AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ FIZYKI I INFORMATYKI STOSOWANEJ

KATEDRA ODDZIAŁYWAŃ I DETEKCJI CZĄSTEK

# Praca Dyplomowa

Rozbudowa i uaktualnienie systemu GGSS detektora ATLAS TRT

Update and upgrade of the GGSS system for ATLAS TRT detector

Autorzy: **Arkadiusz Kasprzak, Jarosław Cierpich**

Kierunek studiów: **Informatyka Stosowana**

Opiekun pracy: **dr hab. inż. Bartosz Mindur, prof. AGH**

**Kraków, 2021**

## Oświadczenie studenta

Uprzedzony(-a) o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. z 2018 r. poz. 1191 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony(-a) o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. — Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

.....

(czytelny podpis)

Kraków, ?? września 2021

**Tematyka pracy magisterskiej i praktyki dyplomowej Jarosława Cierpicha, studenta drugiego roku studiów drugiego stopnia na kierunku informatyka stosowana, specjalności modelowanie i analiza danych**

Temat pracy magisterskiej: **Rozbudowa i uaktualnienie systemu GGSS detektora ATLAS TRT**

| | |
|---|---|
| Opiekun pracy: | dr hab. inż. Bartosz Mindur, prof. AGH |
| Recenzenci pracy: | |
| Miejsce praktyki dyplomowej: | WFiIS AGH, Kraków |

**Program pracy magisterskiej i praktyki dyplomowej**

1. Omówienie realizacji pracy magisterskiej z opiekunem.
2. Zebranie i opracowanie literatury dotyczącej tematu pracy.
3. Praktyka dyplomowa:
   - udział w Krakow Applied Physics and Computer Science Summer School '20
   - zapoznanie z materiałami (wykłady i szkolenia praktyczne) obejmującymi zagadnienia z dziedziny fizyki cząstek, informatyki oraz detektorów i elektroniki
   - praca nad projektem GGSS w dwuosobowym zespole, obejmująca zmiany w oprogramowaniu i architekturze projektu
   - prezentacja rezultatów wykonanej pracy przed uczestnikami oraz opiekunami szkoły
   - prezentacja wykonanych prac podczas wydarzenia TRT Days
4. Kontynuacja prac nad projektem:
   - wykonanie dalszych zmian w oprogramowaniu systemu GGSS, w tym dodanie nowych funkcjonalności
   - przeprowadzanie okresowych testów działania systemu w środowisku docelowym
   - wykonanie prac nad infrastrukturą projektu
5. Opracowanie redakcyjne pracy.

Termin oddania w dziekanacie: ?? września 2021

..... .....

(podpis kierownika katedry) (podpis opiekuna)

Kraków, ?? września 2021

**Tematyka pracy magisterskiej i praktyki dyplomowej Arkadiusza Kasprzaka, studenta drugiego roku studiów drugiego stopnia na kierunku informatyka stosowana, specjalności modelowanie i analiza danych**

Temat pracy magisterskiej: **Rozbudowa i uaktualnienie systemu GGSS detektora ATLAS TRT**

| | |
|---|---|
| Opiekun pracy: | dr hab. inż. Bartosz Mindur, prof. AGH |
| Recenzenci pracy: | |
| Miejsce praktyki dyplomowej: | WFiIS AGH, Kraków |

**Program pracy magisterskiej i praktyki dyplomowej**

1. Omówienie realizacji pracy magisterskiej z opiekunem.
2. Zebranie i opracowanie literatury dotyczącej tematu pracy.
3. Praktyka dyplomowa:
   - udział w Krakow Applied Physics and Computer Science Summer School '20
   - zapoznanie z materiałami (wykłady i szkolenia praktyczne) obejmującymi zagadnienia z dziedziny fizyki cząstek, informatyki oraz detektorów i elektroniki
   - praca nad projektem GGSS w dwuosobowym zespole, obejmująca zmiany w oprogramowaniu i architekturze projektu
   - prezentacja rezultatów wykonanej pracy przed uczestnikami oraz opiekunami szkoły
   - prezentacja wykonanych prac podczas wydarzenia TRT Days
4. Kontynuacja prac nad projektem:
   - wykonanie dalszych zmian w oprogramowaniu systemu GGSS, w tym dodanie nowych funkcjonalności
   - przeprowadzanie okresowych testów działania systemu w środowisku docelowym
   - wykonanie prac nad infrastrukturą projektu
5. Opracowanie redakcyjne pracy.

Termin oddania w dziekanacie: ?? września 2021

..... .....

(podpis kierownika katedry) (podpis opiekuna)

# Spis treści

**1. Wstęp** ... 11
1.1. Wprowadzenie do systemu GGSS ... 11
1.2. Cel pracy ... 11
**2. Budowa i działanie systemu GGSS** ... 13
2.1. Wysokopoziomowa architektura systemu GGSS ... 13
2.2. Urządzenia elektroniczne ... 14
2.3. Warstwa oprogramowania ... 15
2.3.1. Aplikacje wchodzące w skład GGSS ... 15
2.3.2. Język C++ ... 15
2.3.3. Biblioteki zewnętrzne ... 16
2.3.4. Infrastruktura projektu ... 16
2.4. Oprogramowanie WinCC OA ... 17
2.5. Środowisko docelowe i ograniczenia ... 18
**3. Praktyki stosowane w projekcie** ... 21
**4. Prace nad architekturą i infrastrukturą projektu** ... 23
4.1. Zmiany w architekturze projektu ... 23
4.1.1. Początkowa architektura projektu ... 23
4.1.2. Uproszczenie architektury projektu ... 26
4.1.3. Dodanie możliwości odtworzenia pierwotnej wersji kodu źródłowego ... 29
4.1.4. Pomniejsze zmiany ... 29
4.1.5. Podsumowanie: ostateczna struktura projektu ... 30
4.2. Automatyzacja pracy z submodułami ... 30
4.3. Rozwój systemu budowania projektu ... 30
4.4. Automatyzacja i centralizacja wersjonowania projektu ... 30
4.5. Pakietowanie i rozlokowanie projektu ... 30
4.6. Rozwój infrastruktury do testowania warstwy sprzętowej ... 30
**5. Prace nad kodem źródłowym projektu** ... 31

**6. Testy systemu** ..... 33

**7. Podsumowanie** ..... 35

# 1. Wstęp

## 1.1. Wprowadzenie do systemu GGSS

Europejska Organizacja Badań Jądrowych CERN jest jednym z najważniejszych ośrodków naukowo-badawczych na świecie i miejscem rozwoju zarówno fizyki, jak i informatyki. Będąc miejscem powstania technologii takich jak protokół HTTP - Hypertext Transfer Protocol, CERN jest kojarzony dziś przede wszystkim z Wielkim Zderzaczem Hadronów (LHC - Large Hadron Collider) - największym akceleratorem cząstek na świecie. Jednym z pracujących przy LHC eksperymentów jest detektor ATLAS (A Toroidal LHC ApparatuS), pełniący kluczową rolę w rozwoju współczesnej fizyki - przyczynił się on do potwierdzenia istnienia tzw. bozonu Higgsa w 2012 roku.

