Gliwice, 11.06.2010

Programowanie Komputerów

Temat: *Przeglądarka Logów w Czasie Rzeczywistym*

LogLiteViewer.CLI

Autor aplikacji i sprawozdania:

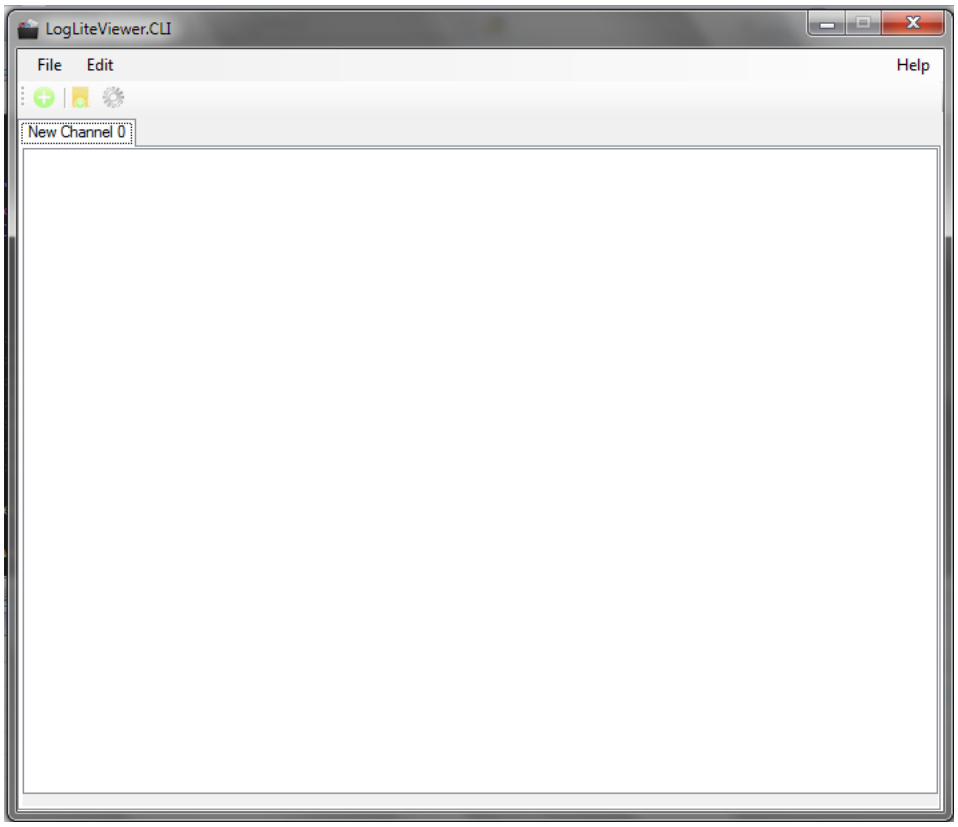
Gawroński Wojciech, *Inf4, gr. 5 sekcja 1*

1. **Analiza tematu.**
    - **Decyzje dotyczące wymagań.**
        - Aplikacja i *GUI* powinna być bardzo lekka.
        - Możliwość obserwacji plików *XML* i *TXT* w konkretnym katalogu.
        - Obsługa danych wysyłanych protokołem *UDP* na konkretnym porcie.
        - Możliwość łączenia wielu wejść w jeden kanał.
        - Obsługa wielu kanałów w jednej aplikacji.

    - **Decyzje dotyczące środowiska pracy.**
        - Zastosowanie języka *C++/CLI*, aby jak najmniejszym kosztem utworzyć lekkie i wydajne GUI (Windows Forms).
        - Użycie *Microsoft Visual Studio 2008*, jako *IDE*.
        - Zastosowanie rozproszonego systemu kontroli wersji (*Mercurial*).
        - Użycie pewnych klas platformy *Microsoft .NET 3.5*.

2. **Specyfikacja wewnętrzna.**
    - **Ogólny opis działania.**

        Po wejściu aplikacja ukazuje poniższy interfejs:



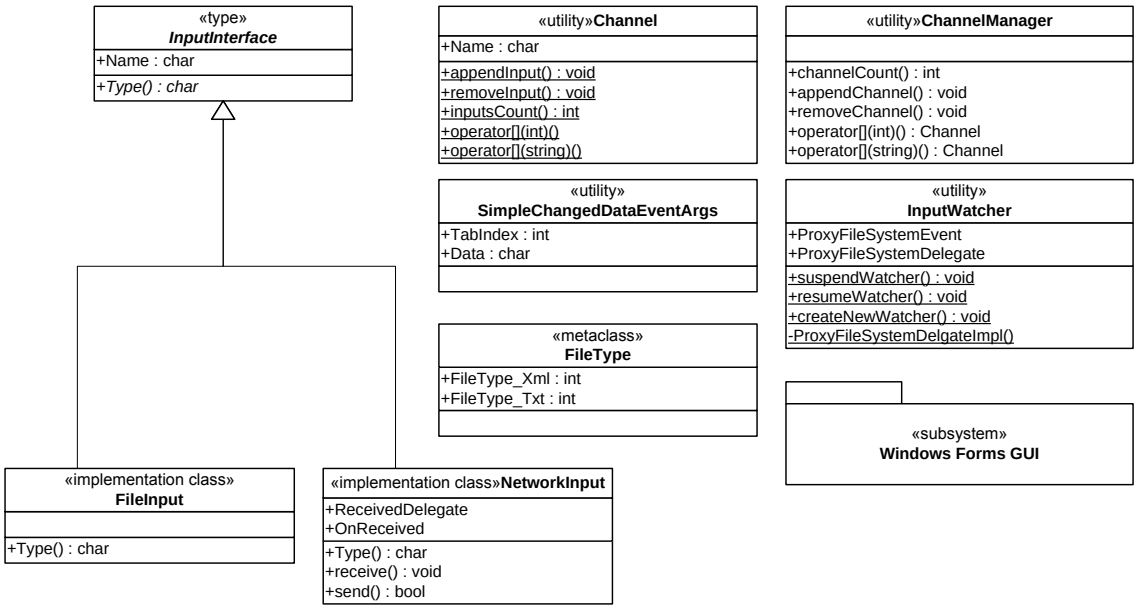
Rysunek 1. Interfejs startowy.

Następnie, aplikacja czeka na działanie użytkownika, który może dodać kanał (zakładkę), podpiąć wejście do aktywnego kanału, włączyć okna zarządzania filtrami oraz okna *About*. W menu aplikacji oraz na pasku narzędzi znajdują się

odpowiadające sobie funkcje. Po podpięciu odpowiedniego wejścia do kanału zauważymy kolejne dodane informacje do podanego kanału.

- **Opis ważniejszych klas i interfejsów.**
  - *InputInterface* – interfejs (klasa abstrakcyjna) reprezentująca wejście w aplikacji. Jest wyposażona w podstawy własnego systemu RTTI (czysto wirtualna metoda *„Type()”*).
  - *FileInput* – Klasa reprezentująca wejście typu plikowego (katalog i filtr na odpowiednie pliki). Implementuje interfejs *InputInterface*.
  - *NetworkInput* – Klasa reprezentująca gniazdo *datagramowe UDP*. Zawiera metody obsługi danych (*send* i *receive*) oraz implementuje interfejs *InputInterface*. Komunikuje się ze światem zewnętrznym za pomocą zdarzenia *OnReceived*.
  - *Channel* – encja reprezentująca kanał, zarządzająca i przechowująca wejścia. Jest zarządcą klas *InputInterface^*, udostępnia minimalny zestaw metod do operacji na elementach kolekcji.
  - *SimpleChangedDataEventsArgs* – Klasa dziedzicząca po klasie systemowej EventArgs, służy do przekazywania własnych komunikatów przy zdarzeniach w systemie, wewnątrz aplikacji (przekazuje: *Data* - dane typu string, *TabIndex* – numer kanału).
  - *FileType* – klasa enumeracyjna (*enum*) zawierająca informacje na temat dostępnych wejść plikowych.
  - *ChannelManager* – klasa dla obiektu globalnego (*singleton*, pole w głównym oknie aplikacji) zarządzająca kanałami. Najważniejszym zadaniem jest zarządzanie, tworzenie i operowanie na stworzonych kanałach (minimalny zestaw metod i akcesorów).
  - *InputWatcher* – typowa klasa narzędziowa, opakowująca (*wrapper*) klasę *.NET Framework 3.5 – FileSystemWatcher*. Zarządza, tworzy i obsługuje wyżej wymieniony obiekt, oraz komunikuje się z systemem za pomocą modelu zdarzeniowego i delegatów (*ProxyFileSystemEvent*).
  - *Windows Forms GUI* – podsystem okien w aplikacji, zawiera pięć bytów:
    1. *LogViewerMainForm* - główne okno aplikacji.
    2. *About* – okno wyświetlające informacje o programie.
    3. *InputTypeDialog* – okno umożliwiające wybór wejścia aplikacji.
    4. *UdpPropertiesDialog* – okno umożliwiające wybór parametrów wejścia UDP.
    5. *ManageChannelsDialog* – okno zarządzające kanałami, umożliwiające usuwanie, zmianę nazw i podglądania podpiętych wejść.

- **Diagram UML aplikacji.**



Rysunek 2. Diagram UML aplikacji.

