Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki

Praca dyplomowa magisterska

PROGRAMOWANIE WIZUALNE URZĄDZEŃ MOBILNYCH

Jakub Bręk

Promotor
Rafał Różycki, Dr. Hab.

Poznań, 2014 r.

Tutaj przychodzi karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

Rozdział 1

Wprowadzenie

1.1 Programowanie

Programowanie jest to między innymi proces tworzenia aplikacji. Jest ono głównie kojarzone z wielkimi ilościami kodu napisanego przez programistów. W większości przypadków tak jest. Jednak sposób pisania kodu może się różnić. Można wydzielić programowanie od bardzo niskiego poziomu do wysokiego poziomu. Przy językach niskiego poziomu, np. Assembler operowanie odbywa się na rejestrach procesora. Operacje są bardzo szybkie, jednak napisanie czegoś bardziej skomplikowanego zajęłoby ogromną ilość lini kodu oraz czasu. Następnie istnieją języki wyższego poziomu, którego składnia ułatwia zrozumienie kodu programu przez osoby, które mają z tym kodem styczność.

Aby jeszcze lepiej zrozumieć pisany kod powstała jeszcze jedna warstwa abstrakcji, gdzie tak naprawdę nie jest wymagana od programistów ani jedna linijka kodu. Jest to programowanie wizualne. Główne źródło tworzonego oprogramowania stanowią bloki graficzne i połączenia między nimi.

1.1.1 Programowanie wizualne

Programowanie wizualne jest to programowanie, które pozwala użytkownikowi tworzyć programy poprzez manipulację elementów graficznie, inaczej niż w większości przypadków przy użyciu edytorów tekstowych. Prawie wszystkie akcje, które możliwe do osiągnięcia mogą zostać zrealizowane tylko za pomocą myszki.

Jednym z narzędzi, które pozwala tworzyć aplikacje wizualnie jest App Inventor. Za pomocą powyższego programu istnieje możliwość tworzenia aplikacji na system operacyjny android. Są to głównie telefony i tablety. App Inventor jest aplikacją internetową, dostępną z poziomu przeglądarki. Nie potrzebujemy dodatkowego środowiska do tworzenia programów. App inventor jest aplikacją stworzoną przez Google, a aktualnie utrzymywaną przez uniwersytet Massachusetts Institute of Technology (MIT). Wszystkie nowe osoby, które chciałyby zacząć programować i tworzyć oprogramowanie na system operacyjny Android mogą zacząć od App Inventora. Tworzenie aplikacji jest intuicyjne dzięki graficznemu interfejsowi, który umożliwia użytkownikowi akcje typu "przeciągnij i upuśćńa interesujących go obiektach.[**?**] Są to proste czynności, które nie wymagają głębokiej wiedzy informatycznej. Osoby, które nigdy nie miały do czynienia z programowaniem, nie będą miały większych kłopotów z napisaniem aplikacji.

1.1.2 Programowanie natywne

Programowanie natywne jest to programowanie na daną platformę, a więc napisane oprogramowanie będzie na niej działać bez dodatkowych programów. W przypadku systemu Android jest to język Java. Jest to język obiektowy wysokiego poziomu. Po napisanu programu, kod jest kompilowany do kodu bajtowego, którym zajmuje się maszyna wirtualna javy (JVM). Ładuje pliki do pamięci, a następnie uruchamia zawarty w nich kod. Jednak Android nie posiada JVM. Zamiast JVM, Google wyposażył Android w maszynę Dalvik'a. Dalvik jest to maszyna wirtualna, przystosowana specjalnie do urządzeń mobilnych, gdzie szczególną uwagę należy zwrócić na małe zasoby pamięci, energii i niewielką prędkość procesorów. Kod bajtowy stworzony przez kompilator nie jest w 100% kompatybilny z kodem bajtowym Javy. Nie można tutaj korzystać z bardziej zaawansowanych cech jakimi są Class Loadery czy Java Reflection API. [**?**]

1.2 Cel i zakres pracy magisterskiej

Celem pracy magisterskiej jest porównanie tworzenia aplikacji na platformę android przy pisaniu aplikacji w języku Java, oraz przy wykorzystaniu narzędzia oferowanego online - App Inventor. Praca zawiera porównanie tworzenia oprogramowania z różnych perespektyw, między innymi takich jak:

- Czas potrzebny na stworzenie aplikacji
- Możliwości jakie daje nam App Inventor, jakich rzeczy tam brakuje, a co można użyć
- Łatwość stworzenia aplikacji
- Porównanie takich samych aplikacji pod względem zużycia procesora oraz pamięci
- Porównanie wydajności tych samych algorytmów pod względem czasu
- Jak wygląda stworzenie bardziej zaawansowanej aplikacji korzystającej z wielu funkcji telefonu
- Czy jakieś dodatkowe narzędzia są potrzebne do tworzenia aplikacji

Dzięki takiemu porównaniu powstanie czystszy obraz na narzędzie jakim jest App Inventor. Młodsze osoby zainteresowane programowaniem łatwiej będą mogły się zdecydować, w którą stronę pójść. Czy warto w ogóle zawracać sobie głowę App Inventorem, czy odrazu uczyć się Javy i mieć dostęp do wszystkch funkcji Androida. Dodatkowo nauczyciele informatyki będą mogli rozważyć naukę podstaw programowania poprzez tworzenie aplikacji na system Android.

W pracy zostały również przedstawione wady oraz zalety pisania oprogramowania przy wykorzystaniu App Inventora. Programowanie wizualne, mimo że wydaje się łatwiejsze niesie ze sobą również pewne niedogodności. Pewnych rzeczy prawdopodobnie nie da się zrealizować, a pewne są możliwe do zrealizowania w wiele prostszy sposób.

1.2.1 Struktura pracy magisterskiej

W rozdziale 2 przedstawiono podstawowe pojęcia, które zostały użyte przy pisaniu pracy magisterskiej. Terminy te zostały wyjaśnione, aby bez problemu zrozumieć bardziej skomplikowane zagadnienia.

W rozdziale 2 zawarto teorię dotyczącą App Inventora.

W rozdziale 4 pokazano zastosowane podejście do rozwiązania problemu.

W rozdziale 5 przedstawiono wyniki uzyskanep podczas pisania pracy magisterskiej.

Rozdział 2

Podstawowe pojęcia

W danym rozdziale zostaną zawarte podstawowe pojęcia i mechanizmy używane przez aplikację App Inventor. Ideą tutaj jest przypomnienie oraz przypliżenie ważnych terminów informatycznych.

2.1 App Inventor

W grudniu 2013 roku został wydany App Inventor w wersji drugiej. Starsza wersja została nazwana jako Classic. Oba narzędzia są bardzo podobne jednak projekty stworzone w starszej wersji nie mogą zostać zaimportowane do nowszej. W danej pracy magisterskiej skupienie zostało na nowej wersji App Inventora.

2.2 Główne komponenty

App Inventor celowo ułatwia programowanie poprzez wizualizację tworzonych komponentów i intuicyjny interfejs. App Inventor składa się z 3 głównych komponentów jakimi są:

- App Inventor Designer
- App Inventor Blocks Editor
- Android Device Emulator

2.2.1 App Inventor Designer

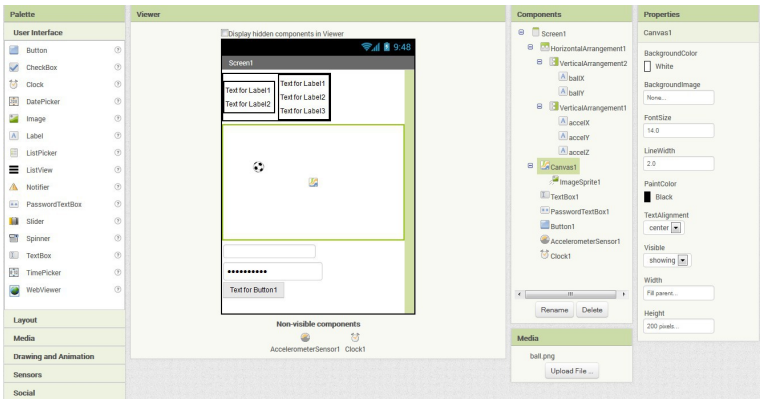
Jednym z głównych widoków jakie można używać jest widok Designera. Projektowanie interfejsu użytkownika polega na przeciąganiu komponentów z dostępnej palety, wliczając w to także niewidoczne komponenty takie jak sensory. W tym widoku można również zmieniać właściwości obiektów, które zostały stworzone. Między innymi istnieje możliwość zmiany położenia, wielkości, układu (pionowy, poziomy).

