Opracował: Bartłomiej Paszkiewicz

**Weryfikacja oprogramowania na modelu turbiny**

1. Zakres projektu

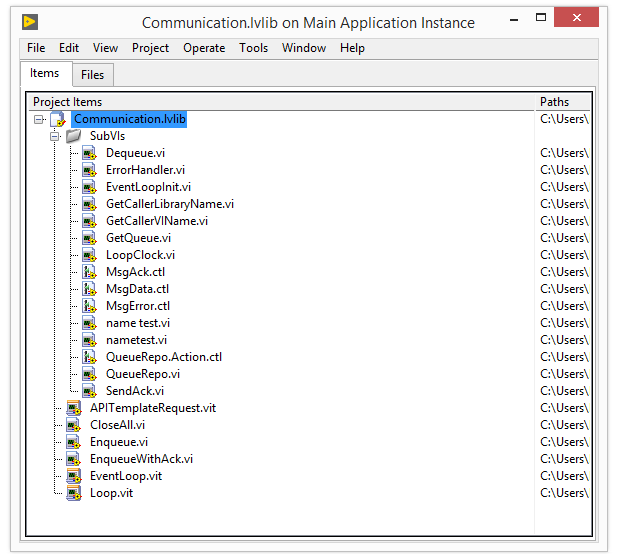
Celem projektu było opracowanie oprogramowania do kontroli i zarządzania pomiarami za pomocą kart DAQ umieszonych w systemie PXI. Realizacja projektu składa się z kilku etapów. Kolejno, konieczne było określenie wymagań, które musi ono spełniać zarówno w obecnej fazie jak i w przyszłości, stworzenie architektury oprogramowania, a także stworzenie części funkcjonalnej odpowiedzialnej za komunikację z warstwą sprzętową oraz interfejs użytkownika.

1. Wymagania

Stworzone oprogramowanie ma być wykorzystane do wykonywania pomiarów wejść analogowych kart DAQ umieszczonych w systemie PXI. Musi umożliwiać synchronizację poszczególnych kart, kontrole częstotliwości pomiarów oraz ich wyzwalania. Sygnał mierzony pochodzi z czujników magnetycznych rejestrujących położenia łopatek na modelu turbiny. Zmierzone dane są następnie przetwarzane w celu określania położeń łopatek i zapisywane do pliku. Zagadnieniem wymagającym dalszej analizy jest ilość danych generowanych przez system podczas pracy i konieczny poziom optymalizacji kodu w celu wydajnego ich przetwarzania. Dane prezentujące położenia łopatek, a także mierzone sygnały są prezentowane użytkownikowi na wykresach. Kluczowe w budowanej aplikacji, jest stworzenie architektury, która umożliwi sprawne dodawanie i testowanie nowych funkcjonalności.

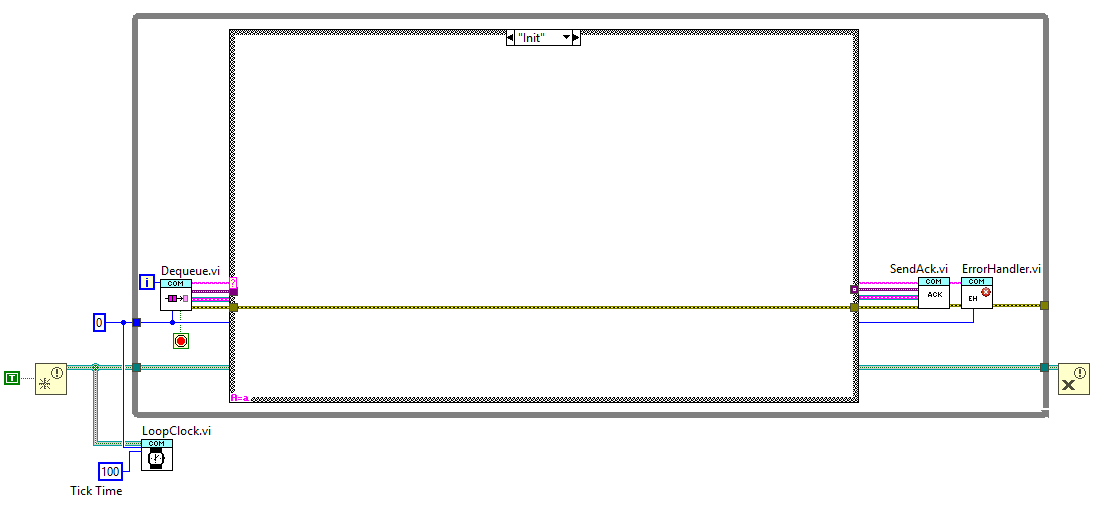
1. Architektura opracowanej aplikacji

Ze względu na specyfikę projektu dotycząca stworzenia aplikacji średniej wielkości łatwej do rozbudowy i utrzymywania, żaden z dostępnych natywnie w LabVIEW wzorców projektów nie spełniał wszystkich wymagań. Wzorce architektury aplikacji QMH (Queued Message Handler) oraz maszyny stanów są zbyt proste i nie umożliwiają budowy aplikacji, w których można uruchomić wiele równoległych procesów w oparciu o ten sam kod. Jednocześnie dostępny natywnie wzorzec wykorzystujący aktory (Actor Framework) jest przez swojej skomplikowanie trudny do konserwacji oraz rozbudowy. Dlatego postanowiono opracować własną architekturę na potrzeby tej aplikacji z myślą o stworzeniu jej w takiej postaci, aby możliwe było jej wykorzystanie, jako ogólne wzorca tworzenia aplikacji we wszystkich następnych projektach tworzonych w organizacji. Cechuje się ona prostotą oraz wydajnością działania oraz szybkością tworzenia kodu funkcjonalnego do poszczególnych modułów.

