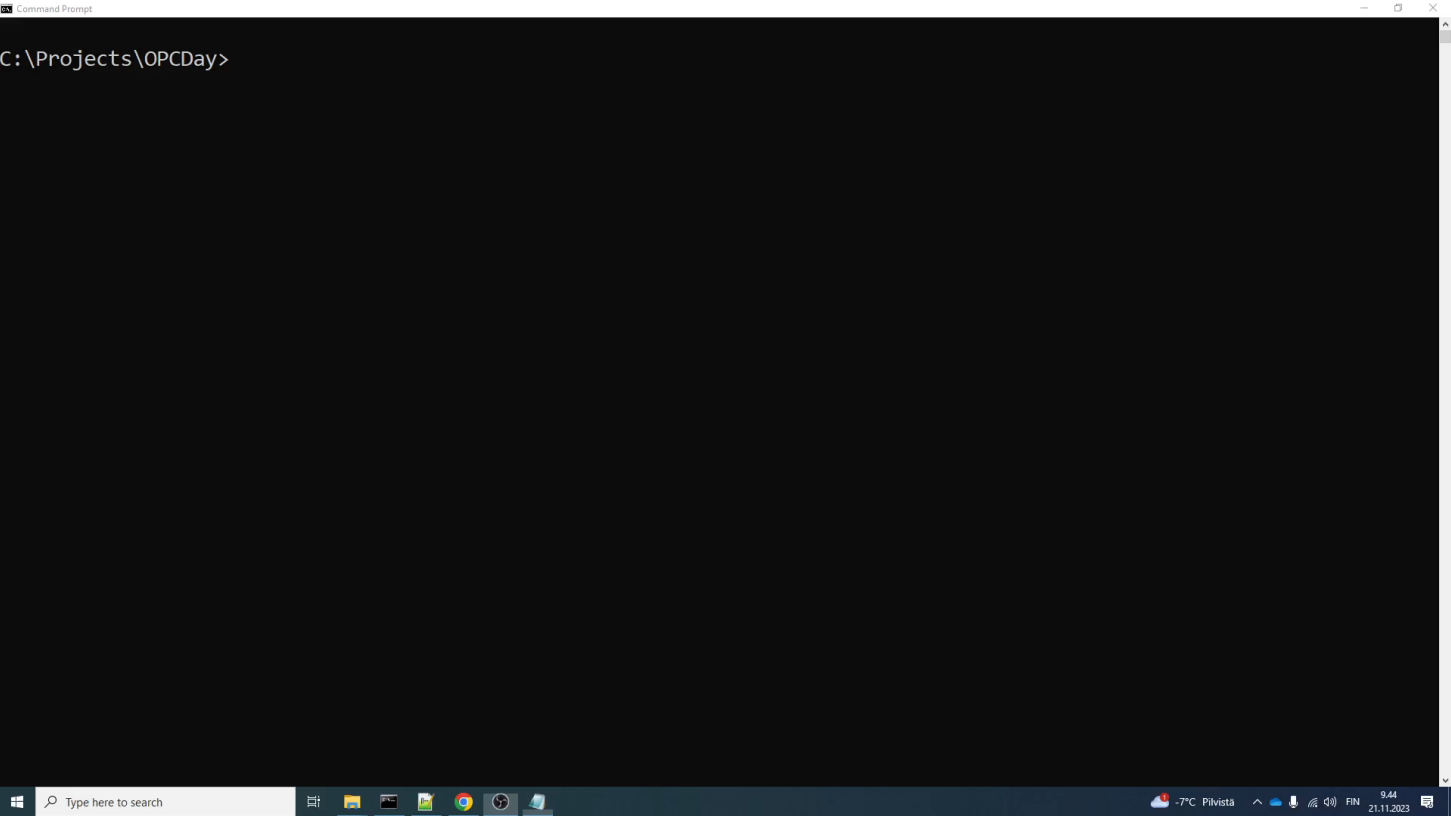
TOP 25 CVEs (memory type errors)

1. CWE-787 Out of bounds write
   - The product writes data past the end, or before the beginning, of the intended buffer
4. CWE-416 Use after free
   - Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code
7. CWE-125 Out of bounds read
   - The product reads data past the end, or before the beginning, of the intended buffer
12. CWE-476 Null pointer dereference
   - A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.

