CLRProfiler

przygotował: Krzysztof Jurczuk
Politechnika Białostocka
Wydział Informatyki Katedra Oprogramowania
ul. Wiejska 45A
15-351 Białystok

Streszczenie: Dokument zawiera podstawowe informacje na temat narzędzia CLRProfiler, które służy do profilowania aplikacji .NET. Została w nim przedstawiona ogólna wiedza na temat problemów z wydajnością aplikacji wykonywanych w CLR, uproszczona instrukcja obsługi prezentowanego narzędzia oraz uwagi dotyczące konfiguracji. Dodatkowo umówione zostały podstawy działania Garbage Collector'a.

1. Wprowadzanie

CLRProfiler jest darmowym narzędziem do profilowania kodu zarządzanego aplikacji stworzonych w środowisku .NET (CLR – Common Language Runtime – execution engine that executes MS .NET applications). Jest to narzędzie, które głównie skupia się na analizie działania Garbage Collector'a. Dzięki temu pozwala śledzić zmiany w pamięci dla każdego z tworzonych obiektów (np. ilość zarezerwowanej pamięci, czas jej alokacji oraz dealokacji). Nie jest to typowe działanie dla narzędzi profilujących, jednak w aplikacjach .NET możne znacząco wpłynąć na poprawienie ich wydajności i jakości.

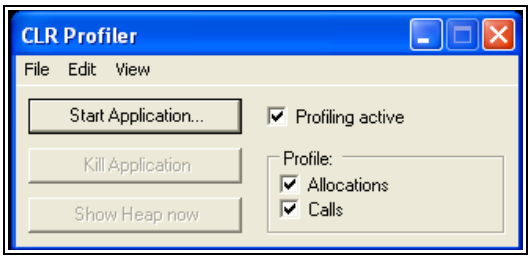
Możemy wyróżnić trzy typy problemów, z którymi najczęściej spotykamy się podczas pracy z zarządzanym językiem programowania:

- alokowanie zbyt dużej ilości pamięci (Garbage Collector musi później poświęcić więcej czasu na jej zaalokowanie i następnie na jej zwolnienie),
- przetrzymywanie nieużywanych obiektów i nie zaznaczanie ich od razu do usunięcia, gdy są już niepotrzebne (dodatkowa zajętość pamięci oraz czas poświęcony na ich oznaczenie do usunięcia przez Garbage Collector'a),
- przetrzymywanie nieużywanych obiektów „na zawsze” (kiedy Garbage Collector nie ma możliwości automatycznego stwierdzenia, że pewne obiekty są już niepotrzebne, tzw. manage memory leaks).

Pierwszym krokiem testującym wydajność aplikacji jest sprawdzenie ilości alokowanej pamięci w jednostce czasu. Architekci oprogramowania z MS uważają, że jeśli jej ilość przekracza 200MB/s mamy doczynienia z problemem, który może powodować spadek wydajności aplikacji. Drugim krokiem jest sprawdzenie czasu, który aplikacja poświęca na pobyt w Garbage Collector'e. Tutaj statystyki mówią, że granica wynosi około 20% całości czasu wykonania testowanego software'u. Po stwierdzeniu przynajmniej jednego z powyższych problemów, standardową procedurą jest dogłębna analiza działania aplikacji pod kątem zarządzania pamięcią, np. za pomocą narzędzia CLRProfiler.

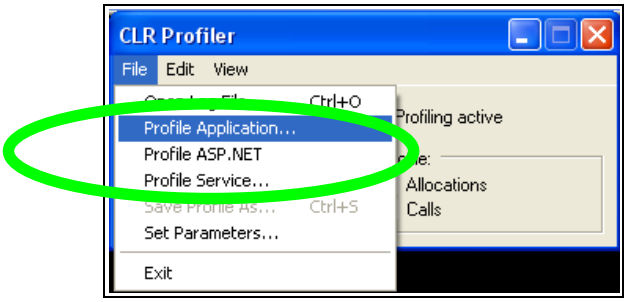
2. Użycie narzędzia CLRProfiler

a) główne okno aplikacji CLRProfiler:

