JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 14 (2020) 26-30
Wysłane: 2019-10-09
Przyjęte: 2019-10-18

Analiza porównawcza technologii Windows Presentation Foundation i Windows Forms

Michał Pasztaleniec*, Maria Skublewska-Paszkowska
Politechnika Lubelska, Katedra Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł przedstawia porównanie dwóch technologii do tworzenia aplikacji desktopowych opartych o język C#. Analiza dotycząca wykorzystania procesora, pamięci RAM oraz czasu wykonywania operacji została przeprowadzona na technologiach Windows Presentation Foundation i Windows Forms. Do badań wykorzystano aplikacje o tej samej funkcjonalności w obu technologiach. W analizie porównawczej uwzględniono czasy danych operacji oraz wydajność aplikacji.

Słowakluczowe: .NET; WPF; Windows Presentation Foundation; Windows Forms; WinForms; analiza wydajności

*Autor do korespondencji.

Adres/adresy e-mail: mmichal210@gmail.com, maria.paszkowska@pollub.pl

Comparative analysis of Windows Presentation Foundation and Windows Forms

Michał Pasztaleniec*, Maria Skublewska-Paszkowska
Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents a comparison of two technologies for creating desktop applications based on C#. The analysis conserns the use of the processor, RAM and time of the operation was carried out on the technologies of Windows Presentation Foundation and Windows Forms. The research use applications with the same functionality in both technologies. The comparative analysis took into account the times of operations and application performance.

Keywords: .NET; WPF; Windows Presentation Foundation; Windows Forms; WinForms; performance analysis

*Corresponding author.

E-mail address/addresses: mmichal210@gmail.com, maria.paszkowska@pollub.pl

1. Wstęp

W artykule porównano dwie technologie, które pomimo swoich lat, nadal są używane do tworzenia aplikacji desktopowych. Wybór został uwarunkowany tym, że na rynek aplikacji desktopowych pisanych w .NET programiści najczęściej wybierają Windows Presentation Foundation i Windows Forms [1]. Drugim powodem jest to, że większość artykułów skupia się na porównaniu teoretycznych różnic tych technologii.

Z corocznych badań serwisu stackoverflow wynika, że aplikacje desktopowe dla systemu Windows zajmują 41% rynku. Także język, w którym pisane są aplikację w Windows Presentation Foundation i Windows Forms, cieszy się dużą popularnością. Język C# według badań serwisu stackoverflow, w 2017 zajmował czwarte miejsce z 34,1% oraz w 2018 na ósmym miejscu z 34,4% udziału [2, 3].

Artykuł skupia się głównie na porównaniu Windows Presentation Foundation i Windows Forms pod względem zużycia pamięci RAM, procesora, a także szybkości wykonywanych operacji. Na potrzeby przeprowadzonej analizy zostało stworzonych osiem aplikacji. Mają one za zadanie wygenerować 1000 kontrolek (typu przycisk lub pola tekstowego) przy uruchomieniu programu, jak i odpowiednio 1000, 2000, 5000 podczas jej działania. Dodatkowo będzie badana szybkość generowania tych kontrolek.

Inne artykuły o podobnym temacie opisanym w tym artykule zwykle skupiają się na różnicach teoretycznych Windows Presentation Foundation i Windows Forms lub tylko zahaczają o tematykę wydajności [4, 5, 6].

2. Windows Presentation Foundation

Windows Presentation Foundation (WPF) jest częścią systemu operacyjnego Windows, która zapewnia graficzny interfejs użytkownika oraz środowisko dla aplikacji i usług. Aplikacje utworzone w tej technologii są uruchamiane tylko w systemie operacyjnym Windows. WPF jest częścią .NET Frameworks 3.0 dzięki czemu posiada narzędzia do tworzenia i obsługiwania grafiki oraz multimediów[7].

Windows Presentation Foundation (WPF) opiera się na języku xaml. Jest to język bardzo podobny do HTML, przez co jest łatwy do zrozumienia oraz bardzo łatwo jest opisać interfejs aplikacji [8]. Dzięki niemu łatwo jest rozdzielić logikę biznesową od interfejsu aplikacji dzięki zastosowaniu

wzorców projektowych takich jak Model-View-Controller (MVC) lub Model–View–ViewModel (MVVM) [9].

