STRESZCZENIE

Krótki wstęp.

Część główna, która powinna być nieco dłuższa. Całe streszcznie powinno zająć około pół strony.

Krótkie podsumowanie wniosków, wyników i ewentualnych proponowanych kolejnych kroków.

Słowa kluczowe: słowo kluczowe 1, słowo kluczowe 2, słowo kluczowe 3

Dziedzina nauki i techniki, zgodnie z wymogami OECD: elektrotechnika, elektronika, inżynieria informatyczna

ABSTRACT

A short introduction.

The main part which should be a bit longer. The whole abstract should take approximately a half page.

A short summary of the outcomes, results, and proposed next steps (if any).

Keywords: keyword 1, keyword 2, keyword 3

Field of science and technology in accordance with OECD requirements: electrical engineering, electronic engineering, information engineering

SPIS TREŚCI

WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

CFD – Computational Fluid Dynamics (obliczeniowa mechanika płynów)

UE – Unia Europejska

1. WPROWADZENIE

Tekst rozdziału do napisania w terminie późniejszym.

1.1. CEL PRACY

Celem pracy dyplomowej jest analiza oraz praktyczna realizacja migracji fragmentu programu udostępnionego przez fundację Mozilla z języka C++ do języka Rust w celu oceny korzyści związanych z bezpieczeństwem i nowoczesnością kodu.

1.2. PRZEGLĄD ROZDZIAŁÓW

W rozdziale drugim pracy przedstawiono istniejące rozwiązania dotyczące migracji kodu z języka C++ do języka Rust, ze szczególnym uwzględnieniem projektów realizowanych przez społeczność open source oraz inicjatyw fundacji Mozilla, które ilustrują praktyczne podejścia i narzędzia wspierające ten proces.

2. ISTNIEJĄCE ROZWIĄZANIA

W niniejszym rozdziale przedstawiono kilka przykładowych rozwiązań wykorzystywanych w procesach migracji kodu źródłowego z języka C++ do języka Rust, ze szczególnym uwzględnieniem narzędzi automatyzujących ten proces oraz doświadczeń z projektów open source, takich jak te rozwijane przez fundację Mozilla.

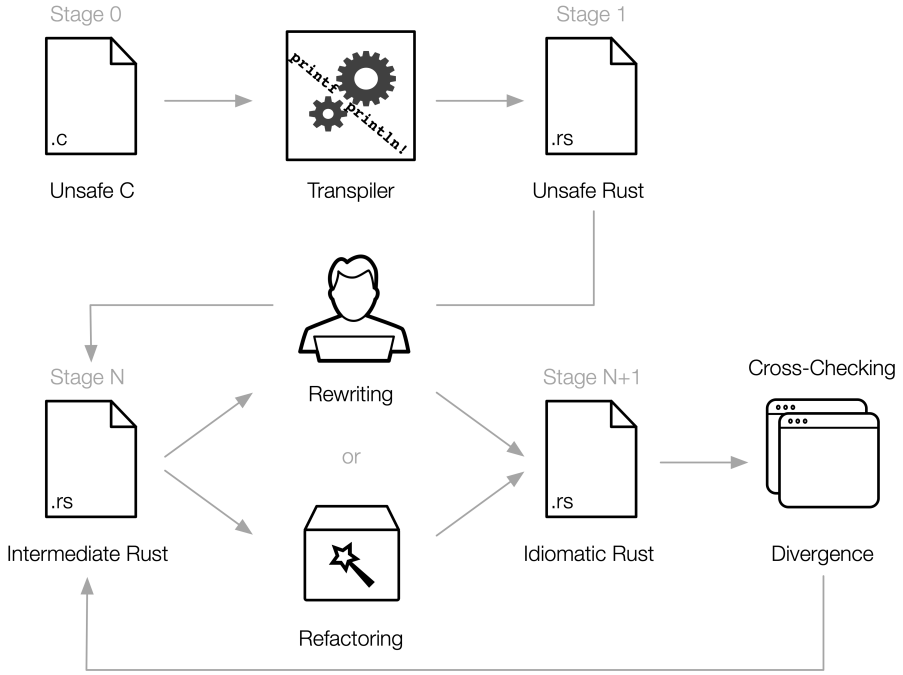
2.1. NARZĘDZIA DO AUTOMATYCZNEJ KONWERSJI C++ NA RUST

W procesie migracji kodu z C++ do Rust wykorzystywane są narzędzia wspomagające automatyzację, choć pełna konwersja nadal wymaga ręcznego dostosowania ze względu na różnice semantyczne między językami. Przykładowe narzędzia to:

- C2Rust framework umożliwiający translację kodu C (i częściowo C++) do Rust, wykorzystujący Clang do parsowania kodu źródłowego. Narzędzie generuje niskopoziomowy kod, który wymaga późniejszej refaktoryzacji (np. wprowadzenia bezpiecznych abstrakcji). [1]
- Bindgen narzędzie rozwijane przez Mozilla, automatycznie generujące powiązania (ang. bindings) kodu Rust do C/C++. [2]
- Corrode eksperymentalny translator C do Rust.[3]

Tłumacz (lub transpiler) generuje niebezpieczny kod w języku Rust, który ściśle odwzorowuje wejściowy kod w języku C. Głównym celem tłumacza jest wygenerowanie kodu funkcjonalnie identycznego z kodem źródłowym w C. Celem tego narzędzia nie jest generowanie bezpiecznego lub idiomatycznego kodu. Najlepszym podejściem dla programisty jest stopniowe przekształcanie przetłumaczonego kodu w języku Rust przy użyciu dedykowanych narzędzi do refaktoryzacji.

Praca z tymi narzędziami może przebiegać w następujący sposób:



Rys. 2.1. Proces tłumaczenia i przekształcania kodu C na idiomatyczny kod w języku Rust(przygotowano na podstawie[1])

2.2. PROJEKTY OPEN SOURCE MIGRUJĄCE Z C++ DO RUST

- Firefox (Mozilla) - stopniowa migracja komponentów (np. silnik CSS Stylo), z wykorzystaniem Rust do poprawy bezpieczeństwa pamięci. Mozilla opracowała też RLBox, narzędzie do sandboxowania niebezpiecznego kodu C++.[4]

- Servo - eksperymentalna przeglądarka napisana całkowicie w Rust, której fragmenty (np. WebRender) zostały zintegrowane z Firefoxem.[5]

- Deno - (JavaScript lub TypeScript runtime) – używa Rust dla wydajnych modułów, podczas gdy core jest w C++.[6]

- Linux Kernel - od wersji 6.1 wspiera Rust jako drugi język systemowy, co umożliwia migrację wybranych modułów.[7]

Projekty te pokazują, że migracja często odbywa się modularnie, z zachowaniem interoperacyjności przez Foreign Function Interface(FFI).

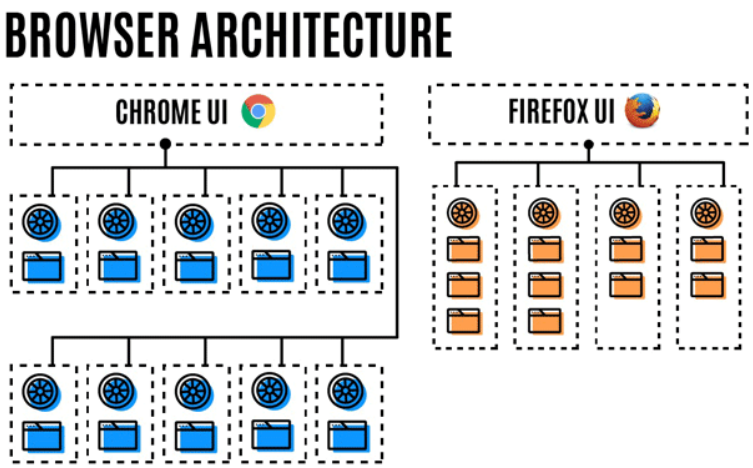
2.3. MIGRACJA KOMPONENTU STYLO (CSS ENGINE) Z C++ DO RUST W PROJEKCIE FIREFOX

Przeglądarka Firefox, rozwijana przez fundację Mozilla, od wielu lat stanowi jedno z głównych środowisk testowych i produkcyjnych dla języka Rust. W ramach projektu *Quantum* zainicjowano serię modernizacji komponentów Firefox, której celem było poprawienie wydajności i bezpieczeństwa. Jednym z najbardziej znaczących efektów tej inicjatywy była migracja silnika CSS, znanego jako *Stylo*, z języka C++ do Rust. To przedsięwzięcie stanowi wzorcowy przypadek efektywnej migracji komponentu systemowego o wysokim stopniu złożoności.

2.3.1. PROJEKT STYLO

Silnik CSS odpowiada za przetwarzanie stylów arkuszy i ich stosowanie do drzewa DOM (Document Object Model) w czasie renderowania strony. Poprzedni silnik (*Gecko*), napisany w C++, miał ograniczoną możliwość wydajnej równoległości i był narażony na typowe problemy z zarządzaniem pamięcią. Rust, jako język systemowy bezpieczny pamięciowo, oferował realną szansę na poprawę niezawodności i skalowalności komponentu CSS[8].