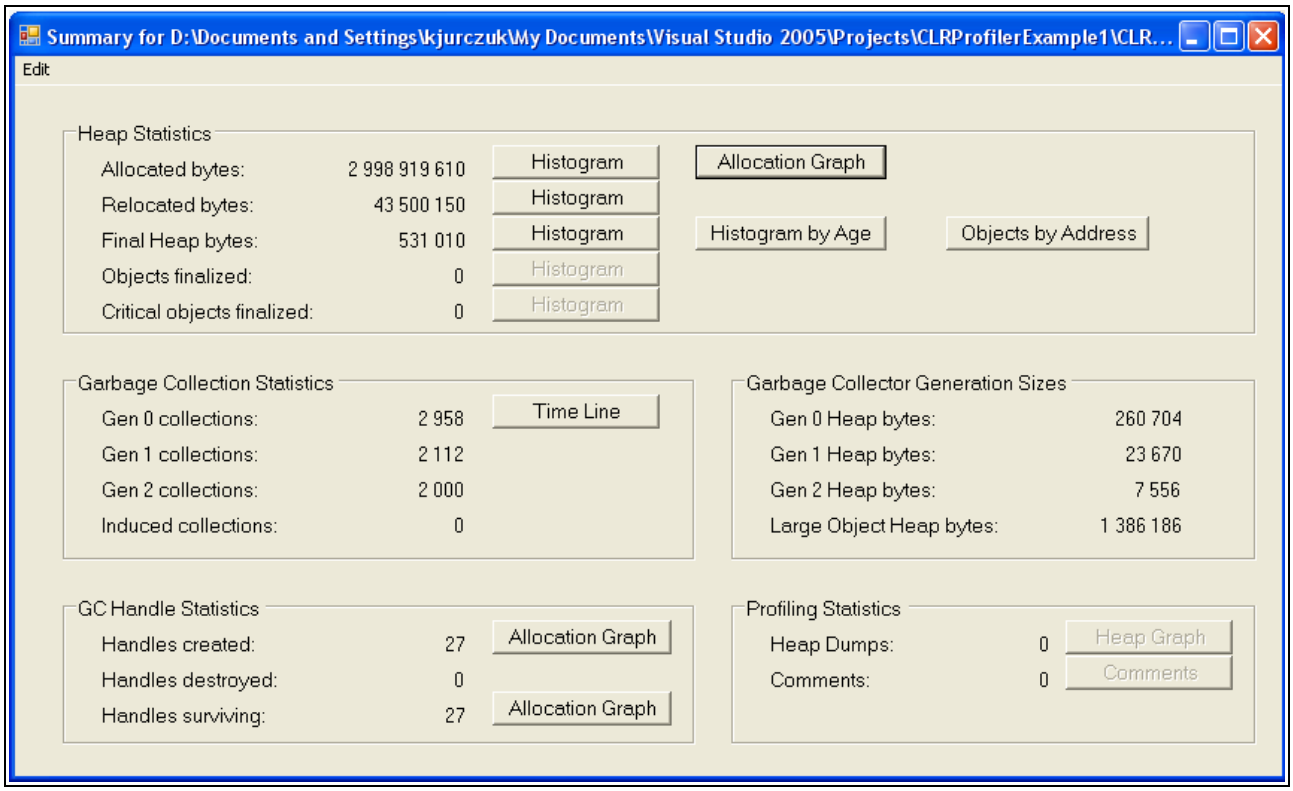


- profiling active (przydaje się, gdy chcemy tymczasowo wyłączyć profilowanie)
- allocations (szczegółowe informacje na temat alokacji i dealokacji pamięci)
- calls (graf wywołań)

b) wybór aplikacji do profilowania



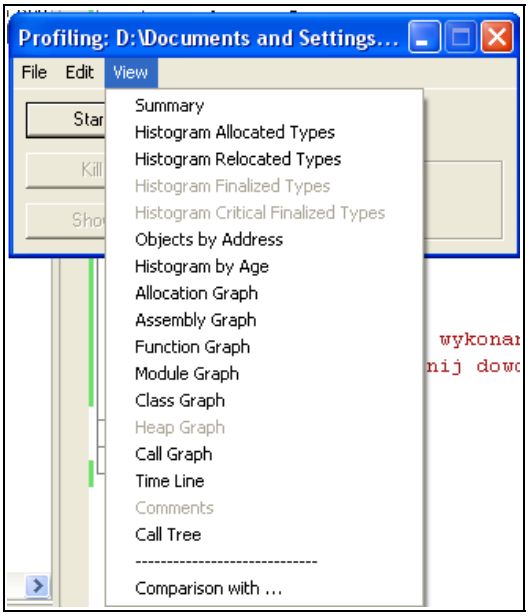
c) podsumowanie po zakończeniu działania profilowanej aplikacji