3. Windows Forms

Windows Forms jest nazwą graficznego interfejsu programowania aplikacji (API) i częścią pakietu Microsoft.NET Framework [10]. Zapewnia dostęp do macierzystych elementów interfejsu systemu operacyjnego Microsoft Windows. WinForms jest częścią środowiska .NET Framework, dzięki czemu umożliwia natywny dostęp do elementów interfejsu graficznego Microsoft Windows. Aplikacje napisane przy pomocy tej technologii bazują na zdarzeniach, co oznacza, że aplikacja oczekuje na działanie użytkownika takie jak np. kliknięcie na przycisk, wpisanie tekstu do pola tekstowego itp. Tworzenie interfejsu graficznego aplikacji w Windows Forms może przebiegać na dwa sposoby. Pierwszy i najłatwiejszy to przeciąganie kontrolek na ekran aplikacji oraz samodzielne ich ustawianie, drugi to pisanie całej kontrolki w warstwie kodu programu poprzez ustawienie rozmiaru, pozycji na ekranie, czy też jej nazwy [11].

4. Aplikacje testowe

Aplikację badające wydajność napisane są w Windows Presentation Foundation i Windows Forms.Są to programy do generowania kontrolek pól tekstowych oraz przycisków. Następna aplikacja służy do wyświetlania listy w kontrolce widoku listy(ListView). Dodatkowo została stworzona aplikacja wspomagająca pracę kierownika zespołu.

Pierwsze aplikacje testowe mają za zadanie generowanie 1000 razy danej kontrolki (pola tekstowego lub przycisku) podczas uruchomienia aplikacji oraz generowanie ich podczas uruchomionego już programu po naciśnięciu odpowiedniego przycisku.

Następna aplikacja polega na wyświetleniu przy uruchomieniu 1000 elementów w widoku listy (ListView). Implementacja tych list jest identyczna w obu technologiach.

Ostatnia aplikacja ma za zadanie wspomagać pracę kierownika. Posiada trzech użytkowników w systemie. Administrator może dodawać pracowników i ich edytować oraz nadawać uprawnienia. Kierownik może dodawać projekty oraz nowe zadania, do zadań można dodawać pracownika bezpośrednio przy tworzeniu ich lub już do istniejącego. Aplikacja korzysta z lokalnej bazy danych MS SQL oraz z Entity Framework 6.

Powyższe aplikacje zostały stworzone tak, aby jak najbardziej były podobne pod względem wizualnym, jak i pod względem implementacyjnym.

5. Metoda i platforma testowa

Aplikacje zostały sprawdzone pod względem szybkości wykonania operacji, zużycia pamięci RAM oraz procesora. Jedynie aplikacja wspomagająca pracę kierownika została zbadana tylko pod względem wykorzystania pamięci RAM i procesora.

Aplikacje służące do generowania zarówno pól tekstowych jak i przycisków mają za zadanie wygenerować podczas uruchomienia 1000 kontrolek. Podczas generowania zostanie mierzony czas tej operacji. Kolejnym zadaniem jest generowanie 1000, 2000 i 5000 kontrolek po naciśnięciu odpowiedniego przycisku. Czas generowania tych operacji również jest mierzony.

Następna aplikacja ma za zadanie wyświetlić listę 1000-elementową. Lista jest generowana w momencie uruchomienia programu, a następnie przekazywana do kontrolki widoku listy (ListView) w celu wyświetlenia ich.W tym przypadku sam czas wyświetlania jest liczony.

W ostatniej aplikacji wspomagającą pracę kierownika jest badane zużycie procesora oraz pamięci RAM. Wykonywane są jej podstawowe funkcjonalności, czyli logowanie się na konta użytkownika, dodawanie/edytowanie pracownika, dodawanie nowych zadań lub projektów.