Migracja Stylo została poprzedzona fazą eksperymentalną w ramach projektu Servo — nowej przeglądarki pisanej od podstaw w Rust. Na podstawie rezultatów z Servo, komponent *WebStylo* został przekształcony i zaadaptowany do Firefox jako *Stylo*.



Rys. 2.2. Architektura Chrome w porównaniu do Firefox (na podstawie Firefox 57 "Quantum")

Klasyczna architektura wieloprocesowa (Chrome):

- Wyspecjalizowane procesy: oddzielne dla każdej karty (tab), rozszerzeń (extensions) i GPU,
- Izolacja przez nadmiar: każda karta = nowy proces (wysokie zużycie RAM),
- Hierarchia kontrolna: proces główny (browser) zarządza procesami potomnymi.

Podejście Quantum:

- Hybrydowy model procesów:
  - jeden główny proces zarządzający (Parent),
  - procesy treści (Content) współdzielone między kartami,
  - dedykowane procesy dla krytycznych komponentów (GPU, Network),
- Optymalizacja zasobów:
  - współdzielenie pamięci dla podobnych stron,
  - dynamiczne alokowanie procesów wg potrzeb,
- Modułowość: wymienne komponenty i lepsza skalowalność.

Kluczowe komponenty nowej architektury:

- **Quantum Flow**
  - Priorytetyzacja zadań: System kolejek oparty o krytyczność operacji
  - Pipeline renderingu: Równoległe przetwarzanie etapów wyświetlania strony
  - Przeplot wątków: Wykorzystanie wszystkich rdzeni CPU
- **Quantum CSS**
  - Równoległe drzewo stylów: Podział pracy na niezależne fragmenty
  - Cache współdzielony: Jedna kopia stylów dla identycznych elementów
  - Inkrementalne aktualizacje: Minimalizacja przeróbek przy dynamicznych zmianach
- **Quantum Render (WebRender)**

- Kompozytowanie na GPU: Traktowanie strony jako sceny 3D
- Listy wyświetleń: Optymalizacja przekazywania danych do karty graficznej
- Wektorowy pipeline: Bezstratne skalowanie elementów UI

2.3.2. PRZEBIEG MIGRACJI I INTEGRACJA STYLO Z FIREFOX

Stylo został zaprojektowany jako komponent kompatybilny z istniejącym systemem budowania Firefoksa. Umożliwiło to tzw. *dual compilation* – kompilowanie części przeglądarki w Rust, a pozostałych w C++. Komunikacja między językami odbywa się poprzez FFI (Foreign Function Interface), co wymagało stworzenia bezpiecznych interfejsów i utrzymania zgodności ABI.

Migracja przebiegała etapami, zaczynając od funkcji odpowiedzialnych za selekcję stylów, a następnie przekształcając kolejne moduły[9]. Każdy etap podlegał intensywnemu testowaniu, zarówno funkcjonalnemu, jak i porównawczemu z poprzednią implementacją C++. Wprowadzenie Rust pozwoliło na równoległe przetwarzanie drzew stylów, co znacząco poprawiło wydajność renderowania.

2.3.3. REZULTATY I ZNACZENIE PROJEKTU STYLO

Migracja silnika Stylo do Rust przyniosła korzyści:

- **Wydajność:** znaczący wzrost wydajności przeglądarki, szczególnie w obszarach dotyczących równoległego stylowania złożonych drzew DOM,
- **Bezpieczeństwo:** redukcja błędów pamięci typowych dla C++,
- **Inspiracja:** projekt stał się wzorem dla dalszych migracji komponentów Firefoksa.

Stylo jest wyłącznym rozwiązaniem Mozilla, rozwijanym tylko dla przeglądarek Servo i Firefox. Od wersji Firefox 57 (Quantum) zastąpił tradycyjny silnik Gecko CSS, wykorzystując architekturę zapoczątkowaną w projekcie Servo. Stylo działa jako hybrydowy silnik - w Firefox wykorzystuje zarówno komponenty Rust (Servo) jak i C++ (Gecko), podczas gdy w Servo istnieje jako czyste rozwiązanie w Rust.