Detektor ATLAS zbudowany jest z kilku pod-detektorów, tworzących strukturę warstową. Najbardziej wewnętrzną część stanowi tzw. Detektor Wewnętrzny (Inner Detector), składający się z kolei z trzech kolejnych podsystemów. Jednym z tychże podsystemów, szczególnie istotnym w kontekście niniejszej pracy, jest detektor promieniowania przejścia (TRT - Transition Radiation Tracker).

System Stabilizacji Wzmocnienia Gazowego (GGSS - Gas Gain Stabilization System) jest jednym z podsystemów detektora TRT, mającym zapewnić jego poprawne działanie. Projekt ten stanowi części systemu kontroli detektora ATLAS (DCS - Detector Control System). W skład systemu GGSS wchodzi zarówno warstwa oprogramowania, jak i szereg urządzeń. Ze względu na jego rolę, jednym z najważniejszych wymagań stawianych przed projektem jest wysoka niezawodność.

W niniejszej pracy autorzy przybliżą najważniejsze zmiany dokonane przez nich w czasie półtorarocznych prac nad rozwojem i usprawnieniem systemu GGSS. Prace obejmują przede wszystkim zmiany w warstwie oprogramowania, mające na celu zarówno wprowadzenia nowych funkcjonalności do systemu, jak również uczynienie go bardziej przystępnym dla korzystających z niego osób, m.in. poprzez automatyzację procesów związanych z cyklem życia oprogramowania (np. tworzenie nowych wydań).

## 1.2. Cel pracy

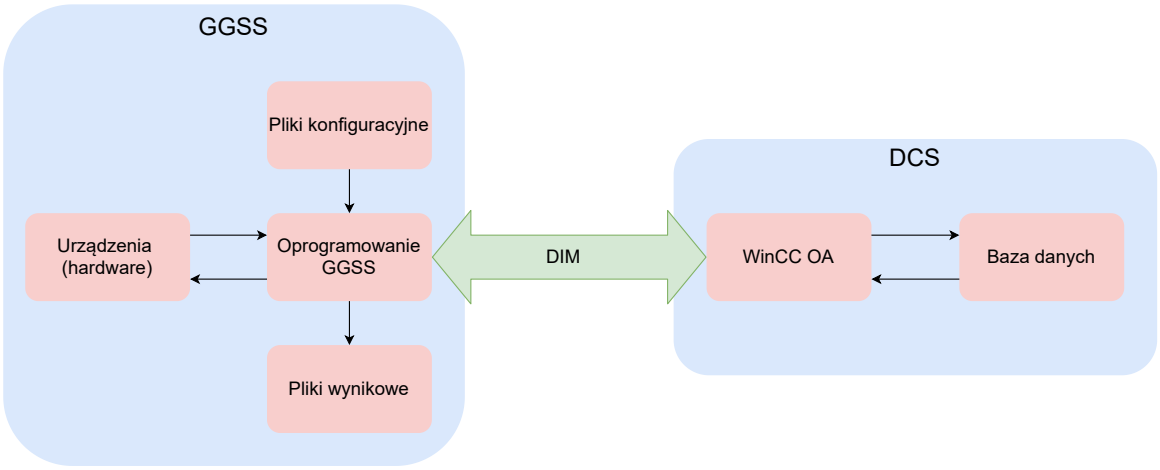
# 2. Budowa i działanie systemu GGSS

Niniejszy rozdział zawiera ważne, z punktu widzenia przeprowadzonych prac, informacje na temat systemu GGSS. Przedstawione tu opisy dotyczą zagadnień takich jak: wysokopoziomowa architektura systemu, struktura warstwy oprogramowania, opis wykorzystywanych przez system urządzeń oraz omówienie cech charakterystycznych środowiska docelowego.

## 2.1. Wysokopoziomowa architektura systemu GGSS

System GGSS składa się z kilku współpracujących ze sobą elementów, przedstawionych (wraz z występującymi między nimi interakcjami) na rysunku 2.1. Znaczenie poszczególnych komponentów projektu jest następujące:

- **oprogramowanie GGSS** - zestaw aplikacji wraz z otaczającą je infrastrukturą, których zadaniem jest sterowanie urządzeniami wchodzącymi w skład systemu GGSS oraz przetwarzanie zbieranych za ich pomocą danych
- **urządzenia (ang. *hardware*)** - zestaw urządzeń elektronicznych (m.in. liczniki słomkowe, zasilacze wysokiego napięcia i multipleksery)
- **pliki konfiguracyjne** - proste pliki tekstowe w formacie XML **rozwin + cytowanie**, zawierające informacje o oczekiwanym sposobie działania systemu (np. maksymalna możliwa wartość napięcia, jakie może zostać ustawione na każdym z zasilaczy)
- **pliki wynikowe** - pliki tekstowe zawierające wyniki pomiarów wykonywanych przez system oraz rejestr zdarzeń
- **system WinCC OA** - **rozwin + cytowanie** - system typu SCADA **rozwin + cytowanie**, stanowiący część systemu kontroli detektora ATLAS (DCS), pozwalający na obserwację i kontrolę działania poszczególnych poddetektorów
- **protokół DIM** - **rozwin + cytowanie** protokół komunikacyjny dla środowisk rozproszonych, oparty o architekturę klient-serwer, zapewniający komunikację między oprogramowaniem systemu GGSS a systemem WinCC OA




**Rys. 2.1.** Wysokopoziomowa architektura projektu GGSS. Strzałkami oznaczono przepływ danych pomiędzy poszczególnymi komponentami systemu.

Szczegóły działania najważniejszych z punktu widzenia niniejszej pracy elementów systemu omówione zostaną w dalszej części tego rozdziału. Znaczna część prac opisanych w niniejszym manuskrypcie skupiona była na udoskonaleniu warstwy oprogramowania systemu GGSS.

