DOI: 10.5604/20830157.1148047

WYDAJNOŚĆ IMPLEMENTACJI PODSTAWOWYCH METOD CAŁKOWANIA W ŚRODOWISKU APP INVENTOR

Kamil Żyła
Politechnika Lubelska, Instytut Informatyki

Streszczenie. *W niniejszym artykule dokonano porównania wydajności podstawowych metod całkowania zaimplementowanych w środowisku App Inventor oraz Java dla platformy Android. Wybrane metody (prostokątów, trapezów i Simpsona) zastosowano dla funkcji liniowej, sześciennej oraz sinusoidy. Rezultaty eksperymentu wykazały, że działanie algorytmów zaimplementowanych w App Inventor jest wielokrotnie wolniejsze niż w Java dla Android, co znacząco ogranicza przydatność środowiska App Inventor do tworzenia aplikacji realizujących obliczenia matematyczne.*

Słowa kluczowe: przetwarzanie mobilne, inżynieria sterowana modelami, MDE, Android, App Inventor, całkowanie numeryczne

IMPLEMENTATION OF BASIC INTEGRATION METHODS IN APP INVENTOR ENVIRONMENT AND THEIR EFFICIENCY

Abstract. *This paper presents comparison of efficiency of basic integration methods implemented in App Inventor and Java for Android environment. Chosen methods (rectangle, trapezoidal and Simpson's rules) were applied for linear, cubic and sine functions. Conducted experiment revealed that applications developed in App Inventor were significantly slower than in case of Java, which makes App Inventor unsuitable for applications involving intensive calculations.*

Keywords: mobile computing, model-driven engineering, MDE, Android, App Inventor, numerical integration

Wstęp

W niniejszym artykule poruszono kwestię wydajności implementacji podstawowych metod całkowania w środowisku App Inventor oraz Java dla platformy Android, celem określenia przydatności środowiska App Inventor do tworzenia aplikacji realizujących operacje matematyczne. W związku z tym przeprowadzono eksperyment polegający na pomiarze czasu całkowania numerycznego funkcji liniowej, sześciennej i sinus, przy pomocy metody prostokątów, trapezów i Simpsona.

Środowisko App Inventor jest narzędziem bliskim inżynierii sterowanej modelami, służącym do modelowania aplikacji mobilnych. [15] podaje, że 1,9 miliona użytkowników ze 195 krajów wykonało modele 4,7 miliona aplikacji. Modele te są, póki co, przekształcane do aplikacji działających jedynie na platformie Android, stąd porównanie wydajności z Java dla Android.

Głównym celem istnienia środowiska App Inventor jest włączenie możliwie szerokiej grupy osób (w tym dzieci i młodzieży) w proces tworzenia aplikacji, poprzez obniżenie bariery trudności technologii [5]. Stąd też do grupy jego adresatów można zaliczyć inżynierów (nie posiadających wiedzy programistycznej niezbędnej do klasycznego programowania aplikacji mobilnych), którzy potrzebowaliby tworzyć własne programy zawierające elementy obliczeń. W związku z tym warto sprawdzić, jak kształtuje się wydajność implementacji podstawowych algorytmów całkowania w App Inventor, jako że algorytmy te zawierają szereg często używanych operacji matematycznych oraz konstrukcji programistycznych. Jest to również konfrontacja dwóch podejść do tworzenia aplikacji – z punktu widzenia projektanta i programisty.

Kolejnym argumentem jest istotna przewaga publikacji traktujących o aspektach użytkowych – m.in. wykorzystaniu w dydaktyce, szybkim prototypowaniu, budowie aplikacji. W dodatku skupiają się one głównie na zaletach, rzadziej są wymieniane wady środowiska, a także braki w jego funkcjonalności. Kwestie wydajności generowanych aplikacji są na dalszym planie, co można wnioskować chociażby na podstawie wyników wyszukiwania hasła „App Inventor", zwróconych przez wyszukiwarki wydawnictw: IEEE, Springer, ACM oraz Science Direct.