Tabela 2.1. Porównanie tradycyjnych silników CSS i Stylo (Quantum CSS)

Cecha	Gecko (C++)	Stylo (Rust)
Przebieg stylowania	Sekwencyjny	Równoległy
Bezpieczeństwo	Manualne zarządzanie pamięcią	Automatyczne (Rust)
Wydajność	1x	Do 18$\times$ na wielordzeniowych CPU
Kompatybilność	Wszystkie przeglądarki	Tylko Firefox/Servo

- 80% redukcji błędów bezpieczeństwa pamięci
- 2-4× szybsze stylowanie stron
- 30% mniejsze zużycie RAM przy złożonych stylach

Projekt Stylo dowodzi, że migracja nawet bardzo złożonych komponentów systemowych jest możliwa i opłacalna, pod warunkiem dobrej integracji narzędzi, testów oraz wsparcia ze strony zespołu inżynierów. Stylo pozostaje jednym z flagowych przypadków użycia Rust w produkcyjnym środowisku i fundamentem sukcesu projektu Quantum.

2.3.4. SUKCES RYNKOWY FIREFOX QUANTUM

Wprowadzenie silnika Stylo w Firefox 57 (*Quantum*) w listopadzie 2017 roku stanowiło punkt zwrotny dla przeglądarki Mozilli:

- Firefox odzyskał 15% użytkowników w ciągu 6 miesięcy od premiery,
- powstało ponad 100 nowych rozszerzeń stworzonych specjalnie dla Quantum,
- nagroda *WebAward* dla najszybszej przeglądarki 2018.

2.3.5. WNIOSKI PROJEKTU STYLO

Przykład migracji Stylo pokazuje, że sukces transformacji kodu do Rust zależy nie tylko od możliwości technicznych, ale również od przyjętej strategii organizacyjnej i zdolności utrzymania kompatybilności z istniejącą bazą kodu. Mozilla, jako pionier wykorzystania Rust w praktyce, wyznaczyła kierunek rozwoju dla innych organizacji poszukujących nowoczesnych metod poprawy jakości oprogramowania systemowego.

2.4. EKSPERYMENTALNA PRZEGLĄDARKA SERVO

Servo to eksperymentalna przeglądarka internetowa rozwijana przez fundację Mozilla, napisana całkowicie w języku Rust. Głównym celem projektu było stworzenie nowoczesnego silnika przeglądarkowego, który wykorzystuje zalety Rust do poprawy wydajności i niezawodności. Servo stał się poligonem doświadczalnym dla wielu innowacyjnych rozwiązań, które później zintegrowano z Firefoxem[5].

2.4.1. ARCHITEKTURA I KLUCZOWE KOMPONENTY SERVO

Servo został zaprojektowany z myślą o modularności i równoległym przetwarzaniu. Jego architektura obejmuje:

- Silnik renderowania *WebRender*: wykorzystuje GPU do komponowania stron, traktując je jako sceny 3D,
- Silnik stylów *Stylo*: równoległe przetwarzanie CSS, które później zostało włączone do Firefoxa,
- Parser HTML i DOM: zoptymalizowany pod kątem bezpieczeństwa i wydajności,
- Wsparcie dla WebAssembly: umożliwia wykonywanie wysokowydajnego kodu w przeglądarce.

2.4.2. PRZEBIEG ROZWOJU I INTEGRACJA SERVO Z FIREFOXEM

Projekt Servo rozpoczął się w 2012 roku jako eksperyment mający na celu przetestowanie możliwości Rust w kontekście przeglądarki. W miarę rozwoju kluczowe komponenty Servo, takie jak *WebRender* i *Stylo*, zostały zintegrowane z Firefoxem w ramach projektu *Quantum*. Dzięki temu Firefox zyskał nowoczesne funkcje, zachowując jednocześnie kompatybilność z istniejącym kodem C++[4].

2.4.3. REZULTATY I ZNACZENIE PROJEKTU SERVO

Servo przyniósł następujące korzyści:

- **Wydajność:** zastosowanie równoległego przetwarzania znacznie przyspieszyło renderowanie stron,
- **Bezpieczeństwo:** brak błędów pamięciowych (typowych dla C++),
- **Innowacje:** Servo stał się inspiracją dla innych projektów wykorzystujących Rust (np. Deno).

2.4.4. WNIOSKI SERVO

Projekt Servo pokazał, że Rust nadaje się do budowy złożonych systemów, takich jak przeglądarki internetowe. Jego modularność i interoperacyjność z C++ umożliwiły stopniowe wdrażanie nowych rozwiązań w istniejących projektach, co jest kluczowe dla dużych organizacji.

2.5. ŚRODOWISKO WYKONAWCZE DENO DLA JAVASCRIPT, TYPESCRIPT I WEBASSEMBLY

Deno to nowoczesne środowisko wykonawcze dla JavaScript, TypeScript i WebAssembly[6]. Deno zostało napisane w Rust, co zapewnia mu wysoką wydajność i bezpieczeństwo. Głównym celem projektu było rozwiązanie problemów Node.js, takich jak złożony system zarządzania zależnościami i brak wsparcia dla TypeScript out-of-the-box.