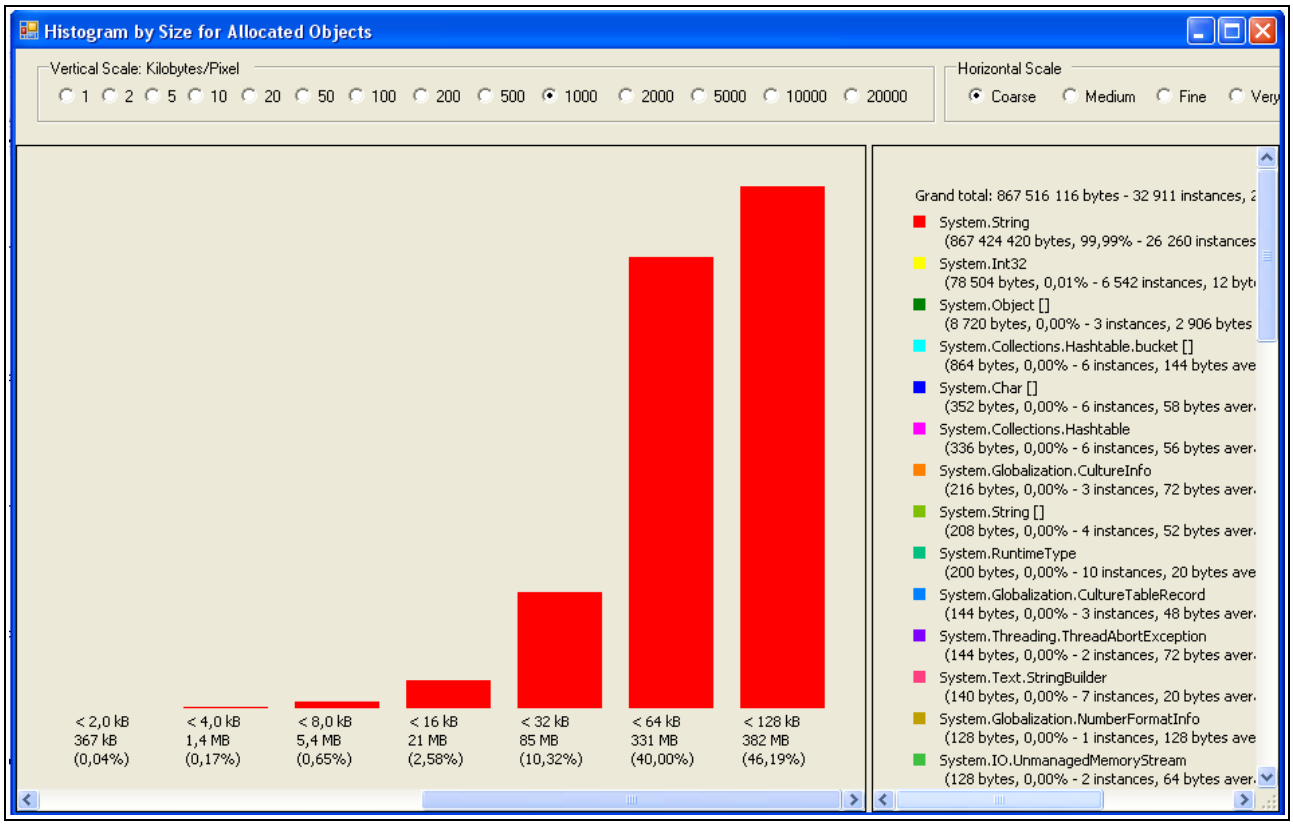
Ważniejsze statystyki podsumowania:

- allocated bytes – ilość zaalokowanej pamięci przez aplikacje dla wszystkich obiektów
- realocated bytes – ilość pamięci, które była „przesuwana” przez GC podczas działania aplikacji (np. do drugiej lub trzeciej kategorii, patrz punkt 4)
- final heap bytes – ilość zaalkowanej pamięci w „garbage collector heap” na koniec działania aplikacji
- objects finalized – liczba obiektów usuniętych