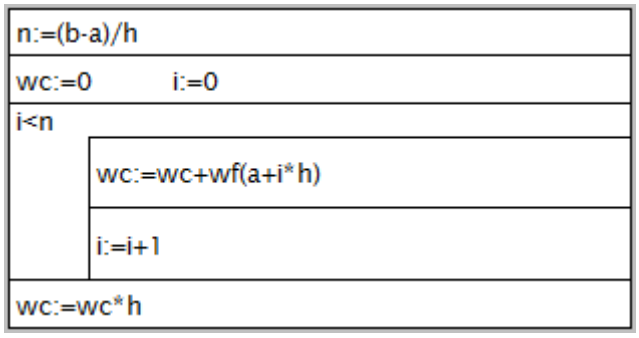
1. Wprowadzenie do metod całkowania

W ramach eksperymentu wykorzystano następujące metody całkowania [3]:

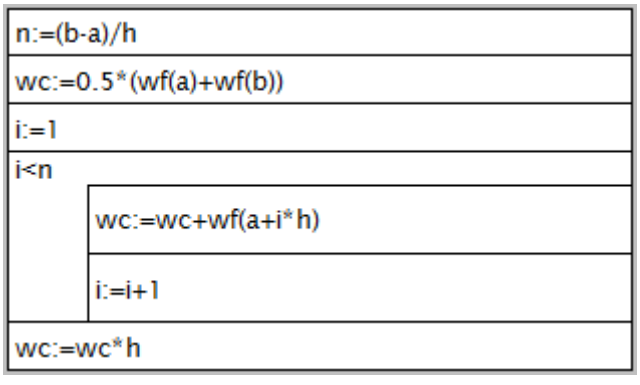
1) Metoda prostokątów (rys. 1) – funkcja podcałkowa jest przybliżana funkcjami stałymi (jest liczona suma pól prostokątów).
2) Metoda trapezów (rys. 2) – funkcja podcałkowa jest przybliżana funkcjami liniowymi (jest liczona suma pól trapezów).
3) Metoda Simpsona (rys. 3) – funkcja podcałkowa jest przybliżana parabolami rozpiętymi na podprzedziałach całkowania.

Schematy zwarte NS (rys. 1-3) przedstawiające ideę poszczególnych algorytmów całkowania (pseudokod jest zgodny z językiem Pascal) wykonano w programie NS Builder opracowanym przez Aleksandra Wojdygę. Na rzeczonych schematach użyto następujących oznaczeń:

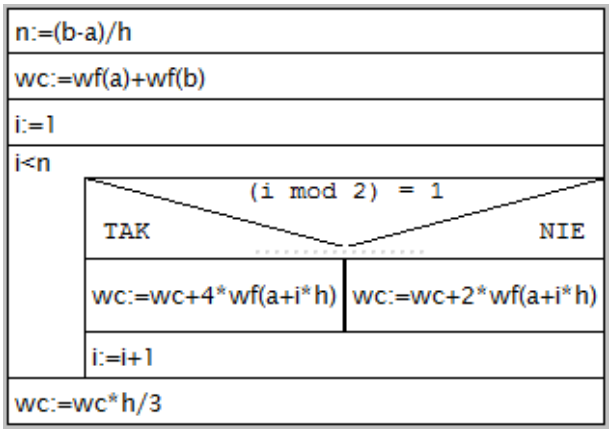
- *a* – dolna granica całkowania,
- *b* – górna granica całkowania,
- *h* – długość podprzedziału całkowania,
- *n* – liczba podprzedziałów całkowania,
- *wc* – przybliżona wartość całki w zadanych granicach,
- *wf(x)* – wartość funkcji podcałkowej w punkcie *x*,
- *i* – zmienna sterująca pętlą.



Rys. 1. Schemat zwarty NS przedstawiający metodę prostokątów



Rys. 2. Schemat zwarty NS przedstawiający metodę trapezów



Rys. 3. Schemat zwarty NS przedstawiający metodę Simpsona

2. Wprowadzenie do środowisk implementacji

Klasyczna metoda tworzenia aplikacji na platformę Android polega na wykorzystaniu dedykowanego jej środowiska deweloperskiego np. środowiska Eclipse w połączeniu z wtyczką ADT (ang. Android Developer Tools) oraz Android SDK (ang. Software Development Kit). Rzeczone narzędzia bazują na dedykowanej platformie Java, w związku z tym, ze względu na duże podobieństwo, nie stanowią znaczącej bariery technologicznej dla posługujących się nią programistów [2].

