LogLiteViewer – przeglądarka logów w czasie rzeczywistym

Wojciech Gawroński gr, 5, Inf4 (PK4)

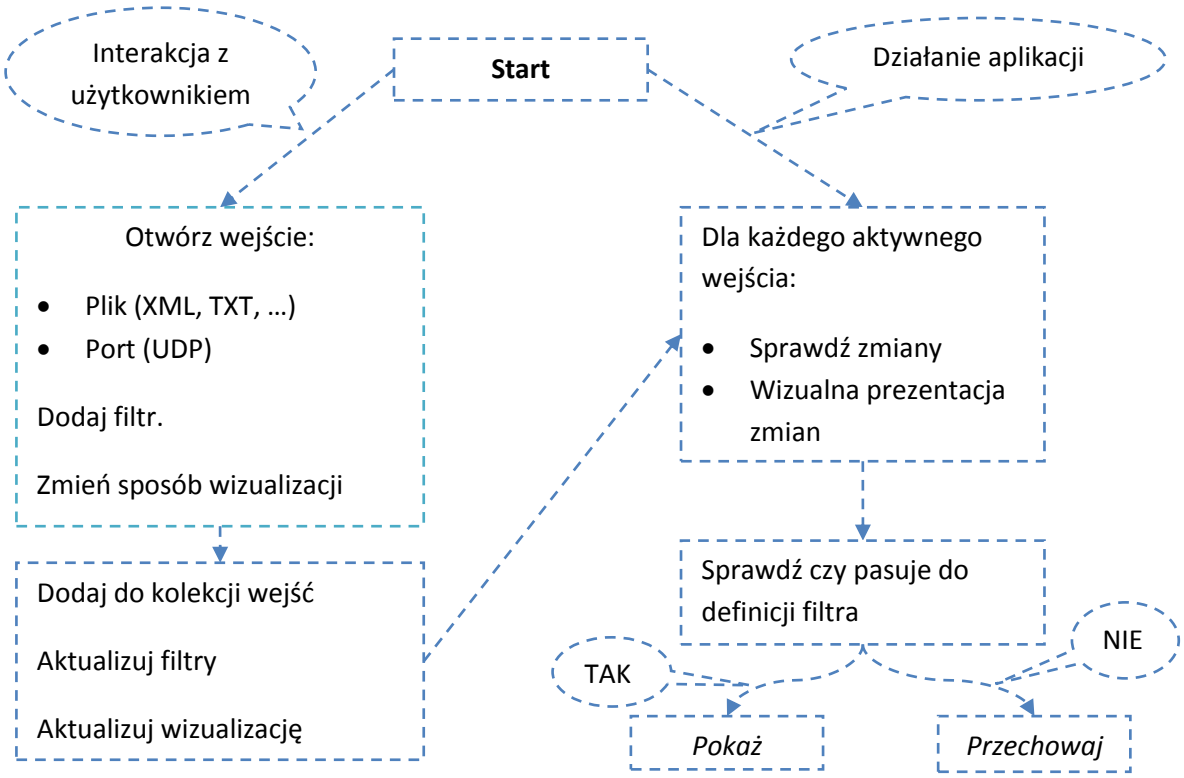
1. Ogólny zarys i przedstawienie problemu.

Załóżmy, że aplikacja generuje wiele logów w trakcie swojego cyklu życia. Logi te to kilka różnych formatów. Przy analizie takich danych potrzebujemy kilku aplikacji, aby przeglądnąć te informacje oraz je np. przefiltrować.

Zadaniem LogLiteViewer'a będzie złączenie wszystkich logów w jednej aplikacji, tak, aby w spójny sposób analizować, filtrować i przetwarzać zgromadzone w nich zdarzenia. Aplikacja powinna mieć możliwość obsługi różnych typów wejść (XML, TXT, UDP), dodatkowo powinna mieć możliwość scalania, rozłączania, filtrowania konkretnych wejść tak, aby ułatwić analizę. Powinna być także lekka, tak, aby jej użycie nie wiązało się ze zwiększonym zużyciem zasobów komputera.

2. Zasada działania.

Po starcie aplikacja czeka na instrukcje od użytkownika. Oto prosty diagram działania dla LogLiteViewer'a:



3. **Koncepty i klasy**
    - Wejście – plik lub port (abstrakcyjna klasa bazowa dla UDP oraz plików tekstowych). Klasy pochodne to:
        - NetworkStream
            - UDPInput
        - FileStream
            - XmlLogInput
            - TxtLogInput
    - Filtr – wyrażenie regularne filtrujące dane w danym kanale (obiekt narzędziowy, opakowanie dla biblioteki boost::regex).
    - Kanał – jedno lub wiele wejść spiętych ze sobą (obiekt narzędziowy przechowujący kolekcję wejść, najpewniej, jako inteligentne wskaźniki na klasę bazową).
    - Format – sposób przetwarzania i analizy pliku (Kolekcja parametrów oraz komend podpiętych do nich).
    - InputWatcher – klasa służąca do obserwacji wejść (plików, socketów itd. Typowa klasa narzędziowa). Osobno będzie reprezentowana klasa do odpytywania wejść plikowych i wejść sieciowych (obserwacja plików oraz sockety są realizowane przez WinAPI, za wielowątkowość odpowiada boost::threads).
    - Kolejną koncepcją jest tabelka, w której będą prezentowane dane. Będzie to tabela o bogatych możliwościach edycyjnych, możliwości sortowania i filtrowania po dowolnej kolumnie – ta tabelka reprezentuje kanał. Dodatkowo jedna tabelka, to jedna zakładka w programie, (czyli jeden kanał = jedna zakładka). Wpisem w tabelce jest informacja pobrana z wejścia.

4. **Struktury danych i użyte algorytmy.**

    Struktury danych to przede wszystkim listy i tablice haszowane (jako kolekcje do przechowywania wejść, filtrów etc).

    Algorytmy użyte to dopasowywanie wyszukanego tekstu do wyrażenia regularnego.

5. **Harmonogram.**
    - **a.** Pierwsza faza: Implementacja pobierania danych z wejść (4 dni).
    - **b.** Druga faza: GUI (7 dni).
    - **c.** Trzecia faza: Ostatnie szlify logiki i podpięcie pod GUI (7 dni).
    - **d.** Ostatnia faza: Szlify oraz testy aplikacji (maksymalnie 7 dni).