d) pozostałe widoki statystyk

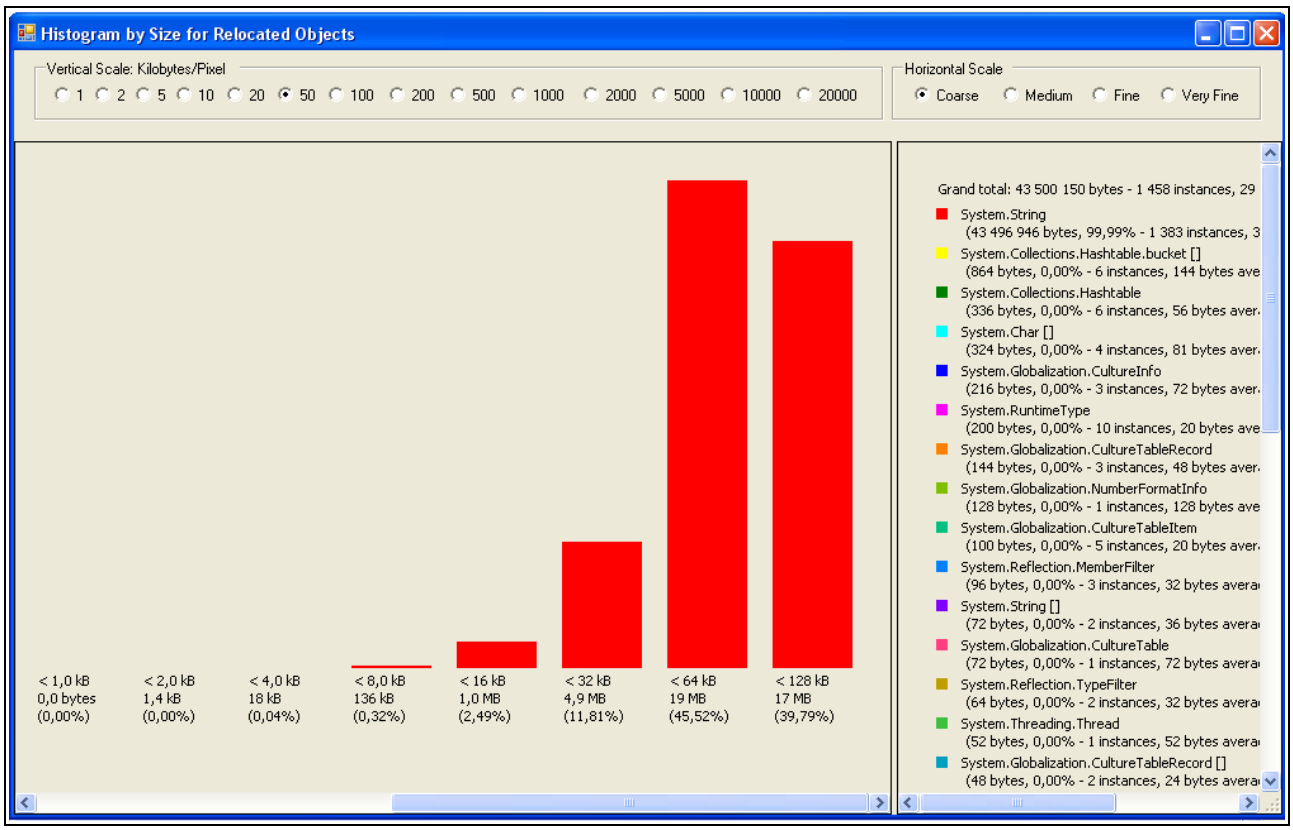


– histogram allocated types



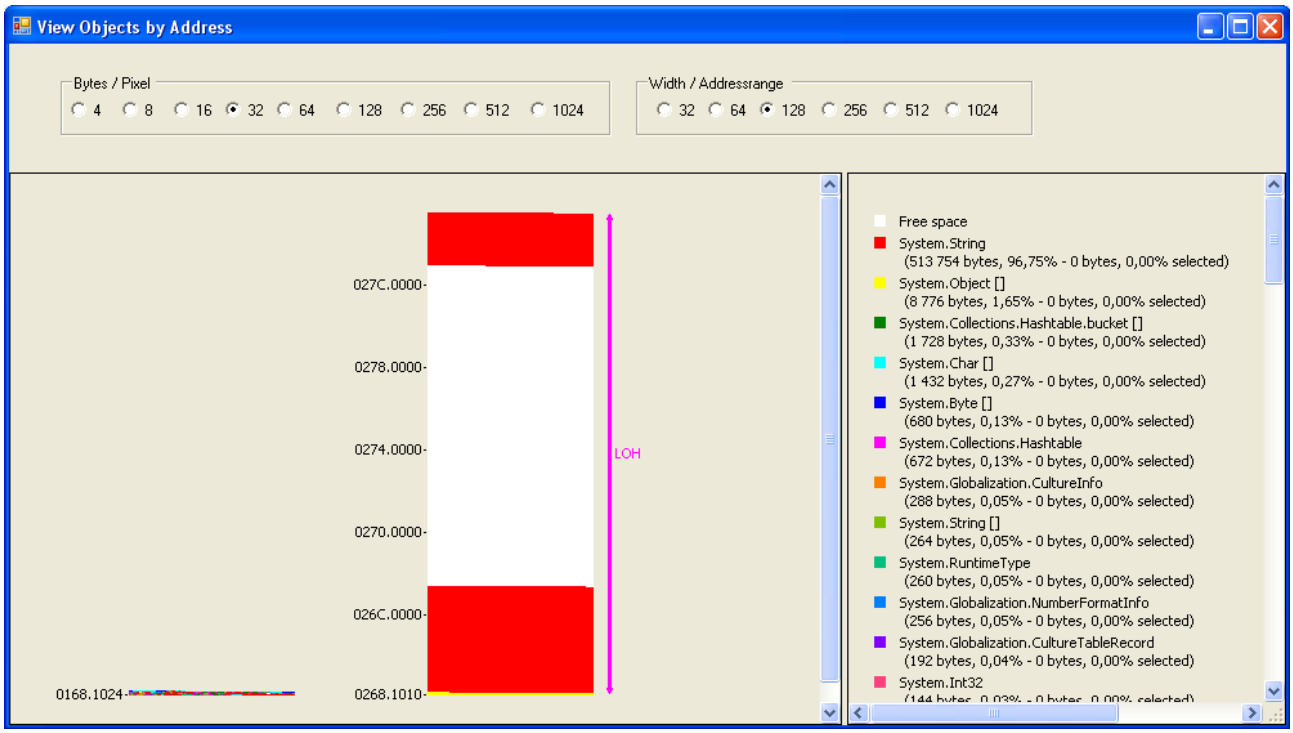
Ilość zaalokowanej pamięci w postaci histogramu dla konkretnych obiektów. Zmiana