## 2.2. Urządzenia elektroniczne

Z punktu widzenia warstwy sprzętowej system GGSS składa się z zestawu tzw. słomkowych liczników proporcjonalnych, zasilanych za pomocą zasilaczy wysokiego napięcia. Sygnały generowane przez liczniki przetwarzane są przez wielokanałowy analizator amplitudy (MCA - **rozwinac**), natomiast wybór licznika słomkowego używanego do wykonania pomiarów następuje za pomocą multipleksera analogowego **cytowanie**. Urządzenia podłączone są do komputera PC, który steruje nimi za pomocą oprogramowania systemu GGSS. W tabeli 2.1 zamieszczone zostało zestawienie informacji na temat wykorzystywanych przez projekt urządzeń. Sposób działania systemu (jego podstawa fizyczna oraz znaczenie przeprowadzanych pomiarów) wykracza poza zakres niniejszej pracy, został natomiast szczegółowo opisany w pracy *Wybrane zagadnienia związane z pracą słomkowych liczników proporcjonalnych w detektorze TRT eksperymentu ATLAS*, której autorem jest dr hab. inż. Bartosz Mindur, prof. AGH **cytowanie**.

**Tabela 2.1.** Zestawienie istotnych z punktu widzenia niniejszej pracy urządzeń wchodzących w skład systemu GGSS.

| Urządzenie | Informacje |
|---|---|
| 4-kanałowy zasilacz wysokiego napięcia | CAEN N1470 |
| wielokanałowy analizator amplitudy | CAEN N957 |
| multiplekser sygnałów analogowych | urządzenie autorstwa Pana Pawła Zadrożniaka |

## 2.3. Warstwa oprogramowania

Poprzez warstwę oprogramowania systemu GGSS autorzy rozumieją zarówno zestaw aplikacji napisanych w języku C++ (standard 11), jak i otaczającą je infrastrukturę (pomocnicze skrypty, system budowania, testowania i tworzenia nowych wydań).

### 2.3.1. Aplikacje wchodzące w skład GGSS

Trzon warstwy oprogramowania projektu GGSS stanowi aplikacja *ggss-runner*, zawierająca logikę odpowiedzialną za komunikację z systemem za pomocą protokołu DIM, gromadzenie i walidację danych oraz sterowanie urządzeniami wchodzącymi w skład warstwy sprzętowej. W skład systemu wchodzi ponadto szereg pomniejszych aplikacji (niektóre z nich stanowią element dodany przez autorów niniejszej pracy, zostaną więc omówione ze szczegółami w dalszych jej częściach):

- *ggss-spector* - aplikacja okienkowa służąca do wizualizacji zebranych przez system danych (zapisanych w plikach wynikowych)
- *ggss-reader* - niezależna aplikacja przeznaczona do wykorzystywania na maszynach deweloperskich, pozwalająca na odtwarzanie działania oprogramowania sterującego GGSS, tzn. wysyłająca do systemu kontroli detektora archiwalne dane z pominięciem warstwy sprzętowej **cytowanie pracy grzeska**
- *ggss-dim-cs* - aplikacja pozwalająca na wysyłanie do systemu komend za pomocą protokołu DIM **tutaj cytowanie**
- zestaw aplikacji *ggss-hardware-service-apps* - proste narzędzia pozwalające na wykonywanie operacji na wchodzących w skład systemu urządzaniach, w tym na wykonywanie testów ich działania.

### 2.3.2. Język C++

Zarówno aplikacja *ggss-runner*, jak i wszystkie aplikacje pomocnicze, napisane zostały za pomocą języka C++. Jest to wydajny, wszechstronny język programowania ogólnego przezna-

czenia, pozwalający programiście zarówno na wykorzystywanie wysokopoziomowych abstrakcji (programowanie obiektowe, generyczne i funkcyjne), jak i na wydajne wykonywanie niskopoziomowych operacji. W ciągu ostatnich dziesięciu lat język ten był intensywnie rozwijany - od 2011 roku pojawiły się cztery nowe standardy, w tym najnowszy w roku 2020, a kolejny przewidziany jest na rok 2023. Zmiany wprowadzane w nowych wydaniach języka mają na celu zarówno dodawanie do niego nowych funkcjonalności, jak również promowanie praktych pozwalających tworzyć prosty w utrzymaniu, czytelny kod. Niestety z uwagi na ograniczenia wynikające ze cech środowiska docelowego, w jakim działać ma system GGSS, w omawianym projekcie możliwe było wykorzystanie jedynie standardu C++11.

### 2.3.3. Biblioteki zewnętrzne

W projekcie wykorzystywane są ponadto biblioteki nie będące częścią standardu języka C++, dostarczające funkcjonalności niezbędnych do poprawnego działania systemu. Najważniejsze z nich to:

- *Boost* **cytowanie** - rozbudowany zestaw bibliotek dla języka C++, cieszący się znaczącą popularnością, m.in. ze względu na wysoką jakość i szeroki zakres wprowadzanych funkcjonalności (m.in. przetwarzanie argumentów linii poleceń, implementacja operacji na grafach, wsparcie dla programowania sieciowego, tworzenia testów jednostkowych czy zaawansowanego metaprogramowania). Ponadto niektóre z bibliotek wchodzących w skład *Boost* były podstawą do implementacji funkcjonalności takich jak inteligentne wskaźniki (ang. smart pointers) czy obsługa wyrażeń regularnych w nowych standardach języka C++.
- *GNU Scientific Library (GSL)* **cytowanie** - biblioteka dla języków C i C++, dostarczająca implementacje popularnych algorytmów numerycznych
- *Qt* oraz *Qwt* **cytowanie** - wieloplatformowy zestaw bibliotek i narzędzi pozwalających na tworzenie aplikacji okienkowych, wykorzystywany przez aplikacje *ggss-spector* oraz *ggss-reader*
- *DIM* **cytowanie** - dostarczona przez CERN biblioteka umożliwiająca wykorzystywanie protokołu DIM
- *CAEN-N957* **cytowanie** - dostarczona przez firmę CAEN biblioteka współdzielona napisana w języku C, służąca do obsługi analizatora wielokanałowego N957

### 2.3.4. Infrastruktura projektu

Projekt GGSS charakteryzuje się rozbudowaną infrastrukturą, w której skład wchodzą systemy odpowiedzialne za budowanie projektu, zarządzanie zależnościami pomiędzy jego komponentami (również wewnątrz samego projektu), automatyzację procesu testowania poszczególnych komponentów oraz automatyzację tworzenia i wersjonowania wydań. Ponadto w jej skład wchodzą skrypty pomocnicze (napisane w językach Bash **cytat** i Python **cytat**), pozwalające

na zarządzanie systemem w jego środowisku docelowym. Gruntowna przebudowa infrastruktury systemu GGSS stanowiła temat pracy inżynierskiej autorów (**cytat i tytul**). W dalszej części niniejszego manuskryptu omówione zostaną wprowadzone w ramach pracy magistersiej rozszerzenia.

