Lekcja 3. Statystyki – narzędzie Result Analysis

Lekcja 3 ma na celu zapoznanie czytelnika z narzędziami do prowadzenia statystyk i analizowania rezultatów symulacji. Objaśnione zostaną zagadnienia związane z plikami wektora oraz skalara wyjściowego. Poznamy również klasy służące do gromadzenia statystyk.

Analizowanie rezultatów symulacji

Wyniki symulacji mogą zostać zapisywane do plików:

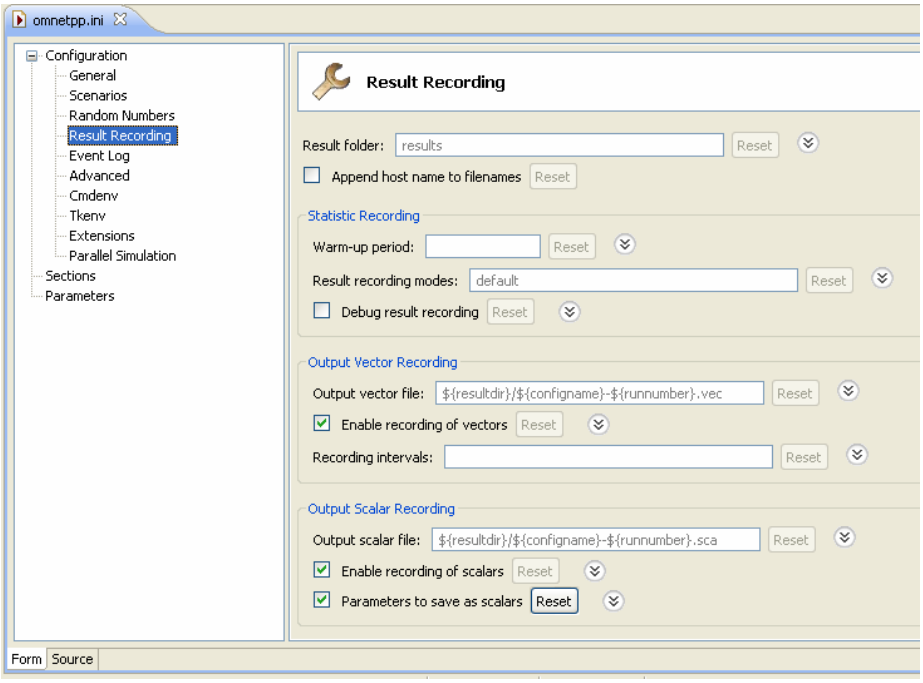
- wektora wyjściowego (`output vector files`)
- skalara wyjściowego (`output scalar files`)

OMNeT++ posiada narzędzie do analizy statystycznej symulacji **Result Analysis.** Umożliwia ono podgląd plików odpowiednio wektora wyjściowego oraz skalara wyjściowego.

Wektory wyjściowe można używać do zapisu czasu obiegu komunikatów, długości kolejek, czasu przebywania obiektu w kolejce, liczby połączeń. Są to informacje, które obrazują wszelkie wartości zmienne w czasie.
Wektory wyjściowe są zapisywane poprzez obiekty klasy `cOutVector` w modułach prostych. Do jednego pliku wektorów wyjściowych może zostać zapisanych wiele obiektów tej klasy. Pliki te mają rozszerzenie *.vec.
Obiektom klasy `cOutVector` możemy nadawać nazwy poprzez użycie funkcji `setName`:

```
wielkosc_przesylanego_zad.setName("wielkosc komunikatu");
```

W pliku *omnetpp.ini* można ustalić szczegóły wykorzystania wektora wyjściowego, określić m.in. przedział czasowy w którym obiekty będą zapisywane, określić odstępy czasu zapisu. Wygodnie jest użyć tym celu formularza, w który Omnet++ 4.2 został wyposażony (Rys 1).



