

Spis treści

1.	Wstęp	3
1.1.	<i>Problematyka i zakres pracy</i>	<i>3</i>
1.2.	<i>Cele i zakres pracy.....</i>	<i>4</i>
1.3.	<i>Przegląd literatury.....</i>	<i>5</i>
1.4.	<i>Układ pracy</i>	<i>6</i>
2.	Podstawowe informacje o Androidzie	8
2.1.	<i>Historia Androida.....</i>	<i>8</i>
2.2.	<i>Android Studio.....</i>	<i>8</i>
2.3.	<i>Gradle.....</i>	<i>9</i>
2.4.	<i>Android Manifest</i>	<i>10</i>
2.5.	<i>Aktywności.....</i>	<i>12</i>
2.6.	<i>Uprawnienia.....</i>	<i>14</i>
2.7.	<i>Zasoby aplikacji.....</i>	<i>15</i>
2.8.	<i>Serwisy.....</i>	<i>18</i>
2.9.	<i>Android Support Library</i>	<i>20</i>
3.	Intencje oraz broadcast receivers	21
3.1.	<i>Intencje</i>	<i>21</i>
3.1.1.	<i>Wstęp.....</i>	<i>21</i>
3.1.2.	<i>Uruchamianie aktywności i serwisów przy pomocy intencji</i>	<i>21</i>
3.2.	<i>Natywne akcje Androida.....</i>	<i>23</i>
3.3.	<i>Dodawanie dodatkowych informacji do intencji.....</i>	<i>26</i>
3.3.1.	<i>Uri.....</i>	<i>26</i>
3.3.2.	<i>Mapa dodatkowych danych</i>	<i>26</i>
3.3.3.	<i>Flagi</i>	<i>27</i>
3.4.	<i>Podstawowe informacje o IntentFilters.....</i>	<i>27</i>
3.4.1.	<i>Jak system rozwiązuje filtry intencji</i>	<i>27</i>
3.5.	<i>Odbiorniki.....</i>	<i>28</i>
3.5.1.	<i>Implementacja klasy BroadcastReceiver.....</i>	<i>29</i>

3.5.2.	Rejestracja odbiornika	29
3.5.3.	Wysyłanie wiadomości oraz cykl życia odbiornika	30
3.6.	<i>Lokalne broadcast receivers</i>	31
3.7.	<i>Broadcast receivers udostępniane przez Androida</i>	32
3.7.1.	Wydarzenia dostępne w klasie <i>Intent</i> do powszechnego użytkowania.....	32
3.7.2.	Przykłady wydarzeń z innych klas dostępnych w API Androida....	34
4.	Opis praktycznej części pracy	36
4.1.	<i>Opis aplikacji</i>	36
4.1.1.	Funkcjonalność aplikacji	36
4.1.2.	Architektura programu	37
4.2.	<i>Opis procesu budowania i instalacji aplikacji na urządzeniu</i>	39
4.2.1.	Wymagania	39
4.2.2.	Proces budowy projektu	39
4.2.3.	Instalacja zbudowanej aplikacji przy użyciu narzędzia adb	40
4.3.	<i>Przykładowe użycie programu</i>	41
4.3.1.	Tworzenie zadania	41
4.3.2.	Usuwanie zadania	47
5.	Podsumowanie pracy.....	49
	Bibliografia	50
	Spis rysunków	53
	Spis tabel.....	54
	Spis listingów zawierających kod źródłowy	55

1. Wstęp

1.1. Problematyka i zakres pracy

W pierwszym kwartale 2014 roku, trzy na pięć sprzedanych telefonów komórkowych nosiło nazwę „inteligentny” [1]. Smartphone¹, jak potocznie na urządzenia tego typu się mówi, nie tylko zawładnęły niemałym rynkiem telefonów bezprzewodowych, ale także znacznie poprawiły komfort naszego życia. Od obsługi wiadomości SMS, po zarządzanie pocztą elektroniczną, do sterowania temperaturą w naszych mieszkaniach – lista rzeczy, które nie potrafią te urządzenia z każdym dniem zmniejsza się jeszcze bardziej.