Typowy projekt aplikacji składa się m.in. z [2]:

- manifestu – pliku XML (ang. Extensible Markup Language) zawierającego informacje o aplikacji potrzebne do jej prawidłowego uruchomienia przez system Android,
- plików źródłowych - plików z klasami/instrukcjami języka Java opisujących działanie aplikacji,
- importowanych bibliotek – pogrupowanych tematycznie zbiorów predefiniowanych konstrukcji programistycznych wykorzystywanych do realizacji celu aplikacji,
- zasobów – plików multimedialnych, plików określających wygląd ekranów aplikacji, itp.

Całość projektu jest kompilowana do pliku .apk (ang. Android Package), który można rozpowszechniać (np. przez Google Play) i instalować na urządzeniach Android – wirtualnych (emulator) lub rzeczywistych (np. tablet) [12].

Przykładowy kod funkcji dla platformy Android, realizujący całkowanie metodą Simpsona, zamieszczono poniżej (obowiązują oznaczenia z rys. 1-3):

```
public static double mSimpsona(...)
   {
      double n = (b - a) / h;
      double wc = wf(a)+wf(b);

      for (int i = 1; i < n; i++) {
         if (i % 2 == 1) {
            wc += 4*wf(a + i*h);
         }
         else {
            wc += 2*wf(a + i*h);
         }
      }
      wc *= h/3.0;

      return wc;
   }
```

W przypadku App Inventor dostęp do środowiska deweloperskiego jest możliwy przy pomocy przeglądarki stron internetowych po zalogowaniu się na konto Google. Na lokalnym hoście, po zainstalowaniu SDK, odbywa się testowanie aplikacji z użyciem urządzenia Android (rzeczywistego bądź wirtualnego). Natomiast przechowywanie i kompilowanie projektów do plików .apk jest realizowane przez zewnętrzny serwer.

Pierwotnie środowisko App Inventor było rozwijane i udostępniane przez Google, które zadecydowało o zaprzestaniu wsparcia z końcem 2011 roku. [14] Do jego uratowania przyczyniło się żywotne zainteresowanie zgromadzonej wokół niego społeczności, a także otworzenie nowego centrum przez MIT (ang. Massachusetts Institute of Technology), który do tej pory jest odpowiedzialny za jego rozwijanie. [16]

W App Inventor, podobnie jak w klasycznej metodzie, proces tworzenia aplikacji polega na zaprojektowaniu wyglądu aplikacji (narzędzie „Designer") oraz określeniu sposobu jej działania (narzędzie „Blocks Editor"). Projektowanie polega na przeciąganiu z palety, upuszczaniu na obszar roboczy i konfigurowaniu elementów interfejsu użytkownika. Specyfika środowiska App Inventor ujawnia się szczególnie podczas drugiego etapu, gdzie zamiast pisania kodu, działanie aplikacji opisuje się (wykorzystując metodę przeciągnij i upuść) przy pomocy puzzli reprezentujących podstawowe struktury programistyczne. Można je uznać za swego rodzaju uproszczoną część wspólną języków programowania 3 generacji, co jest częściowo widoczne na rys. 4 (obowiązują oznaczenia z rys. 1-3).

Klasyczna metoda tworzenia aplikacji prowadzi do rozwiązań wydajnych, a przy tym jest sprawdzona, powszechna, elastyczna, wspierana przez doskonałe narzędzia oraz dużą społeczność profesjonalistów. Jednakże towarzyszy jej wyższa bariera technologiczna. App Inventor jest swego rodzaju fenomenem na jej tle. Dawid Wolber w [11] wskazuje na łatwość obsługi i zdolność środowiska do tworzenia niemal nieograniczonego wachlarza aplikacji. Należy jednak powiedzieć wprost, że ta zdolność jest istotnie ograniczona przez bieżący stan narzędzia. Największym ograniczeniem jest predefiniowana paleta komponentów i puzzli, która utrudnia, a niekiedy uniemożliwia, realizację niestandardowej funkcjonalności. Towarzyszą temu braki w funkcjonalności edytora (słusznie wymienione w [1]), a uproszczenia w strukturach programistycznych prowadzą niekiedy do nadmiernej komplikacji. Ponadto zdarza się, że rozmiar modelu utrudnia jego odczytanie albo jego implementacja w klasycznym języku programowania zajęłaby mniej czasu. Niemniej środowisko App Inventor jest bardzo pozytywnie odbierane, o czym świadczą chociażby:

1) publikacje z dziedziny dydaktyki, np. [4, 6, 8, 9],
2) próby profesjonalnych zastosowań, np. [7, 10],
3) zaangażowanie MIT i społeczności, np. [13],
4) opinie studentów o początkującej wiedzy programistycznej, pojawiające się podczas zajęć prowadzonych w ramach Instytutu Informatyki Politechniki Lubelskiej.