Do testowania aplikacji wykorzystano sprzęt o parametrach podanych w tabeli 1. Testowanie aplikacji odbywało się przy minimalnym obciążeniu sprzętu przez inne zbędne aplikacje na czas testowania, między innymi program antywirusowy został wyłączony.

Tabela 1. Parametry sprzętu badawczego

Nazwa	Dell Inspirion 3542
System operacyjny	Windows 8.1
Procesor	Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz
Pamięć RAM	4 GB
Dysk	SSD

Do lepszego zbadania wykorzystania podzespołów przez aplikację posłużono się programem Process Explorer [12]. Jest to program podobny do zwykłego menadżera zadań z większym zakresem możliwości. Pozwala lepiej monitorować zachowanie danego procesu.

6. Wyniki badań

Badania zostały przeprowadzone głównie pod kątem zużycia pamięci RAM, procesora oraz szybkości wykonania danej operacji. Pierwsze wyniki badań dotyczą aplikacji testowej wspomagającej pracę kierownika, gdzie badane było tylko zużycie procesora i pamięci RAM. Zostały one zaprezentowane w tabeli 2.

Tabela 2. Obciążenie systemu przy działającej aplikacji

Nazwa technologii	**Obciążenie procesora przez aplikację (%)**	**Obciążenie pamięci RAM przez aplikację (MB)**
WPF	12,30	68
WinForms	9,19	28,5

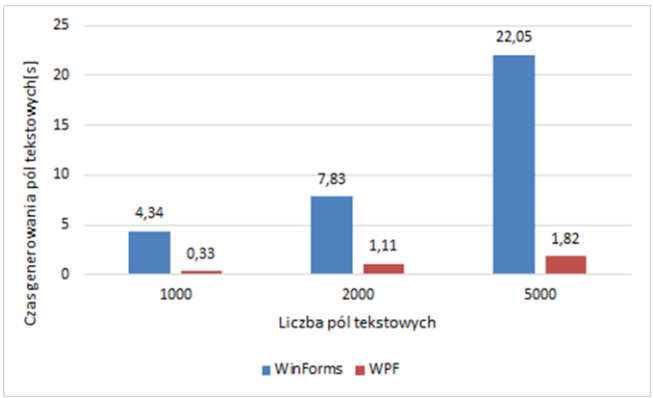
Wyniki uzyskane podczas generowania kontrolek pola tekstowego i przycisku zostały zebrane w Tabeli 3 i 4. Badane było obciążenie procesora, pamięci RAM i szybkość wykonanej operacji przy generowaniu 1000 kontrolek przy uruchomieniu aplikacji oraz wygenerowaniu 1000, 2000 i 5000 kontrolek już przy działającej aplikacji. Na rysunkach od 1 do 6 przedstawiono wykresy porównawcze uśrednionych wyników dla generowania kontrolek przy uruchomionym już programie.

Tabela 3. Średnie wyniki przy generowaniu 1000 Pól tekstowych przy uruchomieniu aplikacji WPF i WinForms

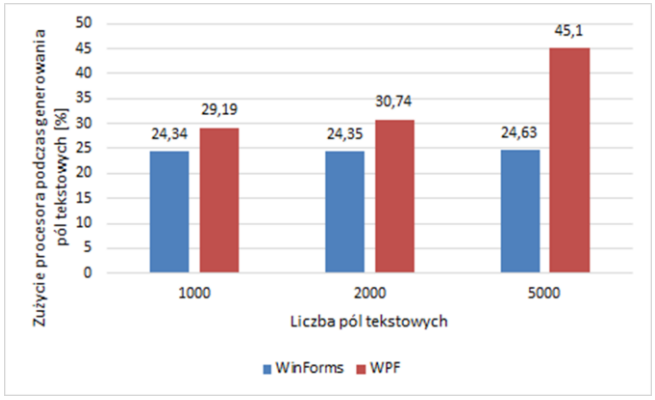
Nazwa technologii	**Obciążenie procesora przez aplikację (%)**	**Obciążenie pamięci RAM przez aplikację (MB)**	**Czas wygenerowania Pól tekstowych (s)**
Średnia dla WPF	29,12	65,96	0,33
Odchylenie standardowe dla WPF	±2,31	±0,11	±0,02
Średnia dla WinForms	23,24	17,48	0,55
Odchylenie standardowe dla WinForms	±0,99	±0,33	±0,06