Wszystkie procesy oraz komunikacja pomiędzy nimi odbywa się z wykorzystaniem biblioteki Communication.lvlib. Zawiera ona w sobie wszystkich subVI odpowiedzialne za wysyłanie i odbieranie wiadomości pomiędzy procesami oraz wzorce (w postaci plików VI Template) służce do pisania własnych modułów oraz ich API. Na rysunku poniżej pokazano zawartość biblioteki Communication.lvlib:

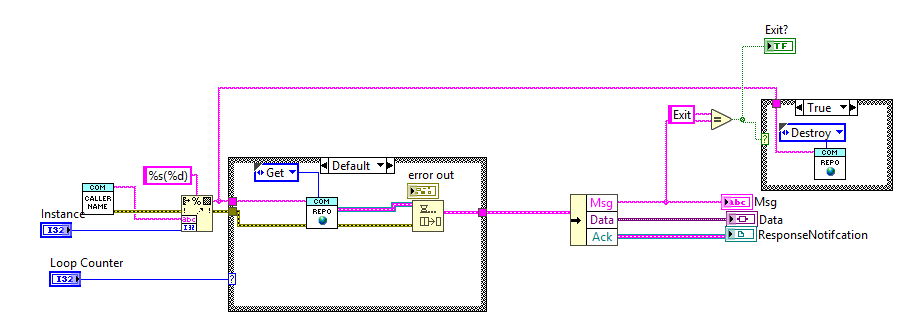
Rys Zawartość biblioteki Communication.lvlib

Biblioteka przewiduje dwóch rodzajów procesów: Interfejsu użytkownika (GUI) oraz przetwarzania danych (QMH). Każdy z procesów, niezależnie od swojego rodzaju musi mieć zaimplementowaną obsługę trzech wiadomości: inicjalizującej (wykonywana jest przed jakąkolwiek inną), zamykającej (na samym końcu) oraz obsługi błędu (w jakimkolwiek innym zdarzeniu). Dodatkową wiadomością, obsługiwaną przez proces przetwarzania danych jest „Tick”, który jest automatycznie wysyłany, co określoną z góry ilość czasu (typowo 100ms). Służy one do taktowana operacji wykonywanych regularnie. Na rysunku poniżej przedstawiono wzorcowy moduł pętli przetwarzającej wiadomości:



Rys 2. Wzorzec pętli przetwarzającej wiadomości (QMH)

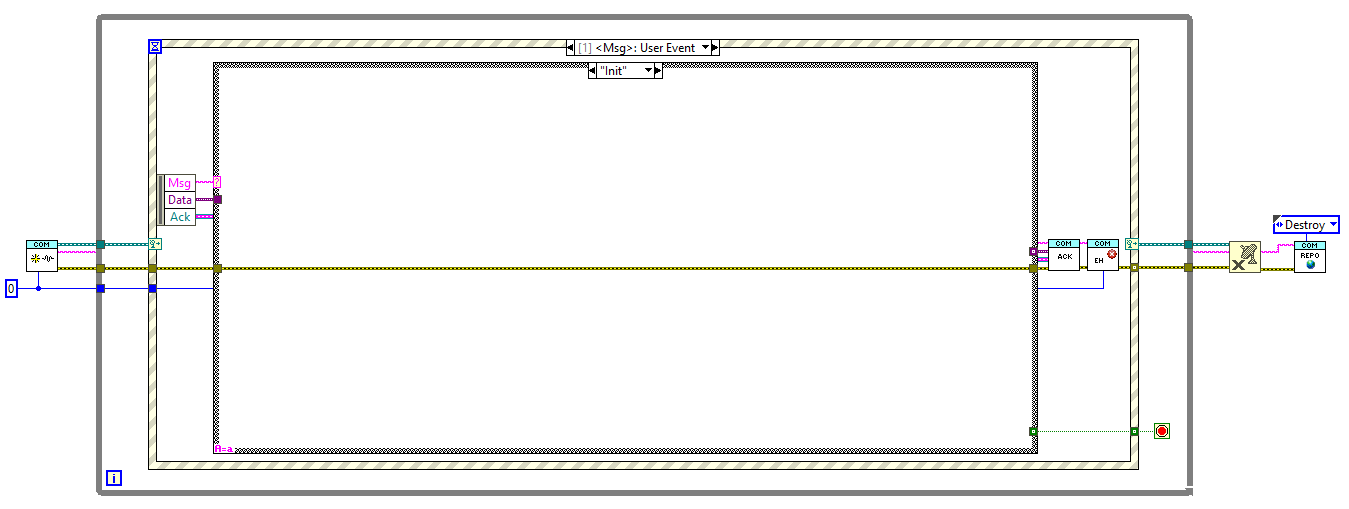
Jej podstawową składową jest pętla while z trzema funkcjami: dequeue.vi która odpowiada za odbierania wiadomości, SendAck .vi, które odpowiada za wysyłanie ewentualnego potwierdzenia do modułu nadającego oraz ErrorHandler.vi, który odpowiada za obsługę błędów. Funkcja Dequeue.vi obserwuje kolejkę zdarzeń, przesyłających do danego procesu trzy-elementowe klastry zawierające: polecenie (String), dane (Variant) oraz referencję do powiadomienia (Notifier Reference). Kolejki do wszystkich utworzonych modułów przechowywane są w „Queue Repo.vi”, dodawane są automatycznie, podczas pierwszego wywołania f-ji dequeu.vi. Nazwy kolejek są tworzone od nazwy biblioteki, w którym jest utworzony proces pętli. Kolejne instancję pętli pochodzących z tych samych bibliotek, są rozróżniane za pomocą nadawanego przed uruchomieniem pętli numeru instancji. Na rysunku poniżej pokazana kod funkcji Deque.vi:



Rys 3. F-cja Dequeu.vi

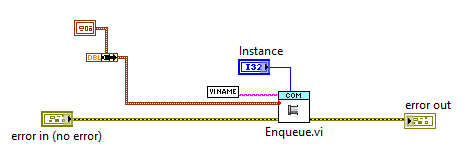
Odebrana wiadomość jest rozbijana na poszczególne element składowe i przekazywana do struktury case, która odpowiada za jej przetworzenie. Są dwa rodzaje wiadomości, pierwszy rodzaj jest wysyłany do danego modułu bez czekania na potwierdzenie wykonania, drugi rodzaj jest przekazywany do modułu oraz f-cja wysyłająca czeka na potwierdzenie wykonania wiadomości. Rozróżnienie rodzaju wysłanej wiadomości następuje w f-cji SendAck.vi. Sprawdza ona czy referencja do powiadomienia wchodząca z skład każdej wiadomości jest różna od zera. Jeżeli tak, to dane w postaci Variant i klastra błędów są wysyłane do pętli, która wysłała daną wiadomość. Ostatnią operacją wykonywaną przy każdym wywołaniu, jest obsługa błędów. Schemat ich obsługi zakłada wysyłanie każdego błędu, który został wygenerowany podczas przetwarzania do stanu Error, razem z informacją o wiadomości, która spowodowała jego wystąpienie. W stanie Error, podejmowana jest decyzja o procedurze obsługi (pominięciu, zalogowaniu, resecie modułu itd.).

Analogicznie do modułu przetwarzania danych działa moduł interfejsu użytkownika, z różnicą dodania do pętli while, struktury obsługi zdarzeń interfejsu użytkownika. Pozwala ona na przechwytywanie operacji użytkownika. W celu zapewnienia komunikacji z innymi procesami aplikacji, struktura zdarzeń obsługuje również zdarzenie wysyłane programowo, w którym przesyłany jest taki sam klaster jak przy pętlach QMH. Poniżej pokazano wzorzec pętli interfejsu użytkownika:



Rys 4. Pętla interfejsu użytkownika

Do komunikacji pomiędzy procesami wykorzystywane są f-cjie kolejkujące. Wywołania API, wszystkie korzystają z funkcji Enqueue.vi lubEnqueueWithAck.vi, która kolejkuje konkretne wiadomości. Template na podstawie tworzone są wywołania umieszczono na rysunku poniżej:



Rys 5. Przykładowa f-cja kolejkująca

Właściwa kolejka, która powinna być użyta jest rozpoznawana po nazwie biblioteki, w której f-cja jest umieszona. Wiadomość, która jest zakolejkowana jest rozpoznawana po nazwie VI. Dzięki temu wymuszone jest logiczne ułożenie kodu. Każdy proces razem ze swoim API jest umieszony w oddzielnej bibliotece, spełnia to jedną z podstawowych zasad programowania obiektowego- enkapsulacje. Dodatkowo w f-cjach Enqueue.vi oraz EnqueueWithAck umieszczono mechanizmy odpowiadającą za kolejkowanie wiadomości do modułów QMH oraz do modułów UI. Globalna zmienna funkcyjna QueueRepository.VI przechowuje zarówno referencję do poszczególnych kolejek jak i do zarejestrowanych zdarzeń interfejsu użytkownika. Dzięki temu, wszystkie moduły wykorzystane w programie mają jednolity interfejs komunikacji.

1. Stworzona aplikacja

Stworzona aplikacja składa się z następujących modułów:

1. MainGUI

Główny proces odpowiedzialny za interfejs użytkownika, w celu uzyskania modułowej struktury główną częścią Front Panelu jest kontener SubVI, który umożliwia umieszczanie w środku Front Paneli innych procesów. Dzięki temu możliwe jest rozdzielenie interfejsu użytkownika na poszczególne funkcjonalności