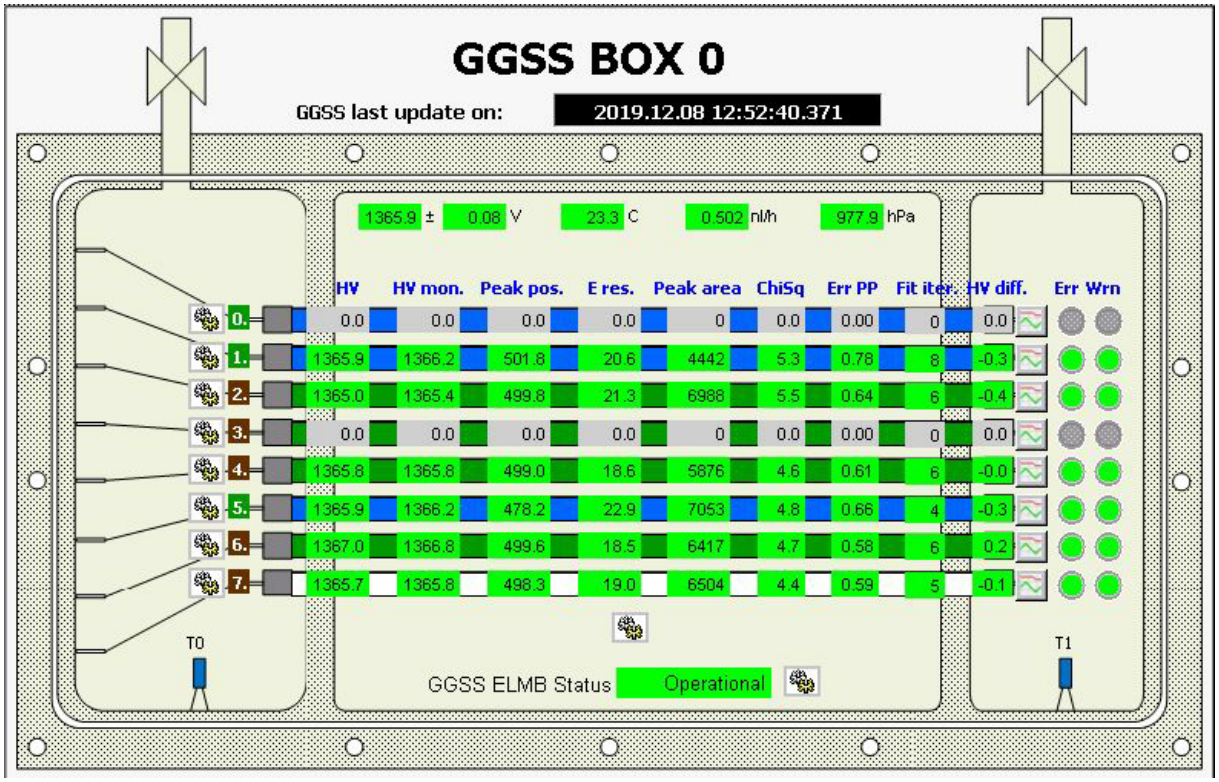
Infrastruktura budowania projektu oparta jest o narzędzie CMake (**cytowanie**) (wersja 2.8) rozwijane przez firmę *Kitware*. Zastosowanie go pozwala na generowanie pliku budującego projekt właściwego dla danej platformy docelowej (np. *Makefile* dla systemów z rodziny UNIX), a co za tym idzie, pozwala na łatwe tworzenie aplikacji wieloplatformowych. Stosując narzędzie CMake, tworzenie systemu budowania projektu polega na przygotowaniu pliku (lub zestawu plików) *CMakeLists.txt*, zawierającego polecenia pozwalające na określenie przez programistę informacji takich jak: standard języka C++, lokalizacja plików źródłowych i bibliotek zewnętrznych.

## 2.4. Oprogramowanie WinCC OA

SIMATIC WinCC Open Architecture jest oprogramowaniem typu SCADA firmy SIEMENS służącym do wizualizacji i sterowania procesami produkcyjnymi. System ten stanowi trzon systemu kontroli detektora ATLAS (DCS) i pozwala na monitorowanie i sterowanie pracą wchodzących w jego skład podsystemów. WinCC OA pozwala m.in. na tworzenie specjalnych paneli, obrazujacych zebrane dane oraz procesy zachodzące w monitorowanym systemie - przykład tego typu panelu, obrazujący pracę liczników słomkowych wykorzystywanych przez GGSS, przedstawiony został na rysunku 2.2.

(https://new.siemens.com/global/en/products/automation/industry-software/automation-software/scada/simatic-wincc-oa/wincc-oa-basic-software.html)

W ramach pracy magisterskiej autorzy nie wykonywali żadnych prac związanych z rozwojem czy utrzymaniem oprogramowania WinCC OA. Jest ono natomiast szczególnie istotne z punktu widzenia testów, jakim musiał być poddawany system GGSS wraz z postępami prac. **dopisac dlaczego, napisac cos jeszcze moze np o pracy grzeska**




**Rys. 2.2**

## 2.5. Środowisko docelowe i ograniczenia

Charakterystyka środowiska docelowego, w jakim działa system GGSS, jest z punktu widzenia niniejszej pracy bardzo istotna - silny związek projektu z infrastrukturą dostarczaną przez CERN stawia przed autorami niniejszej pracy szereg ograniczeń dotyczących wersji wykorzystywanych narzędzi, jak również wymusza dodatkowe działania w przypadku wykonywania pewnych operacji. Do najważniejszych ograniczeń narzucanych przez środowisko docelowe i specyfikę projektu należą:

- dostępna wersje kompilatora języka C++ - w ramach infrastruktury CERN dostępny jest kompilator *g++ (GCC) 4.8.5*. Wersja ta wspiera w większości standard C++11, a zatem funkcjonalności takie jak wyrażenia lambda czy semantyka przenoszenia. Niestety oferowane przez nią wsparcie nie jest pełne - brakuje m.in. poprawnej implementacji biblioteki odpowiedzialnej za przetwarzanie wyrażeń regularnych. Ze względu na wymóg zapewnienia możliwości budowania projektu na maszynie docelowej, ograniczenie to stanowiło znaczące utrudnienie podczas prac nad kodem źródłowym aplikacji wchodzących w skład systemu.
- dostępna wersja narzędzia CMake - na maszynach docelowych dostępna jest wersja *2.8.12.2*, stanowiąca bardzo stare wydanie narzędzia. **dopisac cos**
- związek projektu z wersją jądra systemu **dopisac cos**

– możliwość przeprowadzania testów tylko w określonych momentach prac nad projektem **dopisac cos**
– ograniczone uprawnienia w środowisku docelowym **dopisac cos**
– konieczność zachowania kompatybilności wstecznej **dopisac cos**

# 3. Praktyki stosowane w projekcie

# 4. Prace nad architekturą i infrastrukturą projektu

Niniejszy rozdział zawiera opis prac wykonanych przez autorów w ramach rozwoju architektury i infrastruktury systemu GGSS. Rozdział ten stanowi bezpośrednią kontynuację pracy inżynierskiej autorów, gdzie przygotowane zostały pierwsze wersje rozwijanych w ramach pracy magisterskiej rozwiązań. Przedstawione tu informację dotyczą szerokiego zakresu zagadnień związanych z inżynierią oprogramowania, takich jak: zarządzanie strukturą projektu oraz jego zależnościami, automatyzacja procesów towarzyszących wytwarzaniu oprogramowania czy przygotowanie infrastruktury ułatwiającej testy warstwy sprzętowej systemu.

## 4.1. Zmiany w architekturze projektu

Przez zmiany w architekturze projektu autorzy rozumieją stopniowy rozwój rozwiązania przygotowanego w ramach napisanej przez nich pracy inżynierskiej. Wprowadzone po jej zakończeniu modyfikacje to przede wszystkim uproszczenie powstałej hierarchii zależności między poszczególnymi elementami warstwy oprogramowania (rozumianymi zarówno jako repozytoria, jak i biblioteki), uczynienie systemu bardziej przystępnym dla użytkownika (np. poprzez nadanie komponentom nazw dobrze oddających ich przeznaczenie) oraz przygotowanie systemu pozwalającego w prosty sposób odtworzyć kod źródłowy w wersji bez wprowadzonych w ramach pracy magisterskiej modyfikacji (jako rodzaj zabezpieczenia przed skutkami potencjalnych błędów, które mogły zostać wprowadzone do oprogramowania podczas prac nad nim). Znaczna część zmian opisanych w niniejszej części pracy była możliwa do wprowadzenia z uwagi na trwające jednocześnie prace nad kodem źródłowym systemu GGSS i zmiany zachodzące w ich czasie.

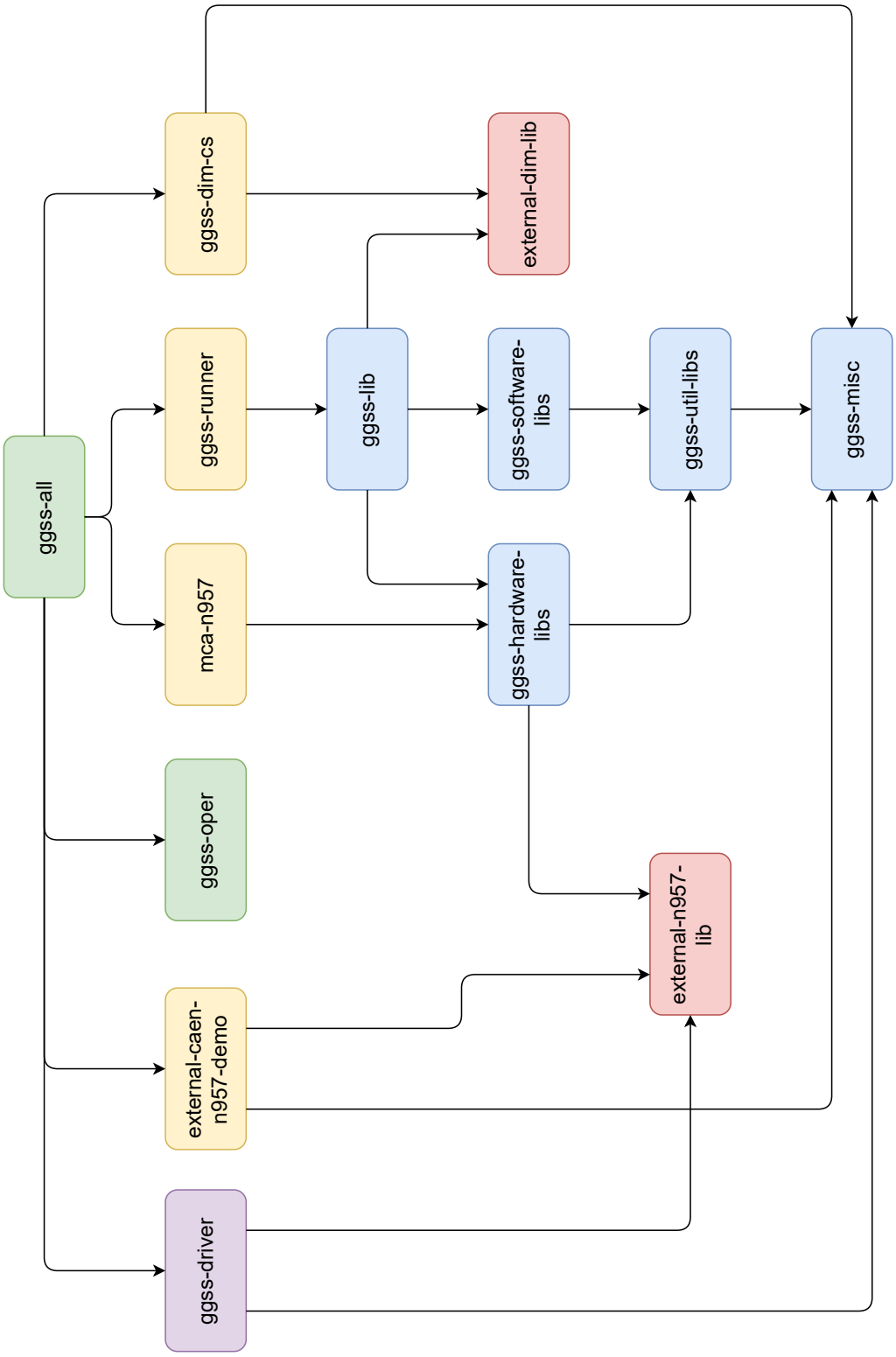
### 4.1.1. Początkowa architektura projektu