3. Organizacja eksperymentu

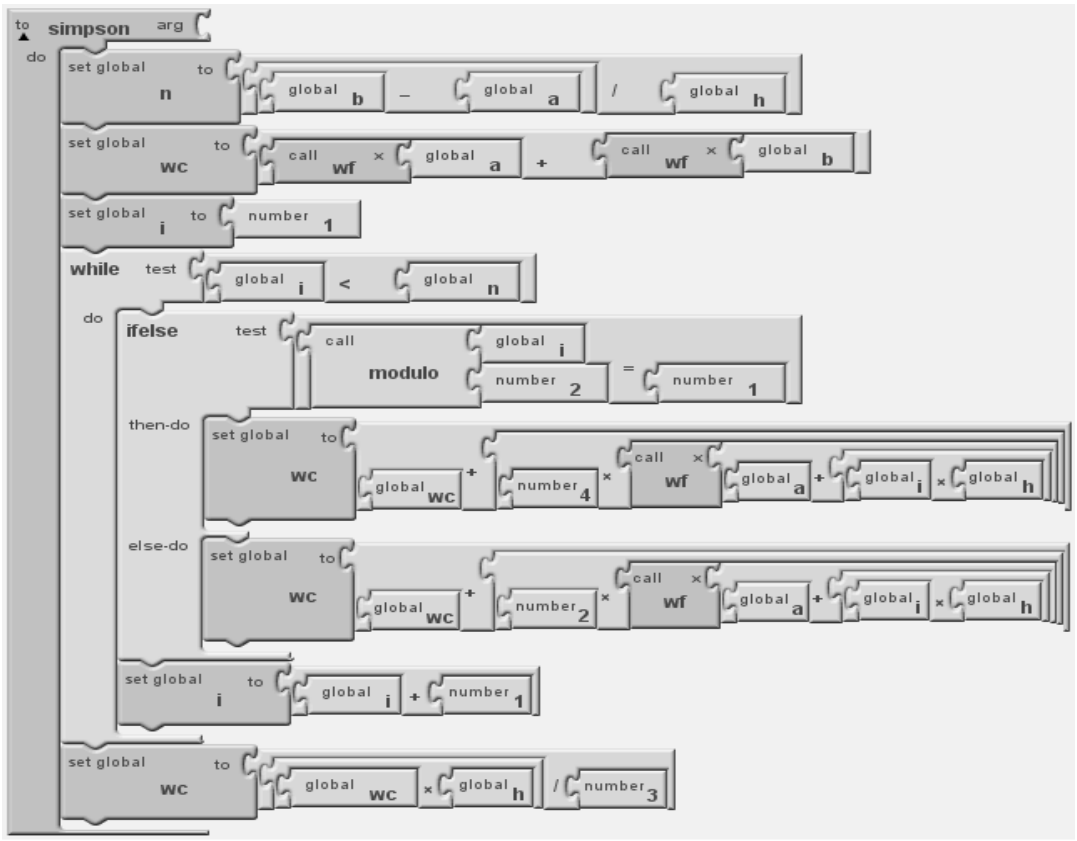
Przeprowadzony eksperyment polegał na pomiarze czasu obliczenia całek oznaczonych kolejnych funkcji:

- liniowej – $ax+b$,
- sześciennej – ax^3+bx^2+cx+d,
- sinus – $csin(ax+b)$,

przy pomocy kolejnych algorytmów:

- metoda prostokątów,
- metoda trapezów,
- metoda Simpsona.