Designer jest zaprojektowany jako zwykła aplikacja internetowa. Tak więc uruchamia się go, jak zwykłą stronę internetową wpisując jej adres www.

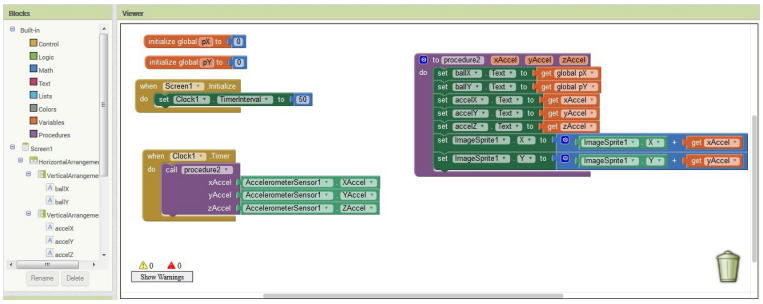
2.2.2 App Inventor Blocks Editor

Drugim widokiem jest Blocks Editor. Zachowanie aplikacji zostaje tutaj zaprogramowane poprzez połączenie odpowiednich bloków. Istnieje możliwość korzystania z bardziej generalnych komponentów, a także z bardziej specyficznych. Dla każdego komponentu, który został stworzony w interfejsie graficznym (Designerze) są dostępne bloki mówiące, co tak naprawdę jest możliwe do zrobienia. Wygląda to w ten sposób, że komponenty są przciągane z dostępnej palety medotą "przeciągnij i upuść", a następnie łączone jak puzzle.

Ta część aplikacji normalnie reprezentowana jest przez kod napisany przez programistę. Więc napisanie zachowania aplikacji odbywa się poprzez łączenie puzzli, bez znajomości języka Java.



Rysunek 2.1: App Inventor Designer



Rysunek 2.2: App Inventor Blocks Editor

2.2.3 Android Device Emulator

Android Device Emulator jest to emulator telefonu lub tabletu. Jest to wirtualna wersja smartphonu, w której znajdują się obsługa dotyku ekranu, przyciski systemowe oraz typowe funkcje.

Zmiany, które zostają wprowadzone, natychmiast reflektują na działanie aplikacji. Nie ma potrzeby jakiejkolwiek kompilacji i uruchamiania aplikacji od nowa. Jeżeli aplikacja zostanie uruchomiana, kompilacja zmienionych fragmentów oraz zainstalowanie ich na emulatorze dzieje się w czasie rzeczywistym. Jest to bardzo wygodna opcja budowania aplikacji i testowania jej. Zmiany, które zrobimy są od razu widoczne na ekranie.

Rysunek 2.3: Android Device Emulator

Rozdział 3

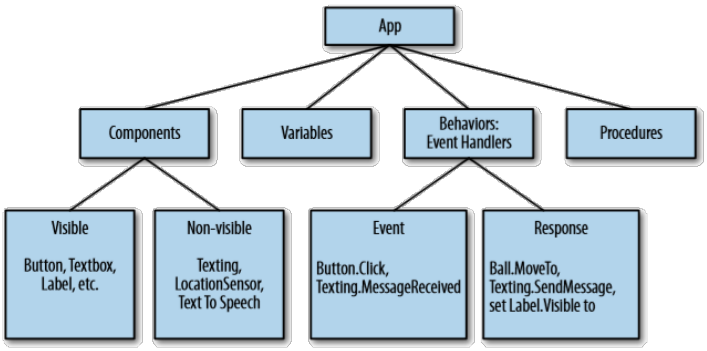
Teoria

3.1 Wstęp

W danym rozdziale zostanie zawarty opis architektury oraz główne komponenty wykorzystywane przez App Inventora. Pomoże to zrozumieć zalety oraz wady powyższego narzędzia.

3.2 Architektura

Każda aplikacja ma swoją wewnętrzną strukturę, którą trzeba dokładnie zrozumieć, aby tworzyć efektywne oprogramowanie. Architektura aplikacji składa się głównie z 2 części: komponenty oraz ich zachowanie. Można z pewnym dystansem przyjąć że za komponenty odpowiada widok Designera, a za zachowanie komponentów widok edytora.