Przeprowadzone przez autorów w ramach pracy inżynierskiej modyfikacje architektury systemu GGSS obejmowały przede wszystkim migrację projektu do systemu kontroli wersji Git, wprowadzenie spójnego nazewnictwa poszczególnych komponentów oraz zastosowanie funkcjonalności submodułów będącej częścią technologii Git do stworzenia hierarchicznej struktury repozytoriów (w odróżnieniu od pierwotnej, płaskiej architektury opartej o katalogi). Celem

tych zmian było ułatwienie pracy nad pojedynczymi komponentami projektu oraz uczynienie struktury projektu przyjazną dla użytkownika, co zostało zdaniem autorów osiągnięte.

Architektura stanowiąca punkt wyjściowy zmian wykonanych w ramach niniejszej pracy przedstawiona została na rysunku 4.1 (z pominięciem repozytoriów pomocniczych, zawierających np. dokumentację). Projekt składał się więc pierwotnie z 14 repozytoriów zawierających wchodzące w skład oprogramowania systemu GGSS biblioteki, aplikacje i skrypty. Przygotowane w ten sposób rozwiązanie charakteryzowało się jednak pewnymi wadami i ograniczeniami, z których najważniejsze to:

- głęboka hierarchia zależności, mająca negatywny wpływ na wydajność działania mechanizmu submodułów
- istnienie repozytorium *ggss-misc*, zawierającego (poza szablonami CMake) elementy kodu źródłowego niepasujące do pozostałych bibliotek wchodzących w skład systemu: bazowe klasy wyjątków stosowanych w całym projekcie oraz flagi konfigurujące projekt w zależności od systemu operacyjnego (konieczność zastosowania tego typu zabiegu wynikła wprost z założenia o niemodyfikowaniu kodu źródłowego w czasie tworzenia pracy inżynierskiej)
- zachowanie oryginalnych nazw bibliotek i aplikacji, dostosowując je jedynie do przyjętej konwencji. Jedną z bibliotek wchodzących w skład projektu była biblioteka statyczna *handle-lib*, odpowiedzialna za implementację mechanizmu slotów i sygnałów (**pisownia**), na co, zdaniem autorów, jej nazwa nie wskazuje.
- wnioskowanie o zależnościach pomiędzy bibliotekami na podstawie dyrektyw preprocesora *include* zawartych w kodzie źródłowym, a nie wykorzystywanych funkcjonalności, co wynikało z niewielkiego doświadczenia i wiedzy autorów na temat systemu podczas tworzenia pracy inżynierskiej oraz wspomnianego już założenia o niemodyfikowaniu kodu źródłowego.
- założenie o tworzeniu oddzielnego repozytorium dla każdej z występujących w projekcie aplikacji, niezależnie od jej rozmiarów, co ostatecznie znacznie skomplikowało powiązania pomiędzy repozytoriami (np. repozytoria *external-caen-n957-demo* oraz *mca-n957* charakteryzują się podobnymi zależnościami i oba zawierają niewielkie aplikacje, których zadaniem jest współpraca z wielokanałowym analizatorem amplitudy CAEN N957 - mogłoby być więc połączone w jedno repozytorium).
- brak łatwego sposobu na odtworzenie pierwotnej postaci kodu źródłowego - mechanizm ten nie był potrzebny na etapie pracy inżynierskiej, ponieważ nie dokonywano wtedy modyfikacji we wspomnianym kodzie.


ggss-all
ggss-dim-cs
ggss-runner
mca-n957
ggss-oper
external-caen-n957-demo
ggss-driver
ggss-lib
ggss-hardware-libs
external-dim-lib
ggss-software-libs
ggss-util-libs
ggss-misc
external-n957-lib

Rys. 4.1

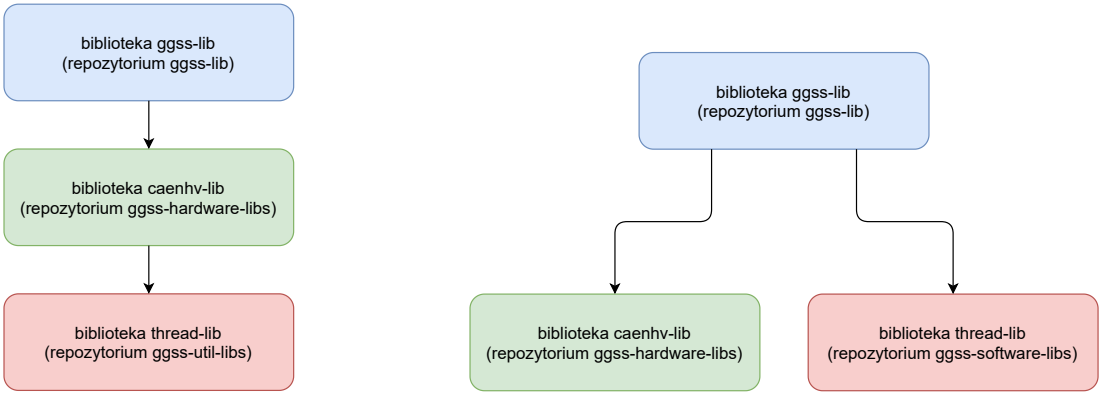
### 4.1.2. Uproszczenie architektury projektu

Pierwszym podjętym przez autorów działaniem mającym na celu modyfikację struktury projektu była próba jej uproszczenia poprzez analizę zależności wewnętrzych systemu (tzn. zależności pomiędzy poszczególnymi bibliotekami). Należy tutaj zwrócić uwagę, że prac tych nie wykonywano przez pierwsze pół roku od rozpoczęcia przez autorów studiów magisterskich - skupienie się na użytkowaniu stworzonej architektury pozwoliło autorom zarówno na zapoznanie się lepiej z projektem, jak również na samodzielną obserwację jej zalet i wad.