Rys 1. Formularz pliku omnetpp.ini– widok na „Result Recording”

Plik ned wygląd wtedy w następujący sposób (widok „source”).

```
[General]
network = Lekcja3
warmup-period = 10s
**.scalar-recording = true
**.vector-recording = true
**.vector-recording-intervals = ..100
```

Badania można przeprowadzać w określonym przedziale czasowym. Służy do tego opcja konfiguracyjna `vector-recording-intervals`. W tej opcji można ustalić przedziały czasu, w których mają być rejestrowane dane np.

```
**.vector-recording-intervals = ..100, 150..290
```

Można także określić czas tzw. rozgrzewania - `warmup-period`. Dzięki tej opcji rezultaty działania symulacji będą zapisywane do zaimplementowanych wektorów oraz skalarów dopiero po upłynięciu określonego czasu np.

```
warmup-period = 10s
```

Wpisy w pliku *.vec* kolejno oznaczają numer wektora, czas w sekundach i zarejestrowaną wartość wraz z modułem, który ją wyprodukował.

Wektory wyjściowe zapisują dane oraz czas ich zarejestrowania, lecz jeśli

potrzebujemy porównać różne zdarzenia dla różnych ustawień zmiennych, dla których nie jest istotna jednostka czasu, musimy wykorzystać **skalary wyjściowe**.

Dane do pliku skalarów wyjściowych (plik z rozszerzeniem *.sca*) zapisywane są za pomocą metody `recordScalar()` z klasy `cSimpleModule`. Każde jego wywołanie powoduje wpis jednej linii do pliku.

Przykład

Wykorzystanie oraz obsługę narzędzi symulacyjnych OMNeT++ przedstawię na przykładzie symulacji, która została wykonana w *lekcji 2*. Zostanie ona zmodyfikowana poprzez dodanie odpowiednich funkcji zapisujących wektory i skalary do odpowiednich plików wyjściowych.

W wektorach oraz skalarach zapisane zostaną dane uzyskane w module `Procesor`: czas obsługi zadań, długość kolejki, wielkość komunikatu, czas transmisji do klienta.

Aby zapisać wektory wyjściowe potrzebujemy zadeklarować zmienne należące do klasy `cOutVector`. Zostanie to zapisane w pliku nagłówkowym *procesor.h*.

```
cOutVector czas_obslugi_wykr, liczba_zadan_wykr,
wielkosc_przesylanego_zad, czas_transmisji;
```

W funkcji `activity()` nadamy tym wektorom nazwy.

```
czas_obslugi_wykr.setName("czas obslugi zadan");
liczba_zadan_wykr.setName("dlugosc kolejki zadan");
wielkosc_przesylanego_zad.setName("wielkosc przesylanego
komunikatu");
czas_transmisji.setName("czas transmisji do klienta");
```

Czas obsługi żądania, długość kolejki oraz wielkość przesyłanego komunikatu mamy dostępne w funkcji `obsluz_zad_na_procesorze` więc w niej zapiszemy odpowiednie wpisy do pliku wyjściowego za pomocą metody `obiekt_klasy_outVector.record(wartosc)`.

```
void Procesor::obsluz_zad_na_procesorze( cMessage *zadanie)
{
    ...

    double wielkosc = zadanie->getByteLength();

    czas_obslugi_wykr.record( czas_wyszukiwania_danych );
    liczba_zadan_wykr.record( kolejka.length() );
    wielkosc_przesylanego_zad.record(wielkosc);
```

```
    ...
}
```

Dla skalarów wystarczy wykorzystać metodę `recordScalar(„nazwa dla wartości”, wartosc)`.

```
recordScalar("czas wyszukiwania danych",
       czas_wyszukiwania_danych);
recordScalar("wielkosc szukanego zadania", wielkosc);
```

Ponieważ czas transmisji do klienta mamy określony w funkcji `wysyla_do_klienta` więc tu wpiszemy odpowiednie metody dotyczące tej wartości.

```
void Procesor::wysyla_do_klient( cMessage *zadanie)
{
     ...

     czas_transmisji.record( czas_transmisji_do_klienta);
     recordScalar("czas transmisji do klienta",
           czas_transmisji_do_klienta);
     ...
}
```

Teraz możemy już uruchomić symulację. Wektory i skalary możemy oglądać podczas symulacji po wybraniu odpowiednio View -> output Vector file lub View -> Scalar output file (po co najmniej jednym zatrzymaniu symulacji) oraz po skończonej symulacji w narzędziach Analysis.