Rysunek 3.1: Architektura aplikacji stworzonej przez App Inventora[**?**]

3.2.1 Komponenty

Komponenty można podzielić na 2 rodzaje: widoczne oraz niewidoczne.

- Widoczne są to takie, które użytkownik widzi gołym okiem np. przyciski, etykiety, pola tekstowe. Definiują one interfejs użytkownika.

- Niewidocznych komponentów, jak sama nazwa wskazuje, użytkownik nie widzi. Nie są one częścią interfejsu. Dostarczają one dostępu do wbudowanych funkcjonalności telefonu. Są to różne sensory np. akcelerometr, moduł gps, komponent zamiany tekstu na mowę itp.

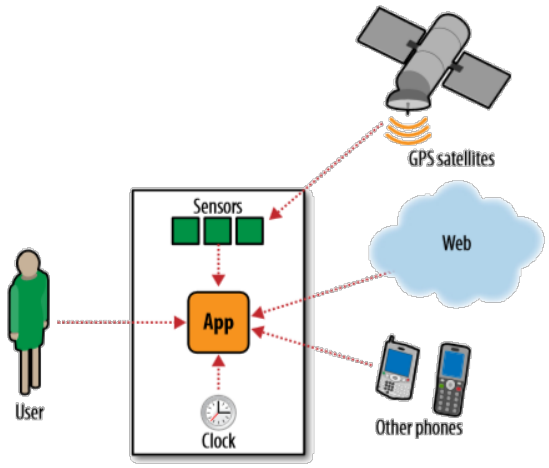
Oba rodzaje komponentów posiadają zbiór swoich właściwości. Właściwości danego komponentu są to informacje jakie komponent posiada. Etykieta posiada między innymi rodzaj, wielkość, dekorację, kolor czcionki, wielkość, widoczność etykiety. Użytkownik nie widzi danych właściwości, obserwuje on rezultat konkretnych ustawień na ekranie urządzenia.

3.2.2 Zachowanie aplikacji

Zrozumienie zasady tworzenia komponentów i ich właściwości jest proste. Nazwy komponentów są intuicyjne, więc nie powinno być problemu z odnalezieniem tego, który interesuje programistę. Z drugiej strony zachowanie komponentów może okazać się bardziej skomplikowane. Mimo wszystko App Inventor stara się wizualizować bloki opisujące zachowanie w jak najprostszej formie.

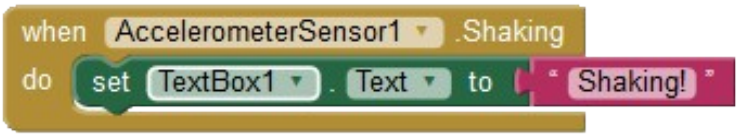
Kiedy zaczynano pisać aplikacje, można było je porównać do recept, gdzie ciąg zdarzeń ukazany jest jako liniowa sekwencja instrukcji. Typowa aplikacja może uruchomić transakcję w banku, dokonać pewnych obliczeń, zmodyfikować stan konta i na koniec wyświetlić nowe saldo.[**?**]

W dzisiejszych czasach większość aplikacji nie wpasowuje się w powyższy schemat. Zamiast wykonywać ciąg instrukcji w odpowiedniej kolejności, reagują na zdarzenia, które są inicjowane przez użytkownika danej aplikacji. Jednym z przykładów jest kliknięcie przycisku lub trzęsienie telefonem, który jest zaprogramowany tak, aby na każdy bodziec móc umieć odpowiedzieć. Wiele zdarzeń jest inicjowanych przez użytkownika, ale są też wyjątki. Aplikacja może reagować na zdarzenia, które w nie wymagają interakcji z użytkownikiem. Często są to niewidoczne komponenty umieszczone w Designerze. Poniższy rysunek prezentuje aplikację otoczoną wieloma zdarzeniami.



Rysunek 3.2: Aplikacja reagująca na zdarzenia zewnętrzne i wewnętrzne[**?**]

Jednym z powodów dlaczego App Inventor jest tak intuicyjny jest zastosowaniej prostej koncepcji nazywania zdarzeń. Zdefiniowanie zdarzenia polega na przeciągnięciu go z palety na główny ekran, a następnie napisanie konkretnego zachowania. Przykładem takiego zdarzenia jest obsługa akcelerometru. Po zatrzęsieniu telefonem, pojawia się tekst Śhaking!".