W celu zrozumienia wprowadzonych w projekcie zmian, konieczne jest zrozumienie toku rozumowania, jakim posługiwali się autorzy pracy podczas tworzenia oryginalnej struktury projektu - dotyczy to przede wszystkim repozytoriów *ggss-lib*, *ggss-software-libs*, *ggss-hardware-libs*, *ggss-util-libs* oraz *ggss-misc*. Ich rola w projekcie prezentuje się następująco:

- *ggss-hardware-libs* - przechowywanie bibliotek odpowiedzialnych za obsługę urządzeń wchodzących w skład warstwy sprzętowej systemu GGSS. W pierwotnej wersji projektu były to następujące biblioteki:
  - *caenhv-lib* oraz *caenn1470-lib* - odpowiedzialne za komunikację z zasilaczami wysokiego napięcia CAEN N1470
  - *mca-lib* oraz *ortecmcb-lib* - odpowiedzialne za obsługę wielokanałowego analizatora amplitudy CAEN N957
  - *usbrm-lib* - odpowiedzialna za obsługę multipleksera sygnałów analogowych
- *ggss-software-libs* - przechowywanie bibliotek odpowiedzialnych za implementację wykorzystywanych przez system algorytmów i struktur danych związanych ściśle z warstwą oprogramowania (tzn. nie mających związku z warstwą sprzętową). W pierwotnej wersji projektu były to następujące biblioteki:
  - *xml-lib*
  - *fifo-lib*
  - *fit-lib*
  - *daemon-lib*
- *ggss-util-libs* - przechowywanie bibliotek, od których zależne są zarówno komponenty odpowiedzialne za obsługę warstwy sprzętowej projektu, jak i związane wyłącznie z warstwą oprogramowania. Innymi słowy, są to biblioteki nie mogące znaleźć się w żadnym z dwóch wymienionych powyżej repozytoriów. W pierwotnej wersji projektu były to:
  - *log-lib*
  - *utils-lib*
  - *handle-lib*
  - *thread-lib*
- *ggss-misc* -
- *ggss-lib* - przechowywanie kodu źródłowego zawierającego główną logikę systemu GGSS

Prace nad kodem źródłowym projektu pozwoliły autorom zaobserwować, iż pewna część występujących w nim dyrektyw preprocesora *include* nie oddaje w poprawny sposób faktycznej struktury zależności między bibliotekami. Najważniejszy przykład stanowi łańcuch zależności występujących pomiędzy biblioteką *ggss-lib*, a bibliotekami *caenhv-lib* oraz *thread-lib*. W oryginalnej wersji projektu zależności między wymienionymi komponentami prezentowały się tak, jak na rysunku 4.2a, tzn. biblioteka *ggss-lib* zależna była od biblioteki *caenhv-lib*, która natomiast zawierała dyrektywę *include* dołączającą plik nagłówkowy z biblioteki *thread-lib*.




(a) Test (b) Test

**Rys. 4.2.** Test

W rzeczywistości biblioteka *caenhv-lib* nie wykorzystywała zawartości wspomnianego pliku nagłówkowego - pełniła jedynie formę swego rodzaju pośrednika, udostępniając znajdujące się tam klasy bibliotece *ggss-lib*. Przeniesienie dyrektywy *include* do biblioteki *ggss-lib* spowodowało, iż żadna z bibliotek wchodzących w skład repozytorium *ggss-hardware-libs* nie zawierała zależności do biblioteki *thread-lib*. Rozwiązanie to pozwoliło dokonać migracji tejże biblioteki, wraz z wykorzystywaną przez nią biblioteką *handle-lib*, do repozytorium *ggss-software-libs*, redukując tym samym liczbę bibliotek znajdujących się w repozytorium *ggss-util-libs*. Rysunek 4.2b przedstawia w sposób schematyczny strukturę otrzymanego rozwiązania.

W związku z opisanymi powyżej zmianami ilość kodu źródłowego znajdującego się w repozytorium *ggss-util-libs* znacznie spadła - pozostałe tam biblioteki *log-lib* oraz *utils-lib* charakteryzowały się niewielkim rozmiarem. Spowodowało to, iż jednoczesne istnienie modułów *ggss-misc* oraz *ggss-util-libs* (po wprowadzonych zmianach spełniających tą samą rolę przechowywania niewielkiej liczby komponentów wykorzystywanych przez wiele modułów projektu GGSS) przestało być uzasadnione. Kolejny etap wykonanych prac stanowiło więc przeprowadzenie integracji tychże repozytoriów - w tym celu zdecydowano się na likwidację modułu *ggss-misc* po wcześniejszym przeniesieniu jego zawartości do *ggss-util-libs*.

Migracja znajdujących się w repozytorium *ggss-misc* plików *.cmake* (modułów wykorzystywanych przez infrastrukturę budowania projektu) wymagała, poza wykonaniem trywialnej czynności przeniesienia katalogu, aktualizacji (na poziomie całego projektu) ścieżek wskazujących lokalizację tychże plików. Działanie to było konieczne, ponieważ narzędzie CMake wymaga od programisty, by wyspecyfikował on lokalizację modułów *.cmake* dołączanych do projektu (np. za pomocą komendy `include()`) poprzez dodanie ścieżki z ich lokalizacją do listy `CMAKE_MODULE_PATH` (przykład wykorzystania tejże listy przedstawiony został na listingu **listing**). Oznaczało to więc konieczność wykonania, w każdym module wykorzystującym pliki *.cmake*, zmiany wspomnianej ścieżki tak, by wskazywała na katalog *cmake-templates* w repozytorium *ggss-util-libs*.

Poza wspomnianymi plikami *.cmake* w repozytorium *ggss-misc* znajdował się katalog *include*, zawierający trzy pliki nagłówkowe z kodem napisanym w języku C++:

- pliki `ggssExceptions.h` oraz `HardwareException.h` zawierające klasy bazowe wyjątków wykorzystywanych w całym projekcie GGSS
- plik `CompatibilityFlags.h`, zawierający flagi konfigurujące projekt w zależności od platformy docelowej (Windows lub Linux)

Pliki te nie wchodziły oryginalnie w skład żadnej z bibliotek projektu GGSS, nie mogły zostać do nich również dodane przez autorów podczas przygotowywania pracy inżynierskiej, ponieważ wymagałoby to modyfikacji kodu źródłowego systemu. Podczas przeprowadzanej w ramach niniejszej pracy migracji tych plików do repozytorium *ggss-util-libs* zdecydowano się na likwidację katalogu *include* i rozdysponowanie jego zawartości do istniejących lub nowych bibliotek. Plik `CompatibilityFlags.h` przeniesiony został więc do biblioteki *utils-lib*, natomiast na potrzebę dwóch pozostałych nagłówków przygotowana została nowa biblioteka *exceptions-lib*.