Tabela 4. Średnie wyniki przy generowaniu 1000 przycisków przy uruchomieniu aplikacji WPF i WinForms

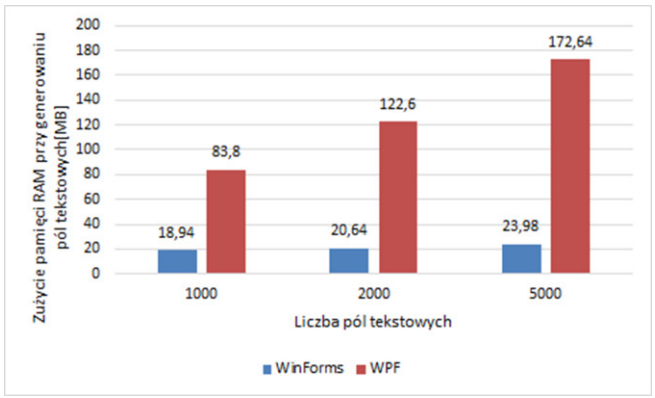
Nazwa technologii	**Obciążenie procesora przez aplikację (%)**	**Obciążenie pamięci RAM przez aplikację (MB)**	**Czas wygenerowania przycisków (s)**
Średnia dla WPF	18,38	49,82	0,09
Odchylenie standardowe dla WPF	±4,32	±0,22	±0,0
Średnia dla WinForms	14,17	14,14	0,31
Odchylenie standardowe dla WinForms	±2,81	±0,11	±0,01



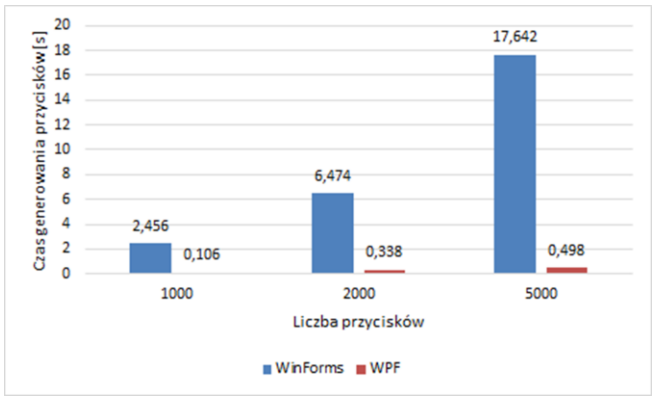
Rys. 1. Średni czas generowania Pól tekstowych



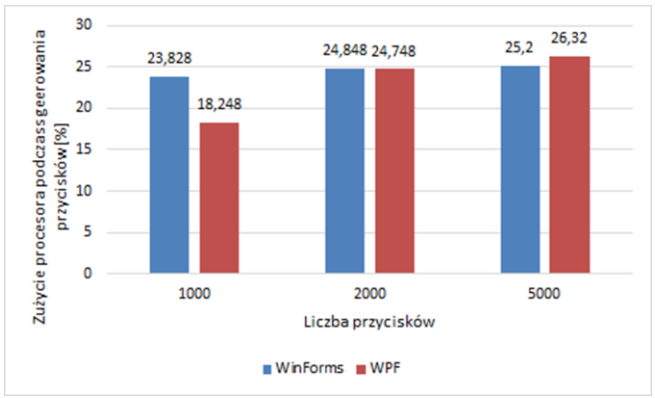
Rys. 2. Średnie zużycie procesora przy generowaniu Pól tekstowych