Stworzono dwie aplikacje, o podobnej strukturze (w miarę możliwości technologii), będące wierną implementacją algorytmów przedstawionych na rysunkach 1-3. Pierwsza z nich została wykonana przy pomocy środowiska App Inventor, druga przy pomocy Java dla Android. Obydwie implementowały algorytmy całkowania w ten sam sposób, czego przykładem jest rys. 4 oraz zamieszczony kod funkcji, gdzie realizowano całkowanie metodą Simpsona. Użyto środowiska App Inventor 1, SDK 1.2, kompilacja do pliku .apk nastąpiła 26 lipca 2014 (podano datę, ponieważ doświadczenie ze środowiskiem uczy, że aktualizacje bywają transparentne dla użytkownika). W przypadku Java dla Android, użyto środowiska Eclipse Kepler SR1, ADT 22.3.0, Android SDK Tools 22.3 i Java 1.7u60.



Rys. 4. Impementacja metody Simpsona w App Inventor Blocks Editor

Pomiar czasu, w przypadku Java dla Android, polegał na wyznaczeniu różnicy wartości zwróconych przez funkcję *System.currentTimeMillis()* wywołaną zaraz przed i zaraz po wywołaniu metody obliczającej wartość całki. W przypadku App Inventor, w zmiennych globalnych zapamiętywano rezultat wywoływania metody *Now* komponentu *Clock* zaraz przed i zaraz po wywołaniu procedury obliczającej wartość całki. Liczba milisekund była wyznaczana przy pomocy funkcji *Duration* komponentu *Clock*, która zwraca liczbę milisekund pomiędzy dwoma chwilami w czasie [15]. W obydwu przypadkach, podczas pomiaru czasu nie były realizowane operacje wejścia/wyjścia.

Za właściwy czas obliczenia całki przyjmowano medianę z 5 powtórzeń wywołania metody całkującej. Miało to na celu ograniczenie wpływu skoków chwilowego obciążenia środowiska testowego, powodowanych przez pozostałe procesy działające w systemie operacyjnym, na osiągnięte rezultaty.

Obydwie aplikacje zostały zainstalowane i pojedynczo uruchamiane na tym samym urządzeniu Android – tablet Manta DUO POWER 3G GPS MID 713 o następujących parametrach:

1) CPU: Dual Core 2 x 1,2 GHz,
2) RAM: 512 MB,
3) OS: Android 4.2.

Tablet pracował z dostępną pełną mocą obliczeniową – wyłączono tryb oszczędzania energii.

4. Rezultaty eksperymentu

Tabela 1 zawiera czasy w milisekundach, jakie były potrzebne do obliczenia wartości całek oznaczonych. Do obliczeń przyjęto następujące wartości współczynników (układ współczynników jak w opisie organizacji eksperymentu): a=3; b=1,4; c=15,6; d=8. Przedział, w jakim były całkowane funkcje, to [0,3], natomiast długość podprzedziału całkowania wynosiła 0,0006, co dało 5000 podprzedziałów całkowania.

Wydajność aplikacji napisanej w Java dla Android okazała się miażdżąca, w porównaniu do aplikacji stworzonej w środowisku App Inventor. Widać to na rys. 5, gdzie przedstawiono ile razy aplikacja w Java dla Android była szybsza – wartości rzędu tysięcy razy. Powodu tak ogromnego dysonansu wydajności należałoby się dopatrywać w jakości procesu generacji (zamiany modelu wykonanego w przeglądarce na plik .apk) w środowisku App Inventor, co jest o tyle dziwne, że opis działania aplikacji w Blocks Editor jest bardzo podobny do klasycznych języków programowania.

Znaczący wpływ na różnice w czasach obliczania wartości całek miała liczba wykonanych operacji matematycznych w jednym obrocie pętli, wynikających nie tylko z metody całkowania, ale również złożoności funkcji, której wartość w punkcie była wyznaczana. Stąd też najdłuższe czasy odnotowano w przypadku funkcji sześciennej.

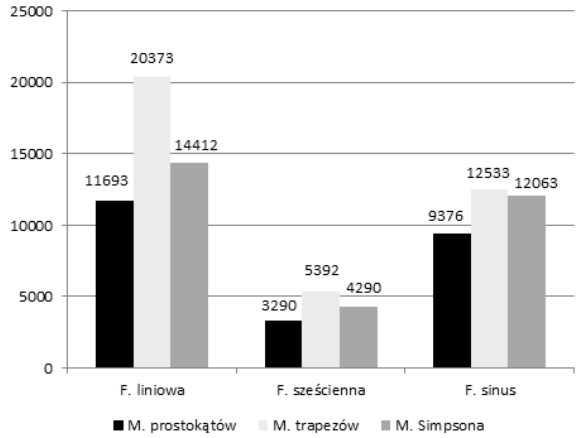
Średnie odchylenie od mediany, liczone dla wszystkich pomiarów, dla Java dla Android wyniosło 0,47 ms, natomiast dla App Inventor 36,56 ms.