Wykorzystanie klasy cStdDev

Do analizy wyników symulacji można wykorzystać klasę `cStdDev`. Posiada ona metodę `collect` do zbierania kolejnych wyników oraz metody `min`, `max`, `mean` i `stddev` zwracające odpowiednio minimalną, maksymalna, srednią i odchylenie standardowe.

Przykładowo do zbierania danych o wielkości generowanego przez klienta zadania umieszczamy w pliku *Klient.h* deklarację:

```
protected:
    cStdDev wielkosc_zadania_sca;
```

Następnie w pliku *Klient.cc* nadajemy nazwę dla zbieranych wyników i zbieramy odpowiednie dane za pomocą `collect`:

```
void Klient::activity()
```

```
{
    wielkosc_zadania_sca.setName("wielkosc generowanego
zadania");
    ...

    for (;;)
    {
        ...
        wielkosc_danych=par("wielkosc_danych");
        wielkosc_zadania_sca.collect(wielkosc_danych);
        ...
     }
}
```

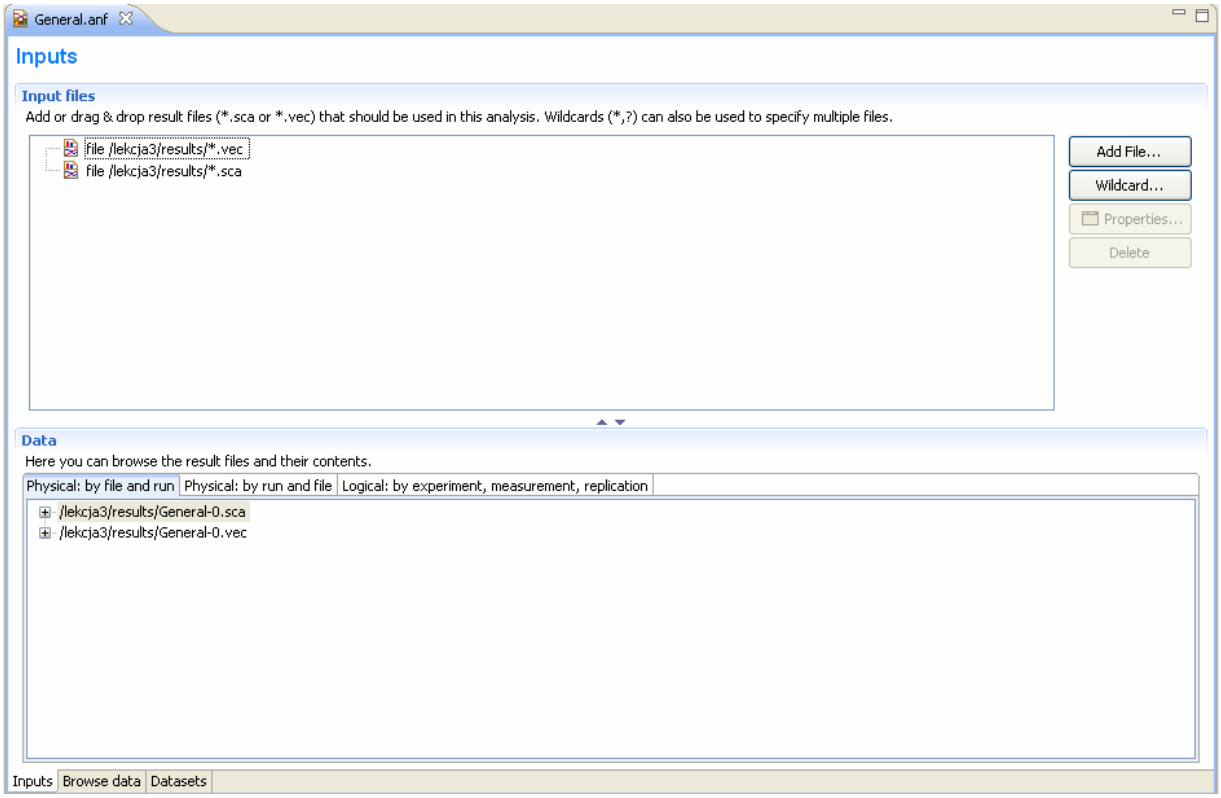
Zapis wyników następuje w momencie zakończenia symulacji po uruchomieniu funkcji `finish()`. Używamy w tym celu metody `record()`:

```
void Klient::finish()
{
    wielkosc_zadania_sca.record();
}
```