skali za pomocą checkbox'ow.

- histogram relocated types



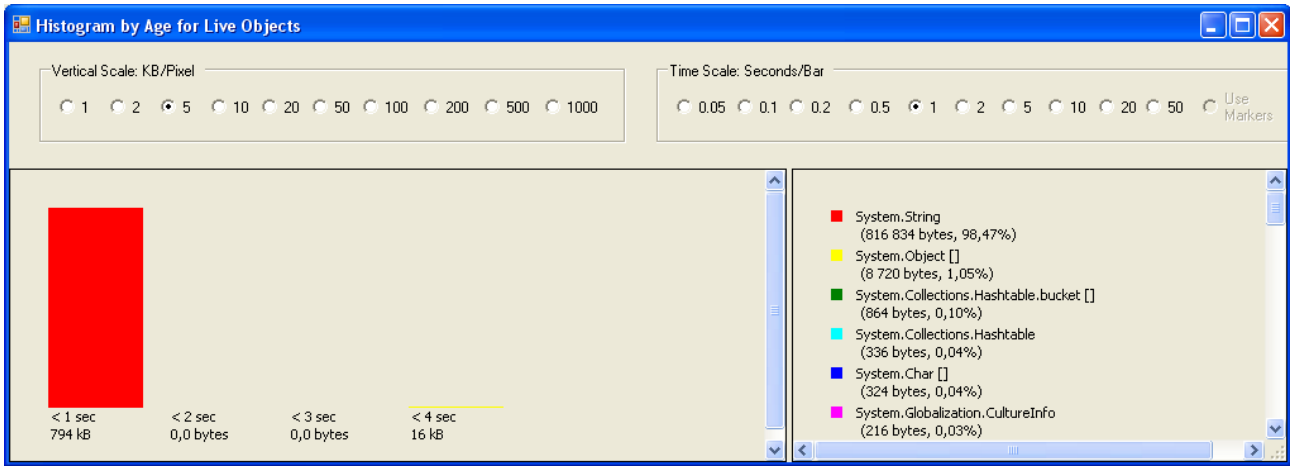
Ilość pamięci realokowanej w postaci histogramu dla konkretnych obiektów. Zmiana skali za pomocą checkbox'ow.