https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html

Example: Buffer overflow exploitation

```c
#include <stdio.h>
#include <string.h>

void vulnerable(void)
{
    // A buffer used for some task later on
    char buf[16];

    // A login status. Somewhere later in our code, we will use this to check if authentication is OK
    int isAuthenticated = 0;

    // Read input from stdin
    printf("\nWrite data to buffer: ");
    gets(buf);

    // Print buffer contents
    printf("\nYou wrote: %s\n", buf);
    printf("Press Any Key to Continue\n");
    getchar();

    // Print authentication status
    if (isAuthenticated == 0)
    {
        printf("isAuthenticated: %d\n", isAuthenticated);
        printf("So far, we are safe...\n");
    }
    else
    {
        printf("isAuthenticated: %d\n", isAuthenticated);
        printf("We are breached!!\n");
    }
}

int main(void)
{
    vulnerable();
    return 0;
}
```


Command Prompt
C:\Projects\OPCDay>
Type here to search
-7°C Pilvistä
FIN
9.44
21.11.2023

How Rust prevents exploiting coding errors

Ownership and Borrowing, Lifetimes, No null pointer, Bounds checking

Ownership

- Rust has **no garbage collector** (that regularly looks for no-longer-used memory as the program runs)
- Memory is managed through **set of rules that are checked at compile time**
  - Each value in Rust has an owner.
  - There can only be one owner to a piece of data at a time.
  - No need to allocate and free the memory: When the owner goes out of scope, the value will be dropped.
- Prevents from memory leaks.
  - Memory is automatically deallocated when it's no longer needed.

Borrowing

- Complex, but fundamental mechanism how Rust prevents issues when accessing data
- **Immutable borrowing**: borrower may not change the value
  - Enforces safe concurrent access of data
  - **No dangling references**. The compiler guarantees that data will ot go out of scope during the reference lifetime
  - Separate code that intends to read only vs code that modifies data
  - Allows compiler to optimize code
- **Mutable borrowing**: Borrower may change the value
  - Borrowing rules check that only one mutable borrow exists to particular set of data
- **Prevents from data races in multithreaded applications**

```rust
fn print_even(vectr: &Vec<i32>) {
    for values in vectr {
        if values % 2 == 0 {
            println!("{}", values);
        }
    }
}

fn main() {
    let number_vector = vec![1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

    // ownership is borrowed, not moved
    print_even(&number_vector);

    println!("Original vector{:?}", number_vector)
}
```

```rust
fn remove_value(vectr: &mut Vec<i32>) -> &Vec<i32> {
    vectr.remove(4);
    return vectr
}

fn main() {
    let mut nums = vec![1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    remove_value(&mut nums); // mutable reference here
    println!("{:?}", nums);
}
```

No null pointers

- *“I call it my billion-dollar mistake. It was the invention of the null reference in 1965” – Sir Tony Hoare, developer al ALGOL, 2009*
- In C and C++ null pointer is a pointer that does not point to a valid area.
- **Crashing the program often serves as an entry point to exploit vulnerabilities**
- Trigger exception → Crash → Revealed debugging information → Understand program logic → Plan subsequent attacks
- **Rust enforces safe use of None** through Option enum
  - Some(T): Represents a value of type T.
  - None: Represents the absence of a value.

No buffer overflows

- Bounds checking in arrays and slices
- Slice type
  - Reference to a portion of an array or another data structure.
  - Allows a safe and efficient access to a sequence of elements without owning the data
- Standard library uses vectors and strings that automatically resize when needed
- Ownership and borrowing rules: One thread cannot change the buffer, when other is accessing it. ;)

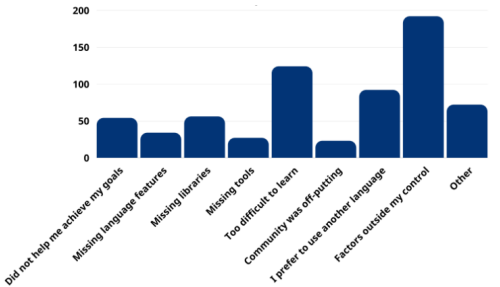
Summary of Rust safety mechanisms

- Immutable by default
  - Mutable variables require explicit declarations
- Type safety
  - Rust is a strongly typed language that enforces strict type checking at compile-time
- No null pointers - not allowed ;)
- No data races
  - Safe concurrent use of data
- No use after free errors and no dangling pointers or references
  - Lifetime of object is verified
- No buffer overflows
  - Strict rules how memory is managed
- Distinction between safe and ”unsafe” code
  - Attention to parts of code that need it