Zebrane wyniki są gromadzone jako pliki skalarów.

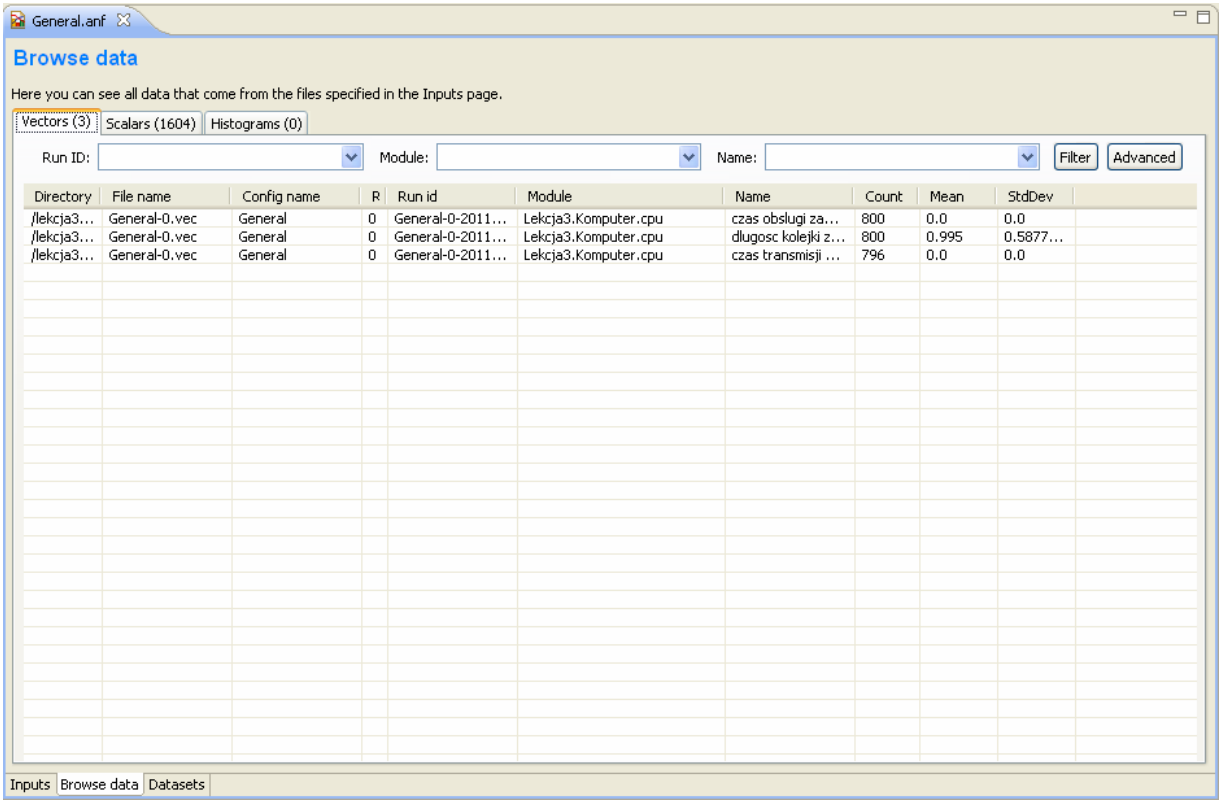
Analysis

Narzędzie *Analysis* służy do odczytywania plików wektorów i skalarów wyjściowych *OMNeT++* i obrazuje wyniki (wykresy, siatki X-Y). Konieczne jest utworzenie pliku narzędzia *Analysis* (.anf). Tworzy się on automatycznie podczas kliknięcia na wybrany plik *.sca lub *.vec otrzymany podczas symulacji. Plik ten edytujemy za pomocą formularzy. Rys 2 przedstawia widok „inputs” formularza edycji pliku .anf.