- objects by address



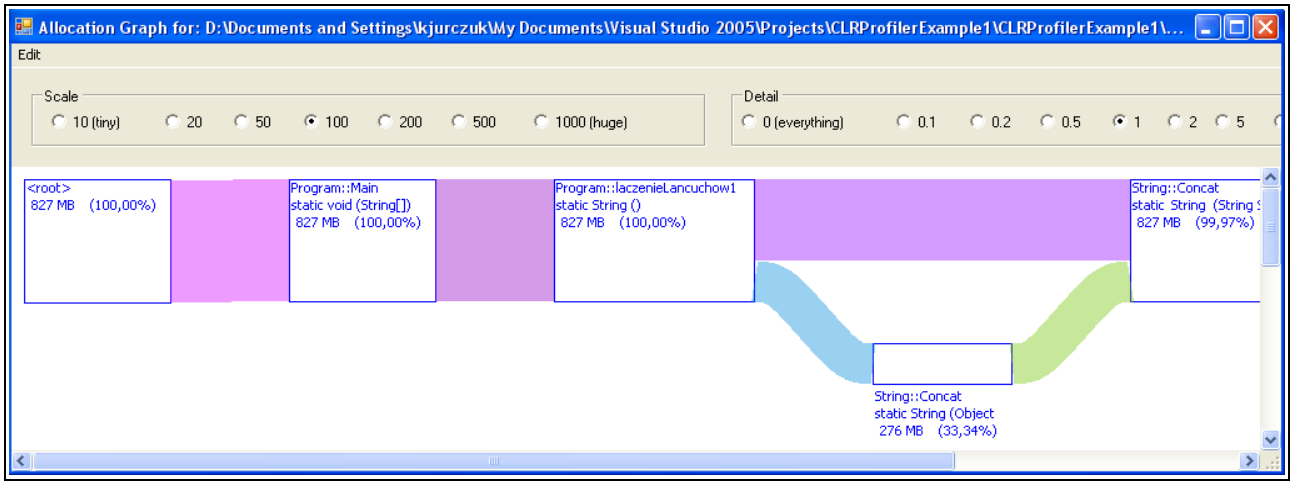
Obraz sterty w momencie zakończenia aplikacji. Obraz w dowolnym momencie czasu wykonania aplikacji można uzyskać poprzez wybranie odpowiedniego momentu w Time Line View.

- histogram by age



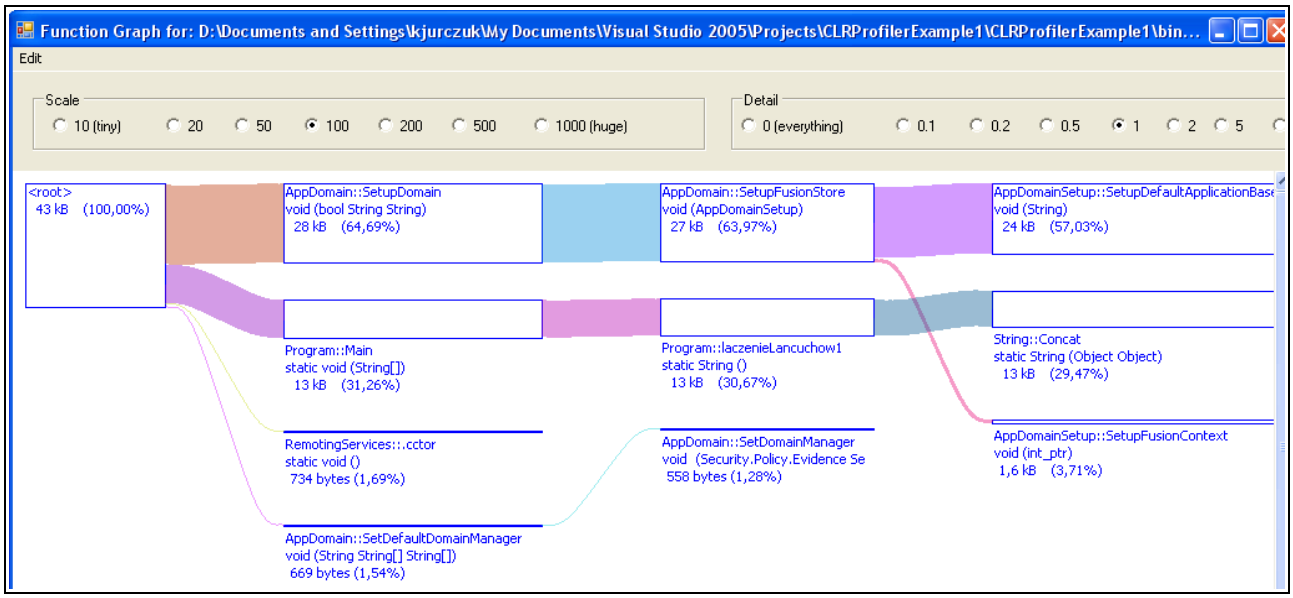
Histogram czasu życia obiektów.