Nic dziwnego, że tym rynkiem, który jest wart miliardy dolarów rocznie, zainteresowały się wielkie firmy. Swoje rozwiązania w tej dziedzinie na przestrzeni ostatnich lat pokazały takie firmy jak: Microsoft, Apple czy Google. W swojej pracy zapoznam się z jedną z części systemu ostatniej z wymienionych (mieszczącej się w Mountain View) korporacji, odpowiedzialną za komunikacji między komponentami zainstalowanymi na urządzeniu. Dzięki systemowi nadajników i odbiorników program, który będzie odbierał wiadomości SMS nie musi posiadać dodatkowego wątku odpowiedzialnego za sprawdzenie czy wiadomość tekstowa przyszła. Jedynie potrzebuje wyczekać na informację od części systemu odpowiedzialnej za odbieranie krótkich wiadomości. Taki sposób rozwiązania komunikacji między aplikacjami w Androidzie przysporzył się do znacznego ograniczenia używanych zasobów przez oprogramowanie działające na tym systemie.

Częścią badawczą mojej pracy było wykonanie aplikacji przedstawiającej działanie tego mechanizmu w Androidzie. Program ten wykorzystuje interfejsy programistyczne, które zostały udostępnione przez firmę Google. Sama aplikacja jest napisana w wersji standardowej języka Java. Jednak sposób, w jaki został

¹ **Smartphone** (z ang. *Intelligentny telefon*) – urządzenie elektroniczne, które można zmieścić w kieszeni posiadające możliwość wykonywania połączeń głosowych oraz pozwalające na instalowanie i uruchamianie oprogramowania firm trzecich.

zaprojektowany interfejs programistyczny Androida różni się od tego zaproponowanego przez Oracle. Programiści znający już podstawową edycję Javy zaczynający swoją przygodę z systemem od firmy z Mountain View muszą zapoznać się nie tylko z nazwami udostępnionych klas i metod, ale z sposobem tworzenia interfejsu użytkownika czy komunikacji między systemem a aplikacją. Z tego powodu uważam, że moja praca nie tylko zapoznała czytelnika z systemem roszyłania i odzierania informacji w Androidzie, ale także zapoznała go z podstawowymi aspektami tworzenia oprogramowania na ten system.

W części teoretycznej zostaną opisane takie zagadnienia jak:

- Jak korzystać z powyższych interfejsów
- Komponenty systemu Android, z których można pobierać lub wysyłać informacje potrzebne w działaniu programu
- Klasy poboczne (jak na przykład *IntentFilter*), które są potrzebne w korzystaniu z tych interfejsów
- Zasięg działania obiektów typu *Intent* oraz *BroadcastReceiver*

1.2. Cele i zakres pracy

Głównym celem pracy jest przedstawienie w mechanizmu komunikacji między komponentami w systemie Android.

Moja praca skupi się na opisanu jednych z najważniejszych mechanizmów w systemie od firmy Google, mianowicie na klasach *Intent* oraz *BroadcastReceiver*. Te obiekty ułatwiają komunikację między komponentami systemu. Trudno wyobrazić sobie scenariusz, gdzie mając na uwadze jak najmniejsze zużycie procesora i pamięci operacyjnej, na jednoczesnym istnieniu dwóch programów na tym samym urządzeniu, które korzystają z osobnych wątków do wykonania tej samej długiej operacji.

Pomniejszym celem mojej pracy dyplomowej jest przedstawienie także takich części programu pisanego pod Androida jak:

- Android Manifest
- Aktywności

- Zarządzenie i rodzaje zasobów, z których korzystają aplikacje napisane pod system Android
- Serwisy
- Uprawnienia

Powyższe komponenty są częścią sporej ilości programów mobilnych. Bez znajomości tych technologii trudno zacząć pisać aplikacje dla systemu Android jak i opisać tytułowy mechanizm.

1.3. Przegląd literatury

Podstawowym źródłem informacji na temat tworzenia aplikacji na Androida jest jego dokumentacja [2]. Wiedza przedstawiona na tej stronie internetowej jest przejrzysta oraz napisana łatwym do zrozumienia językiem. Nie tylko zawiera opis API² Androida, ale także przykłady zastosowań komponentów w niej zawartych [3]. O ile dokumentacja dostarczona przez Google jest bardzo dobrej jakości w większości obszarów, to tak niektóre zagadnienia nie zostały w ogóle wspomniane. Przykładem komponentu, który nie został opisany w tutorialach³, jest klasa *BroadcastReceiver*. Na szczęście to zagadnienie zostało opisane przez osoby niepracujące w firmie z Mountain View [4].