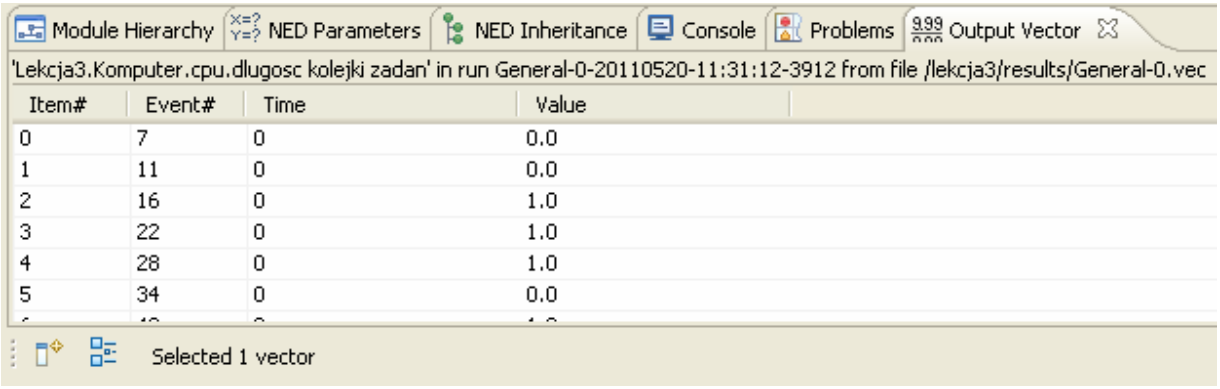
Rys 2. Formularz pliku .anf - widok na „inputs”

Podajemy, które pliki maja być brane pod uwagę w narzędziu *Analysis*. W oknie Browse data (Rys 3) widzimy wektory, skalary i histogramy, które tworzyły się podczas symulacji.



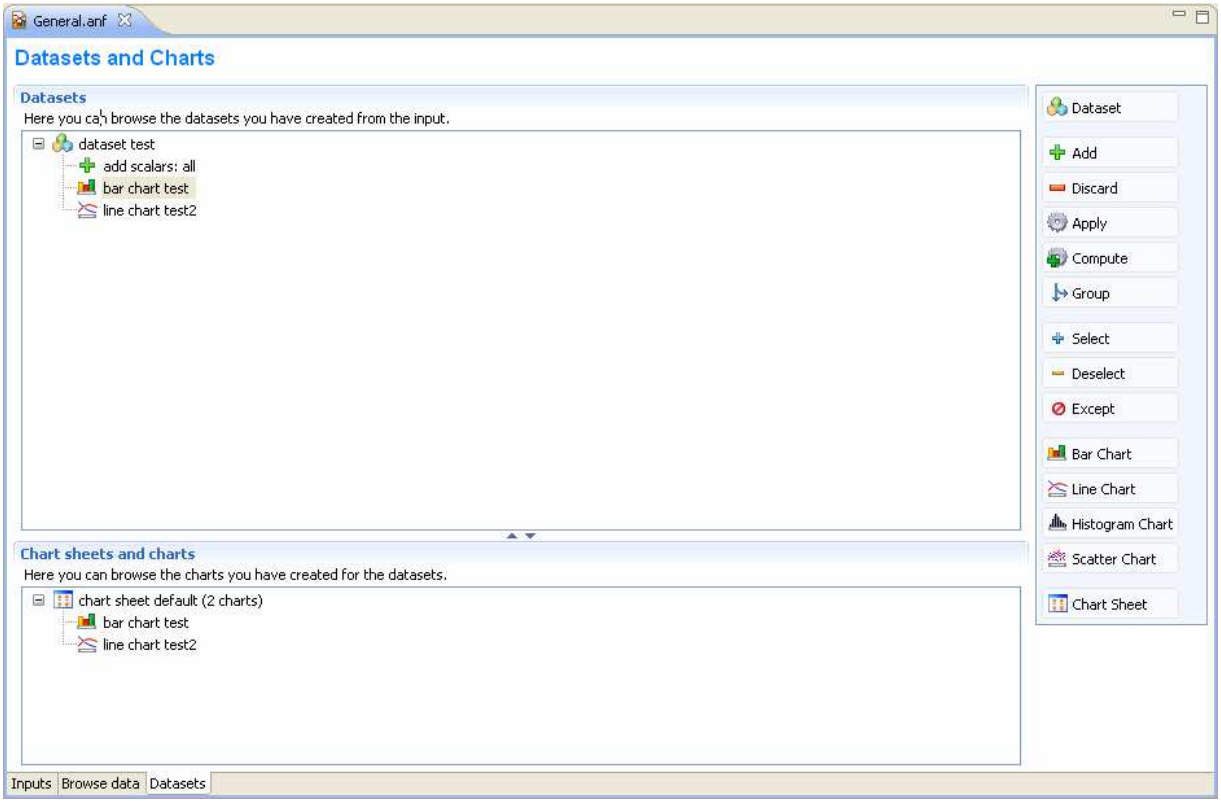
Rys 3. Formularz pliku .anf - widok na „Browse data”