Finalna struktura repozytorium *ggss-util-libs* przedstawiona została na listingu **listing**. Poza wspomnianymi do tej pory zmianami nowość stanowi katalog `doxygen-config`, zawierający prosty plik konfigurujący działanie narzędzia Doxygen **cytat** służącego do generowania dokumentacji programów napisanych w języku C++. Rozszerzenie projektu o możliwość generowania dokumentacji zostanie jednak opisane szczegółowo w dalszej części pracy (sekcja **dodac referencje do sekcji**).

Poza wspomnianymi do tej pory repozytoriami zmianami objęte zostały ponadto moduły przechowujące aplikacje służące do testowania i obsługi urządzeń elektronicznych wchodzących w skład warstwy sprzętowej systemu GGSS. Motywacją do wprowadzenia modyfikacji była konieczność rozbudowy projektu o kolejne tego typu aplikacje (co zostanie szerzej opisane w sekcji **sekcja**) - tworzenie dla każdej z nich osobnego repozytorium znacząco komplikowałoby strukturę projektu. Zdecydowano zatem, iż repozytoria *mca-n957* oraz *external-caen-n957-demo* zostaną dołączone do nowo powstałego repozytorium *ggss-hardware-service-apps*, grupującego niewielkie programy służące do operowania na urządzeniach.

Poza zmniejszeniem progu wejścia do projektu poprzez uczynienie jego struktury prostszą, opisane do tej pory zmiany korzystnie wpłynęły na działanie mechanizmu submodułów systemu Git, na którym oparty został proces zarządzania zależnościami między repozytoriami w projekcie. Redukcja liczby repozytoriów i powiązań między nimi oraz zmniejszenie głębokości drzewa zależności (poprzez likwidację repozytorium *ggss-misc*) miało pozytywny wpływ na wydajność systemu zarządzającego architekturą projektu.

### 4.1.3. Dodanie możliwości odtworzenia pierwotnej wersji kodu źródłowego

### 4.1.4. Pomniejsze zmiany

Poza do tej pory opisanymi, wykonanych zostało kilka pomniejszych modyfikacji mających na celu szeroko pojętą poprawę jakości struktury projektu. Przeprowadzone prace obejmują bogaty zakres wprowadzonych zmian, nie jest więc możliwe zamieszczenie w niniejszej pracy dokładnego opisu każdej z nich. Poniżej krotko opisane zostały więc trzy wybrane przez autorów modyfikacje, charakteryzujące się różnym poziomem skomplikowania, ale operujące na poziomie pojedynczych repozytoriów.

#### 4.1.4.1. Likwidacja repozytorium *ggss-oper*

Jednym z repozytoriów wprowadzonych przez autorów w ramach wykonywania pracy inżynierskiej był moduł *ggss-oper*, zawierający skrypty oraz pliki konfiguracyjne stanowiące znaczną część infrastruktury przeznaczonej do użytkowania wraz z oprogramowaniem GGSS na maszynie docelowej. Zawartość tego repozytorium, nie stanowiąca wkładu wniesionego przez autorów niniejszej pracy w system, obejmowała m.in.:

- pierwsze wersje skryptów służących do przeprowadzania testów urządzeń wchodzących w skład warstwy sprzętowej projektu (napisane z wykorzystaniem języka Python)
- skrypty zarządzające stanem środowiska docelowego (np. ustawiające wymagane zmienne środowiskowe)
- skrypty zarządzające oprogramowaniem systemu GGSS, np. *ggss_monitor.sh* pozwalający na uruchamianie, zatrzymywanie oraz sprawdzanie stanu aplikacji *ggss-runner*

Wraz z postępami prac nad projektem, część z wymienionej powyżej zawartości zastąpiona została przez autorów pracy rozwiązaniami alternatywnymi (np. skrypty służące do przeprowadzania operacji na urządzeniach zastąpione zostały aplikacjami napisanymi w języku C++), pozostałe przeniesione zostały natomiast do repozytorium *ggss-all*. Ostatecznie moduł został więc zlikwidowany.

#### 4.1.4.2. Utworzenie biblioteki *asyncserial-lib*

Podczas prac nad kodem źródłowym bibliotek statycznych wchodzących w skład repozytorium *ggss-hardware-libs* zaobserwowano, że w katalogach bibliotek *usbrm-lib* oraz *caenn1470-lib* zamieszczony został, poza właściwym dla nich kodem źródłowym, zestaw plików zawierających

implementację asynchronicznej komunikacji z urządzeniami za pomocą interfejsu szeregowego. Ponieważ znalezione w obu przypadkach pliki nie różniły się od siebie, i jednocześnie stanowiły niezbędny element wspomnianych komponentów systemu (zawierały kluczową dla działania projektu funkcjonalność), zdecydowano o utworzeniu nowej biblioteki zawierającej omawiane pliki. Biblioteka nazwana została, zgodnie ze swoim przeznaczeniem, *asyncserial-lib* i weszła w skład repozytorium *ggss-hardware-libs*.

#### 4.1.4.3. Zmiana nazwy biblioteki *handle-lib*

Jedną z bibliotek będących częścią systemu GGSS była biblioteka *handle-lib*, odpowiedzialna za implementację mechanizmu slotów i sygnałów **napisac moze co to**. Oryginalnie biblioteka ta znajdowała się w repozytorium *ggss-util-libs*, jednak wraz z postępem prac przeniesiona została, wraz z biblioteką *thread-lib*, do repozytorium *ggss-software-libs* (powód tej migracji opisany został w sekcji **podaj sekcje** niniejszej pracy). Nazwa biblioteki nie pozwalała użytkowniki domyślić się, jakie jest jej zastosowanie - z tego powodu zdecydowano się wprowadzić nową nazwę: *sigslot-lib* (od angielskiego *signals and slots*).

### 4.1.5. Podsumowanie: ostateczna struktura projektu

## 4.2. Automatyzacja pracy z submodułami

## 4.3. Rozwój systemu budowania projektu

## 4.4. Automatyzacja i centralizacja wersjonowania projektu

## 4.5. Pakietowanie i rozlokowanie projektu

## 4.6. Rozwój infrastruktury do testowania warstwy sprzętowej

# 5. Prace nad kodem źródłowym projektu

# 6. Testy systemu

# 7. Podsumowanie