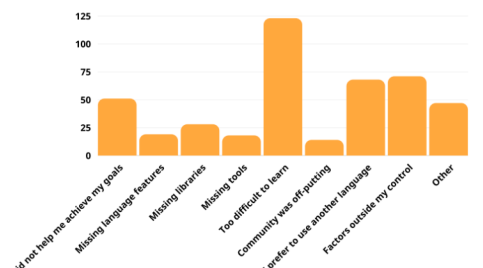
Other Rust features

- Cargo package manager
  - Download and install packages (Crates), resolve dependencies, compile the project
- Trait based generics
  - Object must implement a specific behavior defined by trait
  - Similar to interfaces in other languages
- Functional programming features
- Error handling
- Documentation and community
- Rust online book: https://www.rust-lang.org/learn

Why stopped using Rust?



Why not using Rust?



https://blog.rust-lang.org/2023/08/07/Rust-Survey-2023-Results.html

Rust OPC UA: Getting started

- OPC UA implementation by Adam Lock
  - https://github.com/locka99/opcua
- OPC UA Server/client implementation for Rust
- Mozilla Public License 2.0
- Equivalent to the OPC UA Embedded profile, which allows for:
  - Communication over opc.tcp:// binary protocol, Encryption and user identities, Subscriptions and monitored items, Events
  - Server profiles
    - http://opcfoundation.org/UA-Profile/Server/Behaviour - base server profile
    - http://opcfoundation.org/UA-Profile/Server/EmbeddedUA - embedded UA profile
- Tutorials for both client and server
- Cross-compilation: Raspberry PI example

Rust OPC UA: Getting started

- **simple-server** (publish some variables to address space and updates)
- **simple-client** (connects to a server and subscribes to variables)
- **discovery-client** (Connects to a discovery server and lists the servers registered on it)
- **chess-server** (Connects to a chess engine as its back-end and updates variables representing the state of the game)
- **demo-server** (More complex server. Can be used for compliance testing)
- **mqtt-client** (Subscribes to some values and publishes them to an MQTT broker)
- **web-client** (Subscribes to some values and streams them over a websocket)
- **modbus-server** (OPC UA server that translates variables from MODBUS.)

Rust OPC UA: Server

- Discovery service set
  - GetEndpoints
  - FindServers – stub (BadNotSupported)
  - RegisterServer – stub (BadNotSupported)
  - RegisterServer2 - stub (BadNotSupported)
- Attribute service set
  - Read
  - Write
  - History Read - 0.8+. Callbacks available.
  - History Update - 0.8+. Callbacks available.
- Session service set
  - CreateSession
  - ActivateSession
  - CloseSession
  - Cancel - stub implementation only
- Node Management service set
  - AddNodes
  - AddReferences
  - DeleteNodes
  - DeleteReferences
- Query service set
  - QueryFirst - stub (BadNotSupported)
  - QueryNext - stub (BadNotSupported)
- View service set
  - Browse
  - BrowseNext
  - TranslateBrowsePathsToNodeIds
- MonitoredItem service set
  - CreateMonitoredItems
    - Data change filter including dead band filtering.
    - Event filter
  - ModifyMonitoredItems
  - SetMonitoringMode
  - SetTriggering
  - DeleteMonitoredItems
- Subscription service set
  - CreateSubscription
  - ModifySubscription
  - DeleteSubscriptions
  - TransferSubscriptions - stub implementation
  - Publish
  - Republish
  - SetPublishingMode
- Method service set
  - Call

Rust OPC UA: Client

- The client API is synchronous
  - Request returns when the response is received, or a timeout occurs.
  - Under the hood it is asynchronous though.
- The client exposes functions that correspond to the current server supported profile
  - Look at the server services and there will be client-side functions that are analogous to those services.
- In addition to the server services, the following are also supported.
  - FindServers - when connected to a discovery server, to find other servers
  - RegisterServer - when connected to a discovery server, to register a server.