- allocation graph



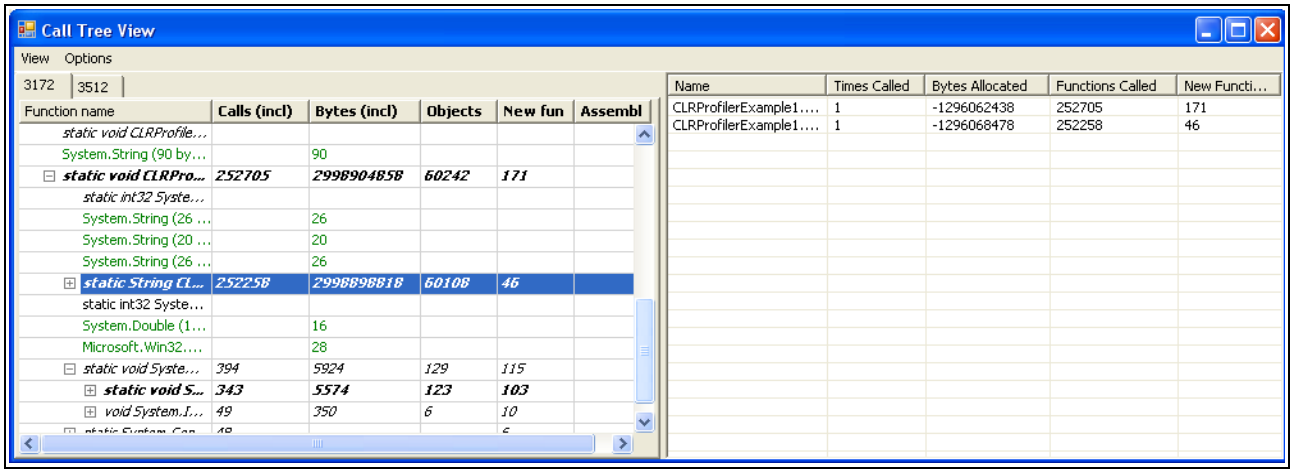
Graf alokowanej pamięci. Za pomocą menu podręcznego możemy filtrować wyniki.

- function graph



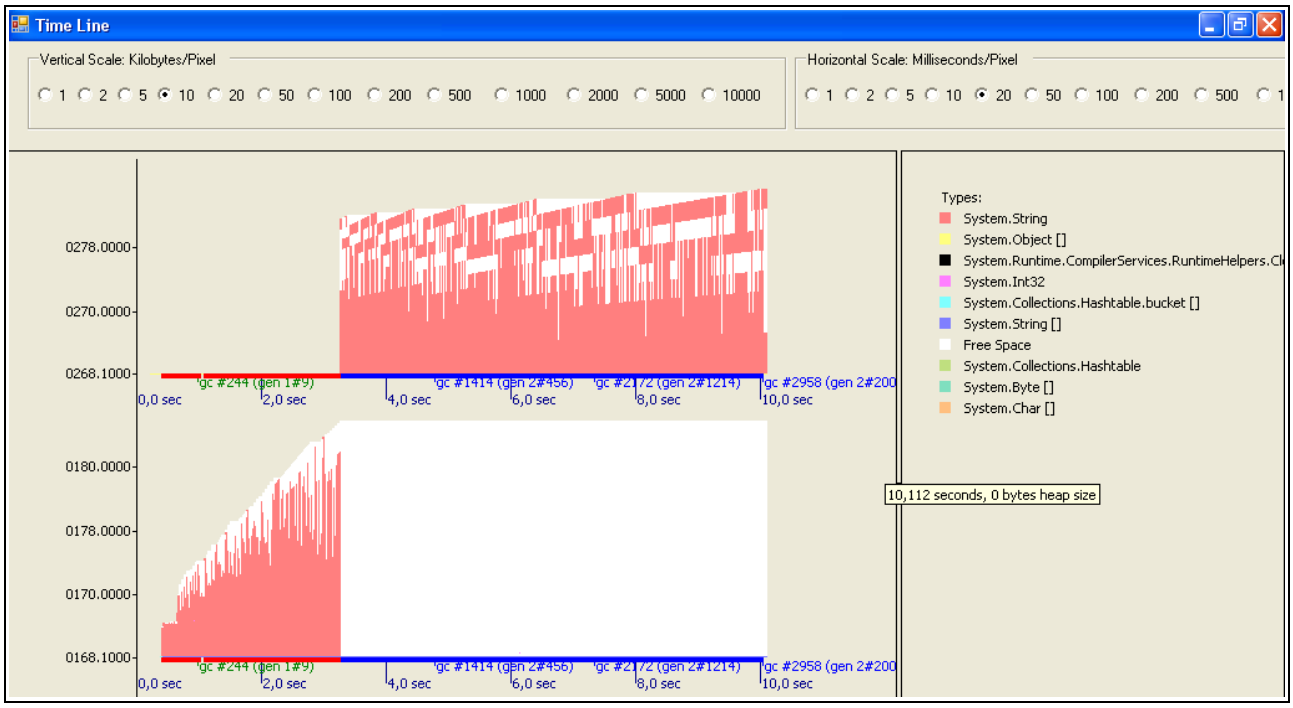
Graf wywołań funkcji. Za pomocą menu podręcznego możemy filtrować wyniki.