3. Zastosowane algorytmy i struktury danych.

W aplikacji zastosowane struktury danych to przede wszystkim listy oraz tablice. Nie używano skomplikowanych algorytmów.

Zupełnie inną kwestią jest użycie w aplikacji wątków (klasa *.NET BakgroundWorker*, obsługująca odbieranie danych z portów UDP) oraz obserwacja katalogów pod kątem zmian w plikach (klasa *.NET FileSystemWatcher*, służąca do obserwacji zmian w wybranych plikach i katalogach).

Wykorzystano następujące własności języka *C++*:

- *ATD* i dziedziczenie klas abstrakcyjnych (implementacja interfejsów).
- Obsługa wyjątków (własnych oraz systemowych).
- Własne *RTTI* (dla rozpoznawanie klas wejść).
- Obsługa strumieni plikowych (pliki tekstowe i *XML*) oraz strumieni sieciowych.

Wątki wymusiły użycie pewnych wzorców w aplikacji (*design patterns*) oraz specyficznej obsługi wyjątków wewnątrz wątku (puste, przyjmujące wszystko klauzule *catch*).

4. Specyfikacja zewnętrzna.

- **Instrukcja.**

Podstawowe pojęcia używane w aplikacji:

- Kanał – zakładka, aktywny kanał to aktualnie wybrana zakładka.
- Wejście – jeden z elementów kanału.

Podstawowe operacje:

- Menu *Help*, pozycja *About* – wyświetla okno dialogowe pokazujące podsumowanie dla całej aplikacji i dane autora.
- Menu *Edit*, pozycja *Manage channels* oraz przycisk na pasku narzędzi – wyświetla okno dialogowe służące do zarządzania kanałami.
- Menu *File*, pozycja *Create channel* oraz przycisk na pasku narzędzi – dodaje nowy kanał (nazwę zmieniamy w oknie dialogowym służącym do zarządzania kanałami).
- Menu *File*, pozycja *Join input* oraz przycisk na pasku narzędzi – dodaje wejście do aktywnego kanału (wybranej aktualnej zakładki).
- Menu *File*, pozycja *Exit* – wyjście z aplikacji.

Okno dialogowe dodawania kanału – możliwość wyboru typu kanału. Dla kanału plikowego (*XML* lub *TXT*) następny etap do wybór katalogu. Dla *UDP socket* następne okno to okno wyboru *portu* oraz *hostname*.

Okno dialogowe zarządzania kanałami – możliwość usunięcia, zmiany nazwy kanału oraz podejrzenia aktywnych wejść w kanale.

Gdy aplikacja jest zminimalizowana powiadomienia o przychodzących informacjach pojawiają się przy ikonce w zasobniku systemowym.

5. Testowanie i uruchamianie.

- **Proces testowania i obsługa błędów.**

  Podczas testowania aplikacji pojawiła się potrzeba tworzenia małych, bezpołączeniowych i lekkich aplikacji wysyłających *datagramy UDP* na odpowiednim porcie. Do tego celu wykorzystano język *Python*, i jego klasy o nazwie *Socket*. Dodatkowo utworzono plik **.bat* uruchamiający serwery. Całość znajduje się w katalogu *Testing*.

  Cała aplikacja była kilkakrotnie przetestowana w różnych kombinacjach kanałów i wejść.

- **Błędy znalezione podczas uruchamiania.**
  - *Błąd nieobecnej kontrolki przy obsłudze danych z portów UDP.*

    Podczas obsługi danych z portów UDP, w momencie zamykania aplikacji kontrolka ListBox, do której mieliśmy dodać dane stawała się pustym wskaźnikiem (a aplikacja mimo to próbowała dodać te dane do niej).

  - *Niewłaściwy indeks podczas dodawania wejść do ChannelManagera.*

    Podczas dodawania wejścia do globalnego ChannelManagera, używany był niepoprawny indeks, przez co wejście znajdowało się w niewłaściwym kanale.

6. Uwagi i wnioski.

- Aplikacja korzysta z *Windows Forms* i *.NET Framework 3.5* - mimo, iż język *C++/CLI* do przyjemnych nie należy, to sama platforma bardzo ułatwia tworzenie aplikacji.
- Podczas tworzenia aplikacji największą przeszkodą było poznanie zasad panujących w tym języku (samą platformę *.NET* znam w stopniu dobrym) oraz zastosowanie odpowiedniej obsługi zdarzeń i komunikatów (*delegates and events pattern*).
- Sam język umożliwia przeplatanie klas natywnych i zarządzanych, przez co możliwe jest używanie wszystkich własności języka C++ na platformie *.NET*.
- Warto do aplikacji dodać informacje (opisy), z którego wejścia pochodzi informacja w danym kanale (dla mnie ta informacja jest nadmiarowa).