Dane z plików są przedstawiane w formie tabel z kolumnami zawierającymi: ścieżkę dostępu, nazwę pliku, nazwę konfiguracji, numer procesu, nazwę modułu, który go wyprodukował oraz nazwę zmiennej i wartość. Możliwe jest podejrzenie wektorów wybierając zakładkę „Output Vector" (Rys 4).



Rys 4. Zakładka „Output Vector"

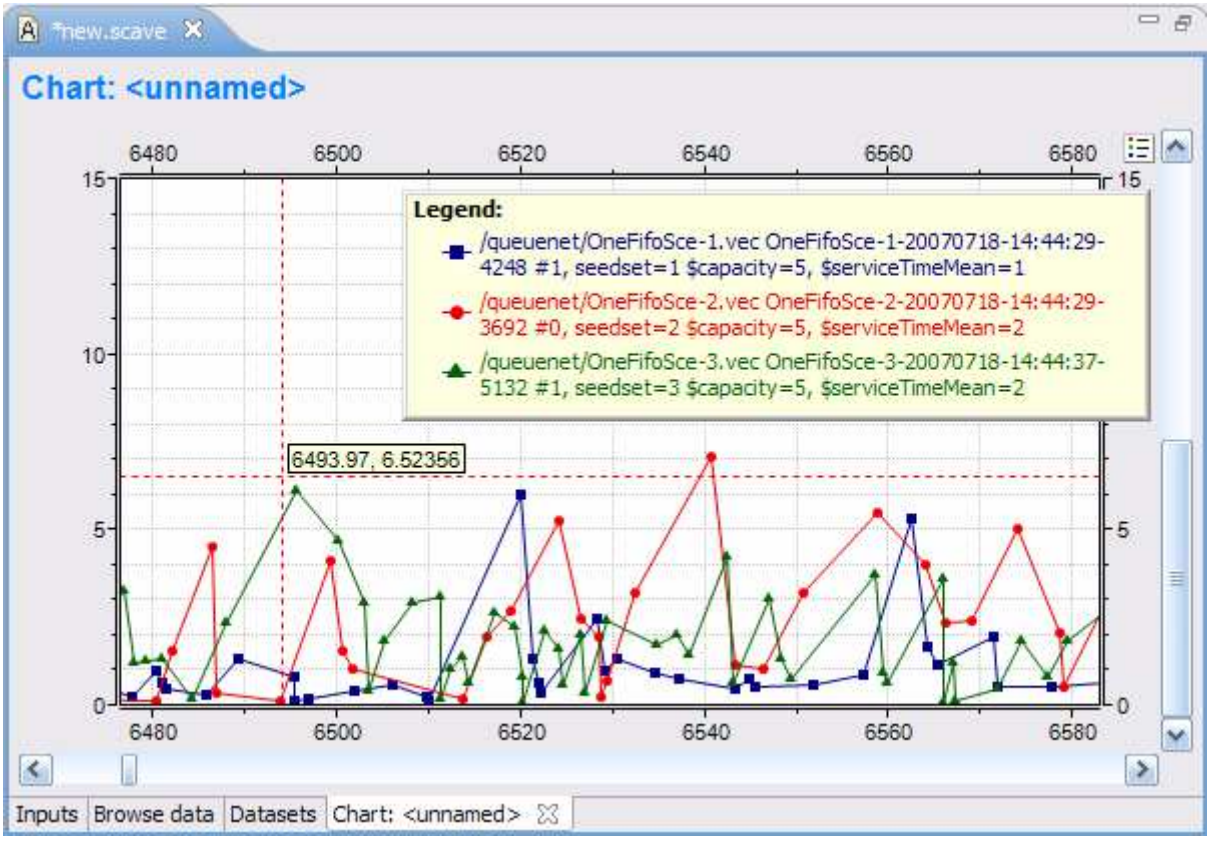
W przeciwieństwie do poprzednich wersji programu *OMNET++,* gdzie były dwa osobne narzędzia *Plove* i *Scalar*, teraz wystarczy nam narzędzie *Analysis.* Rys 5. przedstawia widok umożliwiający utworzenie „Dataset'ów" i utworzenie profesjonalnych wykresów, z wystarczająco rozbudowaną możliwością edycji.