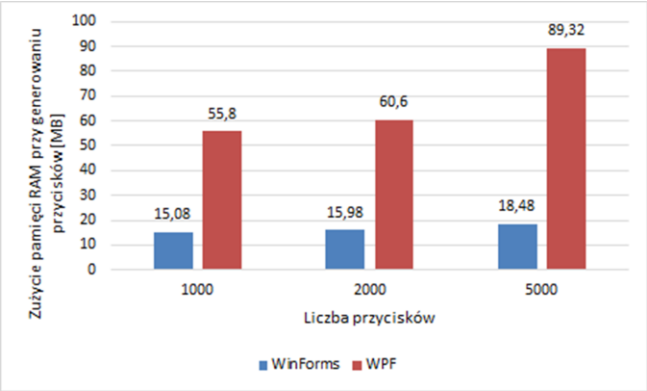
Rys. 3. Średnie zużycie pamięci RAM przy generowaniu Pól tekstowych



Rys. 4. Średnie czas generowania przycisków



Rys. 5. Średnie zużycie procesora przy generowaniu przycisków



Rys. 6. Średnie zużycie pamięci RAM przy generowaniu przycisków

Ostatnie wyniki dotyczą wyświetlania 1000 elementowej listy w komponencie ListView przy uruchomieniu aplikacji, gdzie sprawdzono czas wygenerowania listy oraz zużycie procesora i pamięci RAM. Uzyskane wyniki zgromadzono w tabeli 5.

Tabela 5. Średnie wyniki przy wyświetleniu 1000 elementowej listy w widoku listy (ListView) podczas uruchomienia aplikacji

Nazwa technologii	**Obciążenie procesora przez aplikację (%)**	**Obciążenie pamięci RAM przez aplikację (MB)**	**Czas wygenerowania listy w ListView (s)**
Średnia dla WPF	4,22	40,44	0,0136
Odchylenie standardowe dla WPF	±2,94	±0,17	±0,0
Średnia dla WinForms	22,88	13,7	0,0136
Odchylenie standardowe dla WinForms	±0,12	±0,0	±0,0

7. Dyskusja wyników

Na podstawie tabeli 2 wyraźnie widać przewagę WinForms nad WPF. Zużycie procesora przez WPF jest o 1/3 większy niż w WinForms i aż dwa razy większe jest zużycie pamięci RAM.

W tabeli, 3 jaki i 4 można zaobserwować większe zużycie pamięci RAM i procesora przez WPF. O ile przy zużyciu procesora różnica jest niewielka i wynosi tylko kilka procent, to przy pamięci RAM jest ona ponad trzykrotna. Jeśli chodzi o czas wykonania tej operacji to iw jednym i drugim przypadku lepiej wypadł WPF. WinForms przegrał tylko, jeśli chodzi o czas operacji dla przycisku i pola tekstowego wynosi 0,22s. W podobnych badaniach, jeśli chodzi o czas i zużycie RAM można zaobserwować podobne wyniki, jednak tam uwzględniono tylko jedną kontrolkę Pola tekstowego i nie wzięto pod uwagę zużycia procesora [13].

Badanie generowania kontrolek przy uruchomionym programie ukazują zużycie pamięci RAM, procesora i czasu operacji w zależności od liczby generowanych kontrolek. Na rysunku 1 i 4 widać przewagę technologii WPF w stosunku do WinForms. Przy 1000 kontrolkach WinForms jest wolniejszy w generowaniu tylko o 2 sekundy w przypadku generowania pola tekstowego i o 4 sekundy dla przycisku. Następnym obszarem porównania jest zużycie procesora. W tym przypadku to WinForms jest lepszy przy generowaniu Pola tekstowego. Zużycie zostaje prawie niezmienne w porównaniu do WPF, gdzie wyraźnie widać, że wraz ze wzrostem liczby generowanych ikonek zużycie procesora rośnie. Ciekawym przypadkiem jest generowanie przycisku, gdzie można zaobserwować przewagę technologii WPF dla 1000 kontrolek. Następnie przy 2000 kontrolek zużycie procesora prawie się zrównało. Na końcu dla 5000 kontrolek to WinForms okazuje się lepszy. W WinForms zużycie niemal pozostaje na tym samym poziomie nieznacznie wzrastając jak dla Pola tekstowego. WPF podobnie jak dla Pola tekstowego widać wzrost zużycia procesora w zależności od liczby wygenerowanych komponentów. Na rysunku 3 i 6 przedstawione zostało zużycia procesora. Widoczna jest przewaga technologii WinForms. Niezależnie od liczby wygenerowanych komponentów wzrost zużycia jest dosyć mały w porównaniu do wzrostów zużycia przez WPF. WPF zużywa dużo więcej pamięci oraz posiada większą różnicę między generowanymi kontrolkami w stosunku do WinForms.