Tabela 1. Czasy obliczeń całek dla h=0,0006 (podane w milisekundach)

Metoda prostokątów		
Funkcja	**Java dla Android**	**App Inventor**
Liniowa	2	23 387
Sześcienna	10	32 906
Sinus	3	28 129
Metoda trapezów		
Funkcja	**Java dla Android**	**App Inventor**
Liniowa	3	40 747
Sześcienna	11	59 319
Sinus	4	50 133
Metoda Simpsona		
Funkcja	**Java dla Android**	**App Inventor**
Liniowa	2	28 824
Sześcienna	10	42 907
Sinus	3	36 190

Kolejny test polegał na sprawdzeniu, jakie rozbieżności w wydajności testowanych środowisk pojawią się wraz ze zmianą liczby podprzedziałów całkowania. Wartości współczynników i granice przedziału całkowania nie uległy zmianie. W tabeli 2 i na rys. 6 przedstawiono różnice pomiędzy czasem uzyskanym przez implementacje algorytmów w App Inventor a Java

dla Android. Można stwierdzić, że wraz ze wzrostem liczby podprzedziałów, a zarazem liczby operacji matematycznych do wykonania, różnica czasu ulega znacznemu zwiększeniu, tzn. App Inventor coraz bardziej odstaje od Java dla Android.

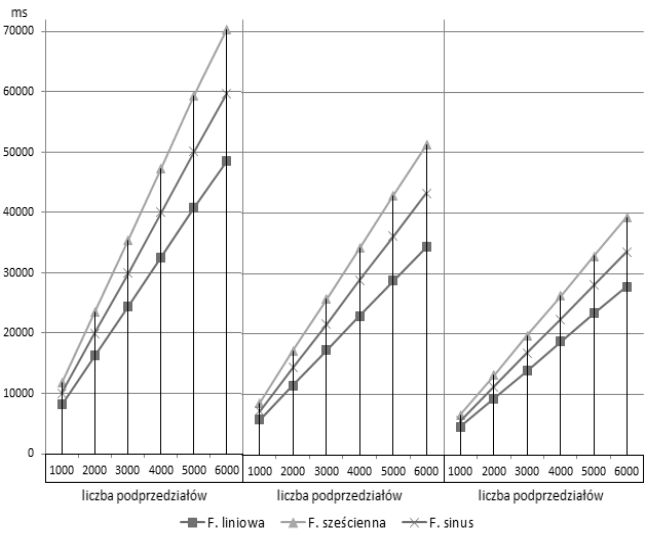


Rys. 5. Stosunek szybkości aplikacji wykonanej w App Inventor do aplikacji wykonanej w Java dla Android

Dopiero zmniejszenie liczby podprzedziałów całkowania do 100 pozwoliło uzyskać czasy poniżej 1300 ms (niezależnie od funkcji i metody całkowania), w przypadku aplikacji wykonanej w App Inventor.

Tabela 2. Różnice w czasie obliczenia całek, pomiędzy App Inventor a Java dla Android, w zależności od liczby podprzedziałów n (podane w milisekundach)

Metoda prostokątów						
Funkcja\n	**1 000**	**2 000**	**3 000**	**4 000**	**5 000**	**6 000**
Liniowa	4 677	9 305	13 935	18 733	23 385	27 806
Sześcienna	6 624	13 146	19 729	26 301	32 895	39 291
Sinus	5 612	11 319	16 839	22 472	28 126	33 523
Metoda trapezów						
Funkcja\n	**1 000**	**2 000**	**3 000**	**4 000**	**5 000**	**6 000**
Liniowa	8 158	16 263	24 353	32 550	40 745	48 524
Sześcienna	11 827	23 596	35 346	47 230	59 308	70 408
Sinus	10 006	19 992	29 961	40 018	50 129	59 787
Metoda Simpsona						
Funkcja\n	**1 000**	**2 000**	**3 000**	**4 000**	**5 000**	**6 000**
Liniowa	5 786	11 492	17 294	22 988	28 822	34 385
Sześcienna	8 601	17 194	25 724	34 234	42 896	51 231
Sinus	7 258	14 459	21 610	28 887	36 187	43 297