Rys 5. Formularz pliku .anf - widok na „Datasets and Charts"

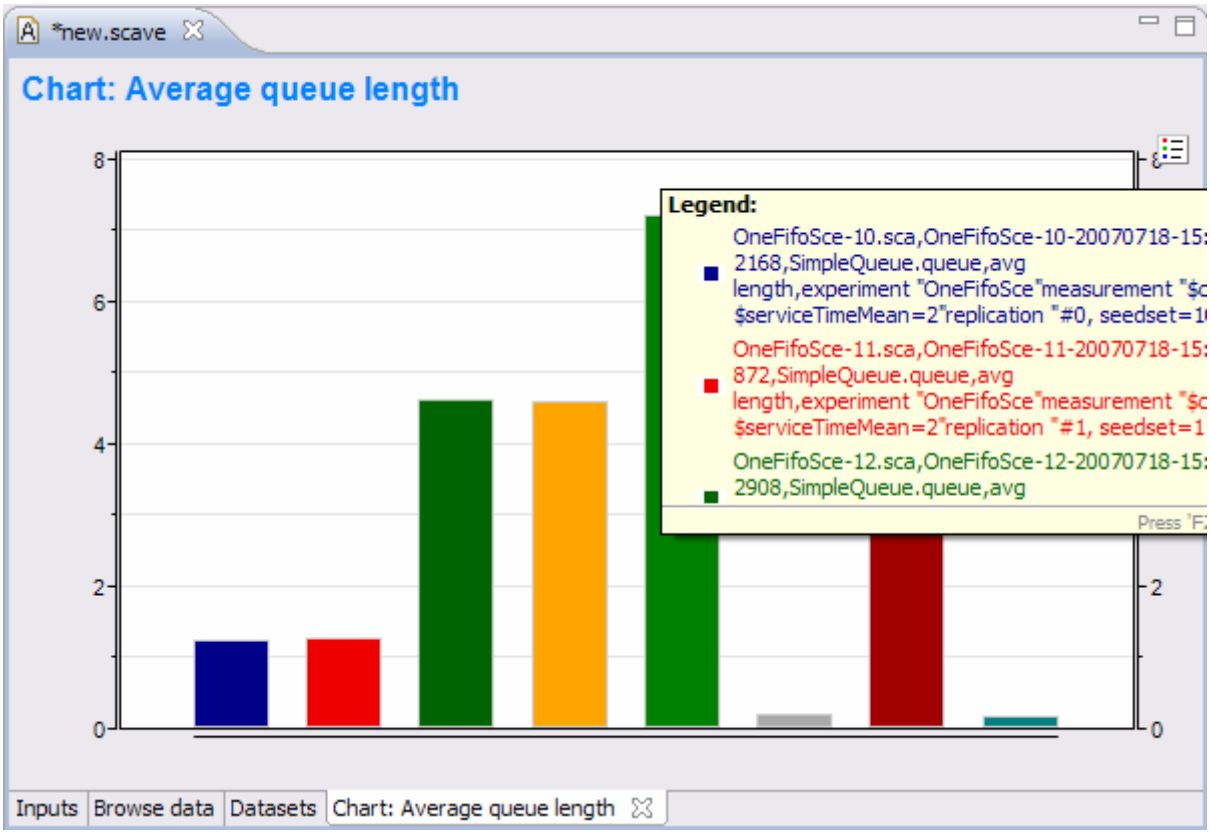
Analysis daje również różne opcje filtrowania wyników przed wyświetleniem wykresów. Istnieje możliwość exportu danych formatach programów:
-CSV
-Matlab
-Octave