W tabeli 5 można zaobserwować przewagę WPF pod względem mniejszego zużycia procesora, a WinForms ma mniejsze zużycia pamięci RAM. Natomiast czas operacji jest równy dla obu technologii.

WPF osiągnął lepsze wyniki przy zużyciu procesora dla generowaniaprzycisku 1000 i 2000 razy w porównaniu do generowania kontrolki Pola tekstowego. Może to być spowodowane pracą środowiska testowego lub czynnikami niezależnymi od aplikacji.

8. Wnioski

Praca aplikacji w dużej mierze zależy od środowiska, na którym jest uruchamiana. Generowanie dużej liczby kontrolek w aplikacji zawsze obciąża słabsze środowisko uruchomieniowe. Dlatego większość aplikacji desktopowych rezygnuje z dużej liczby kontrolek wyświetlanych w jednym oknie. Zamiast tego wybiera się mniejszą liczbę kontrolek i możliwości przejścia między oknami aplikacji.

Na podstawie przeprowadzonych badań można wywnioskować, że WinForms sprawdza się lepiej pod względem zużycia pamięci RAM i procesora. Zużycie ich jest stosunkowo małe w porównaniu do WPF. Podczas generowania komponentów w WinFormsmniej obciąża środowisko uruchomieniowe. Obciążenie to utrzymuje się na podobnym poziomie niezależnie od ich liczby. WPF, natomiast sprawdza się lepiej, jeśli chodzi o szybkość. We wszystkich badanych operacjach, oprócz wczytywania listy, był szybszy od WinForms. Podsumowując badania, WinForms znacznie lepiej sprawdzałby się w aplikacjach, w których zależy użytkownikowi na mniejszym zużyciu procesora, jak i pamięci RAM. WPF sprawdza się lepiej, jeśli w aplikacji chodzi o szybkość wykonania danego zadania.

Literatura

[1] Nora Georgieva, Survey Report: Who is the .NET Developer of 2016?, https://www.telerik.com/blogs/survey-report-the-dotnet-developer-of-2016 [06.10.19]

[2] Developer Survey Results 2018, https://insights.stackoverflow.com/survey/2018/ [06.10.2019]
[3] Developer Survey Results 2017, https://insights.stackoverflow.com/survey/2017/ [06.10.2019]
[4] Anurag Misra, Use of Windows Presentation Foundation and Windows Forms in Windows Application Programming, Volume 7, No. 7, Nov-Dec 2016 International Journal of Advanced Research in Computer Science
[5] Winforms vs WPF, https://www.educba.com/winforms-vs-wpf/ [06.10.2019]
[6] WPF vs. WinForms, https://www.wpf-tutorial.com/pl/2/o-wpf/wpf-vs-winforms/ [06.10.2019]
[7] Windows Presentation Foundation (WPF), Margaret Rouse, https://searchwindevelopment.techtarget.com/definition/Windows-Presentation-Foundation [06.10.2019]
[8] Wei-Meng Lee, An Overview of Windows Presentation Foundation, Codemagazine 03/04 2006,
[9] Paul D. Sheriff, Why Use WPF?, Codemagazine 11/12 2009
[10] Sells, Chris, Windows Forms Programming in C# (1st ed.). Addison-Wesley Professional. p. xxxviiii.
[11] https://www.worddisk.com/wiki/WinForms/ [06.10.2019]
[12] Mark Russinovich, Process Explorer v16.30, https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer [06.10.2019]
[13] yashsoman, Window Form Controls v/s WPF Controls Memory Comparison, http://www.codeproject.com/Articles/1038077/Window-Form-Controls-v-s-WPF-Controls-MemoryCompa [06.10.2019]