- call tree



Szczegółowe drzewo wywołań. Po prawej statystyki wybranego elementu.

- time line



Obraz sterty podczas działania aplikacji (w przykładzie GC był uruchamiany łącznie 2958).

3. Uwagi

Prawidłowe działanie aplikacji CLRProfiler w systemie MS Vista wymaga zarejestrowania biblioteki ProfilerObj.dll. W przeciwnym wypadku aplikacja nie będzie potrafiła połączyć się z profilowanym software'em, co zostanie zakomunikowane poprzez okno z wiadomością „waiting for application to start ...”

4. Garbage Collector

Pamięć dla aplikacji .NET jest przydzielana na stercie. Alokacją i dealokacją pamięci zajmuje się Garbage Collector. Jest to system automatycznego zarządzania pamięcią, dlatego też programista nie ma całkowitej kontroli nad aplikacją. Gdy tworzymy obiekt, Garbage Collector zajmuje się alokacją pamięci dla niego oraz odpowiednim umieszczeniem przydzielonej pamięci na stercie. Gdy nie

potrzebujemy już jakiegoś obiektu jest on oznaczany do usunięcia. Domyślnie nie dzieje się to natychmiast. Co jakiś czas lub po zapełnieniu ustalonej ilości pamięci sterty zwalniana jest niewykorzystywana pamięć. Jest to narzędzie, które ułatwia pracę wielu programistom, ale jednocześnie uniemożliwia pełną kontrolę nad pamięcią. Co więcej algorytmy automatycznego zarządzania pamięcią zużywają wiele zasobów komputera.

Podczas pracy Garbage Collector stara się zachowywać wolne miejsce w postaci ciągłej. Dlatego też co jakiś czas przeprowadza reorganizacje pamięci. W celu optymalizacji tego procesu obiekty zostały podzielone na 3 kategorie. Pierwsza z nich zawiera obiekty, które żyją krótko i tym samym obszar pamięci dla nich przydzielony jest najczęściej poddawany defragmentacji oraz czyszczeniu. Każda kolejna grupa zawiera obiekty, które dłużej egzystują w aplikacji i tym samym wydzielony dla nich fragment pamięci rzadziej ulega reorganizacji.

Istnieje możliwość teoretycznego przyśpieszenia zwolnienia pamięci przez niepotrzebny już obiekt. Służy do tego metoda Dispose(). Z dokumentacji wynika, że powoduje ona zwolnienie pamięci najszybciej jak to możliwe. Jednak zbyt częste jej używanie może wpłynąć na optymalność aplikacji, ponieważ każda ingerencja w pracę Garbage Collector'a może wydłużyć czas jego pracy.

Źródła

[1] Kevin Burton, „.NET CLR. Księga eksperta”, 2006
[2] „How To: Use CLR Profiler”, http://msdn2.microsoft.com/en-us/library/ms979205.aspx
[3] Peter Sollich, „CLRProfiler”, 2005