Rysunek 3.3: Obsługa zdarzenia trzęsienia telefonem

Zdarzenia można podzielić w zależności od jego typu:

- Zainicjowane przez użytkownika - najbardziej popularny typ zdrzania - głównie jest to obsługa zdarzeń dotyku ekranu.
- Inicjalizujące - są wykonywane, gdy dany komponent jest tworzony.
- Czasowe - są uruchamiane, co pewien interwał czasowy.
- Animacje - są zależne od obiektów (spritów) które zostały stworzone. Mogą zostać uruchomione gdy obiekty ze sobą kolidują, wylatują poza ekran.
- Zewnętrzne - są uruchamiane gdy urządzenie odbierze jakiś sygnał zewnętrzny typu, odczyt pozycji urządzenia z satelity, reakcja na przychodzący sms.

Programowanie aplikacji odbywa się poprzez zdefiniowanie interfejsu, a następnie napisanie zachowania danej aplikacji, dla różnych zdarzeń, które mogą wystąpić. Inaczej mówiąc, tworzymy najpierw komponenty w Designerze i ustawiamy im właściwości. Kiedy otrzymaliśmy interesujący nas wygląd zabieramy się za opisanie zdarzeń.

3.3 Debugowanie aplikacji

Rozdział 4

Zastosowane podejście

W danym rozdziale zawarto opis zastosowanego podejścia, do porównania programowania wizualnego i programowania natywnego.

4.1 Wstęp

Zastosowane podejście polegało na stworzeniu jak największej ilości aplikacji, wykorzystujących różne komponenty. Następnie stworzenie takich samych aplikacji w języku Java. Mając dużą liczbę aplikacji pokrywającą prawie wszystkie możliwości App Inventora, będziemy mogli odpowiedzieć na wiele pytań jego dotyczących, które zostały postawione we wstępie pracy.(1.2)

4.2 Dalvik Debug Monitor

Dalvik Debug Monitor (DDMS) jest to narzędzie pomocne w debugowaniu aplikacji. Dostarcza on takich funkcji jak przekierowanie portów, przechwyt obrazu na urządzeniu, informacje o wątkach, stosie, a także o metodach, które są uruchomione jeżeli włączymy ich profilowanie.

W systemie Android każda aplikacja jest uruchamiana w osobnym procesie, a każdy z procesów działa na swojej własnej wirtualnej maszynie. Każda z tych wirtualnych maszyn wystawia unikalny port, do którego może się podłączyć debbuger. Dalvik Debug Monitor zaraz po starcie podłącza się do Android Debug Bridge (ADB) - narzędzia, które pozwala na komunikację z podłączonym urządzeniem.

Po podłączeniu urządzenia tworzony jest serwis monitorujący pomiędzy ADB a DDMS, który powiadamia DDMS, kiedy wirtualna maszyna na urządzeniu jest uruchomiona lub zakończona. Gdy wirtualna maszyna wystartuje DDMS odbiera ID (pid) procesu uruchomionego na tej maszynie korzystając z ADB. Następnie tworzone jest połączenie do debbugera maszyny wirtualnej. Po tych operacjach DDMS jest w stanie komuniokować się z maszyną wirtualną, korzystając z dostosowanego protokołu.[**?**]

Poniżej widać narzędzie Dalvik Debug Monitor. Na telefonie uruchomione są dwa dodatkowe, poza systemowymi, procesy jednocześnie. Po prawej stronie widać wykres obciążenia procesora, poszczególnych procesów.

4.3 Zużycie procesora i pamięci

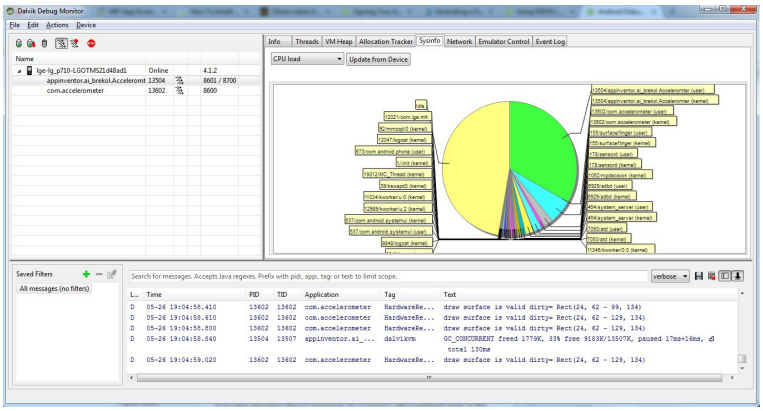
Każda aplikacja powoduje zużycie procesora oraz zajmuje miejsce w pamięci. Do pomiaru tych wielkości został użyty Dalvik Debug Monitor.