2.5.1. ARCHITEKTURA I KLUCZOWE KOMPONENTY DENO

Deno opiera się na następujących komponentach:

- Rust jako podstawa: większość funkcji systemowych jest zaimplementowana w Rust,
- Modułowy system bezpieczeństwa: Deno domyślnie uruchamia kod w sandboxie, co minimalizuje ryzyko zagrożeń,
- Wsparcie dla WebAssembly: umożliwia wykonywanie kodu napisanego w innych językach,
- Silnik V8: ten sam silnik JavaScript używany w Chrome i Node.js.

Deno wykorzystuje język Rust do implementacji niskopoziomowych funkcji, takich jak operacje wejścia/wyjścia (I/O) czy zarządzanie procesami. Komunikacja między JavaScriptem a Rust odbywa się za pośrednictwem interfejsu Foreign Function Interface (FFI), co umożliwia zachowanie wysokiej wydajności przy jednoczesnym zapewnieniu bezpieczeństwa.

2.5.2. REZULTATY I ZNACZENIE PROJEKTU DENO

Deno przyniósł następujące korzyści:

- **Wydajność:** dzięki Rust Deno osiąga lepszą wydajność niż Node.js (w niektórych zadaniach),
- **Bezpieczeństwo:** sandboxing i domyślne ograniczenia minimalizują ryzyko ataków,
- **Nowoczesne funkcje:** wsparcie dla TypeScript i WebAssembly out-of-the-box.

2.5.3. WNIOSKI DENO

Deno jest przykładem udanego połączenia JavaScript i Rust, pokazując, że migracja wybranych komponentów do Rust może przynieść znaczące korzyści w zakresie wydajności i bezpieczeństwa.

2.6. INICJATYWY MOZILLA WSPIERAJĄCE MIGRACJĘ

Mozilla, jako jeden z głównych fundatorów rozwoju Rust, prowadzi projekty ułatwiające przejście z C++:

- **Oxidization** – wewnętrzny program Mozilla mający na celu identyfikację komponentów Firefox, których migracja do Rust przyniesie największe korzyści bezpieczeństwa[10],
- **CXX** – biblioteka do bezpiecznej interoperacyjności C++ i Rust, minimalizująca ryzyko błędów na styku języków[11],
- **Rust-C++ dual compilation** – wsparcie w build systemie Firefox dla mieszanych projektów[12].

Działania te pokazują, że migracja w dużych projektach wymaga nie tylko narzędzi, ale też wsparcia organizacyjnego i rozwoju oprogramowania.

3. MIGRACJA FRAGMENTU KODU Z JĘZYKA PROGRAMOWANIA C++ NA JĘZYK RUST

Tekst do napisania w terminie późniejszym.

4. TESTOWANIE I WNIOSKI KOŃCOWE

Tekst do napisania w terminie późniejszym.

WYKAZ LITERATURY

[1] Galois, Inc. and Immunant, Inc. *C2Rust Manual*. https://c2rust.com/manual. 2023.

[2] The Rust Programming Language. *The bindgen User Guide*. https://rust-lang.github.io/rust-bindgen. 2025.

[3] J. Sharp. *Corrode: C to Rust Translator*. https://github.com/jameysharp/corrode. 2021.

[4] Mozilla Foundation. *Quantum/Stylo Project*. https://wiki.mozilla.org/Quantum/Stylo. 2017.

[5] Servo Project. *Servo*. https://github.com/servo/servo. 2025.

[6] Deno Team. *Deno Runtime Documentation*. https://docs.deno.com/runtime/. 2024.

[7] Linux Kernel Developers. *Rust in the Linux Kernel – Quick Start*. https://docs.kernel.org/rust/quick-start.html. 2024.

[8] L. Clark. *Inside a super fast CSS engine: Quantum CSS (aka Stylo)*. https://hacks.mozilla.org/2017/08/inside-a-super-fast-css-engine-quantum-css-aka-stylo. 2017.

[9] Wikipedia contributors. *Quantum/Stylo*. https://wiki.mozilla.org/Quantum/Stylo. 2018.

[10] Mozilla Foundation. *Oxidation Initiative*. https://wiki.mozilla.org/Oxidation. 2020.

[11] CXX Project. *CXX: Safe FFI between Rust and C++*. https://cxx.rs/. 2024.

[12] Mozilla Build Team. *Rust in Firefox Build System*. https://firefox-source-docs.mozilla.org/build/buildsystem/rust.html. 2024.

WYKAZ RYSUNKÓW

WYKAZ TABEL