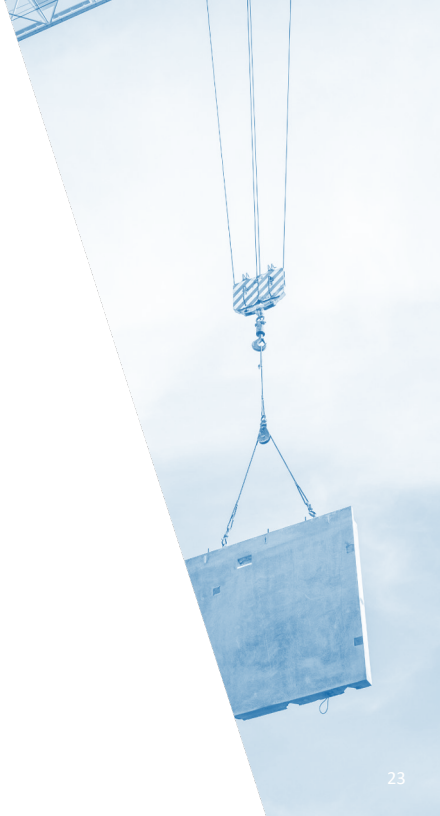
Rust OPC UA: Encryption

- Message security modes
  - None
  - Sign
  - SignAndEncrypt
- Security policies
  - None
  - Basic128Rsa15
  - Basic256
  - Basic256Rsa256
  - Aes128-Sha256-RsaOaep
  - Aes256-Sha256-RsaPss

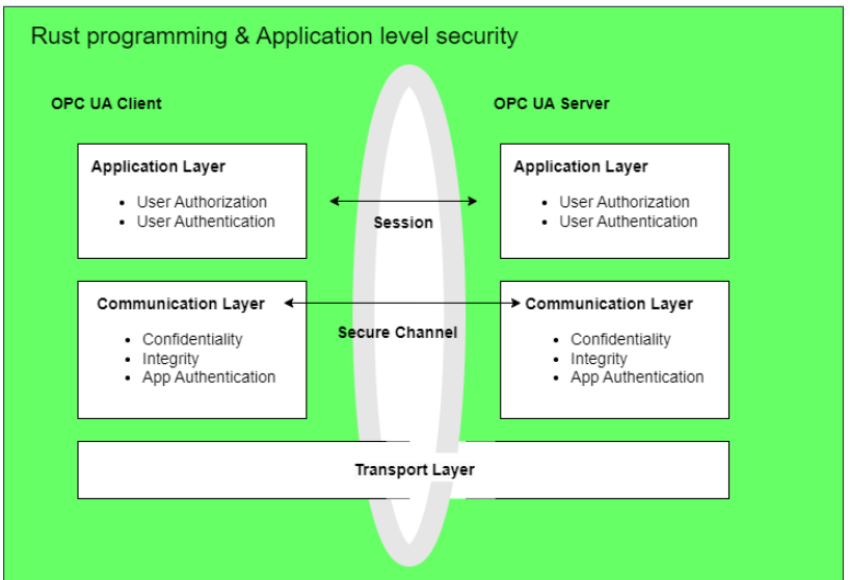
- User identities
  - Anonymous - i.e. no identity
  - UserName - encrypted and plaintext. User/pass identities are defined by configuration.
  - X509 certificates
- Crypto
  - Sign, verify, encrypt and decrypt data.
  - Create, load and save certificates and keys.

Rust OPC UA: Future work

- JSON serialization of most built-in data types (Tag 0.12.0)
- Update to Rust 2021 profile (Tag 0.12.0)
- Increase asynchronous processing of operations.
- User-level permission model, i.e. ability to limit access to address space based on identity
- Replace OpenSSL with a native Rust equivalent library (OpenSSL is external to Rust and implemented in C so it adds complexity)
- Rust crypto / PKI related crates are not yet sufficient to replace OpenSSL
- Tokio codec - use a codec and frame writer to write message chunks (Tokio is runtime for writing reliable asynchronous applications in Rust programming language)



Rust OPC UA: Conclusion



CREATING A SMARTER
FUTURE TODAY

Visit **wapice.com**