Aplikację napisaną w Javie możemy konfigurować dowolnie. Między innymi możemy umożliwić aby była debugowalna, ustawiając parameter:

```
android:debbugable="true"
```

Jest to ważne, ponieważ aplikacja (plik *.apk) wyeksportowana z App Inventora jest niemożliwa do debugowania. Powyższy parametr ma fałszywą wartość logiczną. Aby to zmienić trzeba aplikację zdekompilować, aby zobaczyć źródła aplikacji i zmienić opcję debugowania. Dekompilacja odbywa się za pomocą darmowego narzędzia apktool.

```
apktool -d aplikacja.apk
```



Rysunek 4.1: Przykładowy zrzut ekranu DDMS

Po wykonaniu powyższej komendy zostaje tworzony folder z taką samą nazwą jak nazwa aplikacji. Plik AndroidManifest.xml jest już czytelny i możemy zmienić w nim parametr odpowiadający za debugowanie. Po zmianie, aplikację trzeba skompilować ponownie. Trzeba uruchomić poniższą komendę:

```
apktool -b aplikacja
```

Aplikacja została skompilowana ponownie do pliku *.apk. Aby zainstalować ją na urządzeniu trzeba ją jeszcze cyfrowo podpisać. Generujemy klucz dla aplikacji:

```
keytool -genkey -v -keystore keystore -alias alias_aplikacji -keyalg RSA
-keysize 2048 -validity 20000
```

Następnie podpisujemy aplikację:

```
jarsigner -verbose -keystore keystore aplikacja.apk alias_aplikacji
```

Ostatecznym krokiem jest zainstalowanie aplikacji na telefonie:

```
adb install aplikacja.apk
```

Dzięki tym wszystkim czynnościom maszyna wirtualna uruchamiająca aplikacja uruchomiona na telefonie udostępnia na port umożliwiający debugowanie. Do tego portu podłącza się Dalvik Debug Monitor, z którego możemy odczytać różne statystyki aplikacji i porównać je ze statystykami aplikacji napisanej natywnie w języku Java.

4.4 Możliwości App Inventora

App Inventor posiada bardzo dużą liczbę komponentów, którą możemy użyć. Są jednak pewne braki. Nowe telefony posiadają coraz więcej sensorów oraz innych funkcji, które nie są możliwe do obsłużenia przez powyższe narzędzie.

4.5 Łatwość tworzenia aplikacji

Ilość dokumentacji oraz poblemów, z którymi spotkali się użytkownicy może być różna. Stworzenie bardziej zaawansowanej aplikacji może być wyzwaniem dla App Inventora.

Rozdział 5

Wyniki eksperymentu

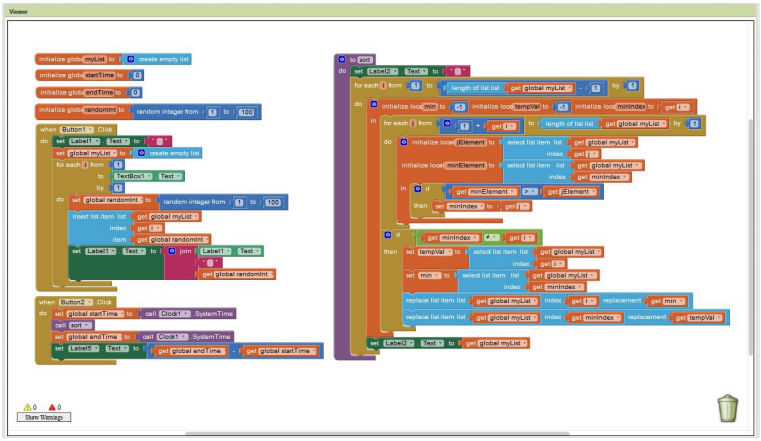
Dany rozdział zawiera wyniki z przeprowadzonych badań oraz wnioski. Każda aplikacja, która została napisana została przestawiona i opisana z różnych perespektyw. Na końcu rozdziału zostały przedstawione wady i zalety obu podejść.