Rys. 6. Różnice w czasie obliczenia całek pomiędzy App Inventor a Java dla Android – od lewej metoda trapezów, Simpsona i prostokątów

5. Podsumowanie

Otrzymane podczas eksperymentu czasy należy traktować orientacyjnie, gdyż na ich wartość wpływa chociażby obciążenie chwilowe środowiska testowego nie związane z testowaną aplikacją. Niemniej są wystarczająco dokładne, aby stwierdzić, że działanie algorytmów zaimplementowanych w App Inventor jest wielokrotnie wolniejsze od algorytmów zaimplementowanych w środowisku Java dla Android. Implikuje to, że należy unikać App Inventor jako środowiska tworzenia aplikacji wykonujących intensywne obliczenia matematyczne, ze względu na ich niską wydajność. Jako odpowiedzialną za taki stan rzeczy należałoby wskazać konstrukcję generatora dołączonego do środowiska App Inventor, niemniej wymaga to dalszych badań. Sytuacja jest o tyle dziwna, że obydwie implementacje korzystają z podobnych konstrukcji programistycznych, a język modelowania zachowania aplikacji wydaje się być dobrą podstawą do generacji wydajnego kodu.

Literatura

[1] Chadha K.: Improving the usability of App Inventor through conversion between blocks and text. Undergraduate thesis. Wellesley College, 2014.
[2] Conder S., Darcey L.: Android wireless application development. Addison-Wesley, 2011.
[3] Dahlquist G., Bjorck A.: Metody numeryczne. PWN, Warszawa 1983.
[4] Grover S., Pea R.: Using a discourse-intensive pedagogy and android's App Inventor for introducing computational concepts to middle school students. Proceedings of the 44th ACM technical symposium on Computer science education. ACM, New York 2013.
[5] Jordan L., Greyling P.: Practical Android Projects. Apress, 2011.
[6] Karakus M., Uludag S., Guler E., Turner S.W., Ugur A.: Teaching computing and programming fundamentals via App Inventor for Android. 2012 International Conference on Information Technology Based Higher Education and Training (ITHET). IEEE, 2012.
[7] Kepley S.: Rapid development of mobile apps using App Inventor and AGCO API. Master of Science thesis. Kansas State University, 2014.
[8] Morelli R., de Lanerolle T., Lake P., Limardo N., Tamotsu E., Uche C.: Can Android App Inventor bring computational thinking to K-12?. Proceedings of the 42nd ACM technical symposium on Computer science education. ACM, New York 2011.
[9] Roy K., Rousse W.C., DeMeritt D.B.: Comparing the mobile novice programming environments: App Inventor for Android vs. GameSalad. Frontiers in Education Conference (FIE), 2012. IEEE, 2012.
[10] Sung W.-T., Lin J.-S.: Design and implementation of a smart LED lighting system using a self adaptive weighted data fusion algorithm. Sensors 13/2013, s. 16915-16939.
[11] Wolber D., Abelson H., Spertus E., Looney L.: App Inventor: Create your own Android apps. O'Reilly Media, 2011.
[12] developer.android.com, dostęp 04.09.2014r.
[13] appinventorblog.com, dostęp 04.09.2014r.
[14] appinventorblog.com/2011/08/09/app-inventor-discontinued-the-good-the-bad-and-the-ugly/, dostęp 04.09.2014r.
[15] appinventor.mit.edu, dostęp 07.09.2014r.
[16] readwrite.com/2011/08/16/mit_launches_center_for_mobile_learning_with_suppo, dostęp 04.09.2014r.

Mgr inż. Kamil Żyła
e-mail: k.zyla@pollub.pl

Asystent w Instytucie Informatyki Politechniki Lubelskiej, zajmuje się inżynierią sterowaną modelami, projektowaniem systemów informatycznych oraz zagadnieniami przetwarzania mobilnego.

otrzymano/received: 27.10.2014 *przyjęto do druku/accepted: 28.12.2014*