Możliwe jest zdefiniowanie jednorazowych zestawów danych, które są w zasadzie wzorcami, jak i wybrać i przetwarzać dane otrzymane z symulacji. Można dokonać selekcji i przetwarzania danych węzłów w zbiorze danych.
Wykresy liniowe (Rys 6) są zwykle używane do przedstawiania danych przechowywanych w plikach wektorowych. Wstępne przetwarzanie (edycja) danych jest możliwe w „Dataset'ach". Dane wektorowe można dowolnie konfigurować, aby wyświetlić dane w zależności od potrzeb.



Rys 6. Przykład wykresu liniowego

Wykresy słupkowe (Rys 7) tworzone są najczęściej z wyników symulacji zapisanych w skalarach i histogramach. Dane mogą być grupowane. Kolory, typ wykresu i inne atrybuty można dowolnie edytować.



Rys 7. Przykład wykresu słupkowego

Klasy do gromadzenia danych statystycznych

OMNeT++ posiada wiele klas, które przeznaczone są do gromadzenia danych i wykonywania na nich różnych obliczeń.

Główną klasa jest klasa `cStatistic` do której należą m.in. funkcje:

- `samples()` – liczba zliczonych próbek
- `sum()` – suma wartości
- `weighst()` – suma wag
- `min()` – najmniejsza zgromadzona dana
- `max()` – największa zgromadzona dana
- `mean()` – średnia arytmetyczna
- `stddev()` – odchylenie standardowe
- `recordScalar()` – zapis do pliku skalarów wyjściowych

Z klasy `cStatistic` wywodzą się z klasy histogramów `cVarHistogram`, `cLongHistogram` oraz `cDoubleHistogram`, które posiadają funkcje m.in. ustawiania rozmiarów histogramów.

Istnieje również klasa `cOutVector`, za pomocą której można gromadzić wszelkie wartości zmienne w czasie jak np. długość kolejki, opóźnienia, czas kolejkowania.

Zadanie do samodzielnego przygotowania:

W zadaniu do samodzielnego wykonania z lekcji 2 proszę dokonać odpowiednich modyfikacji pozwalających na gromadzenie statystyk dotyczących dysku twardego. Proszę gromadzić również statystyki dotyczące czasu odpowiedzi na żądanie wysyłane przez klienta – czas ten definiowany jest jako czas od momentu wysłania żądania przez klienta do momentu otrzymania przez niego odpowiedzi.
Proszę nauczyć się interpretować otrzymane i wyświetlone dane w programie Analysis.

Uwaga!

W celu otrzymania czasu odpowiedzi na żądanie wysyłane przez klienta należy skorzystać
z funkcji obiektu cPacket. W wersji 4.2 funkcje zostały nieco zmodyfikowane, a mianowicie obecnie wykorzystuje się zamiast:
arrivalTime() - getArrivalTime() oraz
creationTime() - getCreationTime().