5.1 Stworzone aplikacje

Aplikacje zostały najpierw stworzone w App Inventorze, a następnie zostały przepisane na język Java.

5.1.1 Sortowanie

Aplikacja polega na wygenerowaniu listy losowych elementów, a następnie posortowaniu jej. Do sortowania został użyty prosty algorytm sortowania przez wybieranie ang. *Selection Sort*.



Rysunek 5.1: Aplikacja sortująca - App Inventor

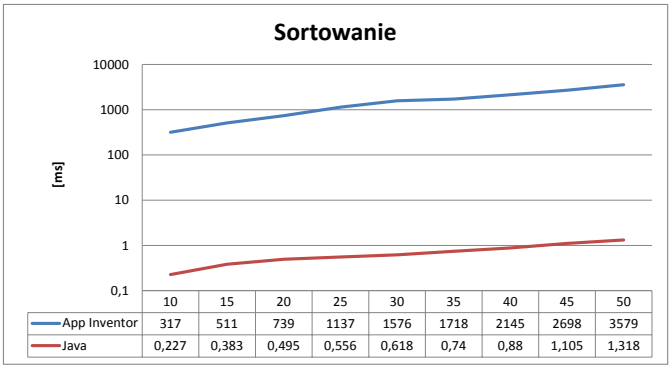
Na powyższym rysunku widać bloki potrzebne do stowrzenia aplikacji w App Inventorze. Bez głębszej analizy zrozumienie działania bloków, może okazać się kłopotliwe. Jest to prosty algorytm, a napisanie go za pomocą dostępnych bloków okazało się skomplikowane. Można sobie łatwo wyobrazić, że napisanie bardziej skomplikowanego algorytmu byłoby bardzo nieczytelne. Ilość użytych bloków zdecydowanieby wzrosła, dodatkowo utrzymanie takiej aplikacji niesie za sobą wysokie koszty wprowadzenia nowych osób do jej rozwijania.

Sortowanie napisanie w javie jest zrozumiałe dla każdego programisty. Do sortowania została użyta lista, jako odpowiednik listy w App Inventorze, nie ma tam dostępnych tablic.

```
void sort(List<Integer> list){
        for(int i =0;i<list.size()-1;i++){
            int index = i;
            for(int j=i+1;j<list.size();j++){
                if(list.get(j) < list.get(index) ){
```

```
                index = j;
            }
        }
        if(index != i){
            int tmp = list.get(i);
            list.set(i,list.get(index));
            list.set(index,tmp);
        }
    }
}
```

W algorytmach bardzo ważna jest wydajność. Oba algorytmy działają w ten sam sposób, jednak wydajność sortowania listy napisanej w Javie jest zdecydowanie wyższa. Można to zaobserwować na poniższym wykresie. Przesortowanie bardzo małej liczby elementów zajmuje App Inventorowi bardzo dużo czasu. Przy 25 elementach czas sortowania przekroczył 1 sekundę. Jest to bardzo słaby wynik w porównaniu do sortowania napisanego w Javie. Średnio czas sortowania był 2 tysiące razy mniejszy! Na danym wykresie została zastosowana skala logarytmiczna, aby zobaczyć różnicę.



Rysunek 5.2: Wykres przedstawiający czas sortowania

Napisanie tej aplikacji w Javie nie było żadnym problemem. Bardzo łatwo było zdebugować kod i sprawdzić jego poprawność. Stworzenie tej samej aplikacji w App Inventorze nie było trywialne.

5.1.2 Akcelerometr

Kolejną aplikacją jest wykorzystująca akcelerometr. Odczytuje ona dane z akcelerometru, a następnie wyświetla je na ekran telefonu, z zadaną częstotliwością. O ile

5.2 Zalety programowania wizualnego

5.3 Wady programowania wizualnego

Rozdział 6

Powiązana praca

6.1 Data extraction

6.2 Inductive wrappers creation

Rozdział 7

Conclusion

Whole conclusion for one page.

Dodatek A

Content of the DVD

As an addition to this document, the DVD is attached. It provides some materials connected with the presented subject in electronic form for potential users or people, who would want to continue works on this topic.

The DVD content consists of several items:

1. Item 1
2. Item 2
3. Item 3

Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@mastersthesis{ key,
    author = "Jakub Bręk",
    title = "{Programowanie wizualne urządzeń mobilnych}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2014",
}
```