O ile wiedza dostępna w Internecie na temat tworzenia aplikacji działających na systemie Android jest duża i najczęściej darmowa, jest ona rozszkana po całej sieci. Czas spędzany na przeszukiwaniu Internetu w poszukiwaniu dobrej jakości artykułu na temat nas interesujący jest czasami ogromny, dlatego istnienie książek z tej domeny jest bardzo potrzebny. Moim zdaniem jednym z najlepszych podręczników mówiących o programowaniu na Androida jest wolumen autorstwa Reto Meier [5]. Książka ta, jest jednym z najlepszych kompendiów wiedzy dostępny na rynku. Zawarte w niej jest wszystko, co musi wiedzieć każdy, kto chce programować na system od Google.

² **API** (skrót od *Application Programming Interface* – z ang. *interfejs programowania aplikacji*) – podprogramy, struktury danych, klasy obiektów, które udostępnia dany program, biblioteka, system operacyjny w celu komunikacji w innym oprogramowaniu.

³ **Tutorial** (z ang. *Samouczek*) – zestaw instrukcji pozwalający na łatwe nauczanie się określonego zagadnienia.

Trudno o dobry podręcznik traktujący o tworzeniu aplikacji na Androida napisany po polsku. Najbliżej stwierdzeniu „kompedium wiedzy” jest przetłumaczona książka autorstwa Shane Conder oraz Lauren Darcey [5]. Podręcznik ten zawiera informacje potrzebne do bezproblemowego napisania pierwszej poważnej aplikacji na Androida. Warto dodać, że informacje na temat klas *Intent* oraz *BroadcastReceiver* są rozłożone po książce, nie jak w wolumenie pana Reto Meiera.

1.4. Układ pracy

Tematem pracy jest mechanizm rozsyłania informacji o zdarzeniach w systemie Android, a głównym celem mojej pracy jest przedstawienie tego problemu w języku polskim. Pierwszy rozdział zawiera opis problematykę i cele pracy dyplomowej oraz przegląd literatury z tej dziedziny.

Drugi rozdział to przedstawienie podstawowych informacji potrzebnych do opisu tytułowego mechanizmu. Ta część pracy zawiera między innymi wiedzę na temat:

- Manifestu aplikacji
- Zasobów, które używa aplikacja pod Androida
- Uprawnień, które może uzyskać program

W trzecim rozdziale został opisany tytułowy mechanizm. Informacje w nim zawarte dotyczą:

- Tworzenia, używania obiektów klas *Intent* oraz *BroadcastReceiver*
- Opisu identyfikacji broadcast receivera przez system
- Zasięgu nadajnika oraz odbiornika
- Natywnych akcji Androida

Czwarty rozdział to opis programu napisanego na potrzeby pracy. Została tam opisana architektura aplikacji oraz przykładowe jej zastosowanie.

Ostatnim rozdziałem jest podsumowanie, w którym zaprezentowano rezultaty pracy. Wynika z nich przede wszystkim, że bez mechanizmu intencji oraz odbiorników aplikacje pisane pod Androida wymagałyby urządzeń z dużą ilością pamięci operacyjnej oraz szybkim procesorem. W rezultacie praca

ukazuje wady i zalety mechanizmu rozsyłania informacji o zdarzeniach w systemie od firmy Google oraz jego opis w języku polskim.

2. Podstawowe informacje o Androidzie

2.1. Historia Androida

W roku 2008 na Mobile World Congress w Barcelonie firma Google pokazała pierwsze prototypowe urządzenie z ich nowym systemem mobilnym na pokładzie nazwanym Android [7]. Był to telefon przypominający popularne w tamtym okresie produkty spółki Research In Motion (teraz BlackBerry). Kilka miesięcy później firma HTC wprowadziła do sprzedaży pierwszy telefon z tym systemem [8] – HTC Dream, znany bardziej jako T-Mobile G1.

Od tego czasu Androida działa na ponad 1 miliardzie urządzeń [9], miał kilkanaście dużych wydań oraz jest obecny na prawie, co drugim sprzedanym smartphonie [10].

Sukces Androida może przypisać modelowi biznesowemu, jaki przyjęła firma z Mountain View:

- System ten jest dostępny, jako projekt open-source, czyli każdy kto tylko ma wiedzę i możliwości może pobrać jego kod i stworzyć własną wersję Androida
- Może działać na prawie każdym urządzeniu wyposażonym w mikroprocesor, na przykład telefony komórkowe, telewizory, a nawet lodówki
- Tworzenie aplikacji na niego jest całkowicie darmowe. Każdy może wejść na stronę dla developerów Androida i pobrać narzędzia potrzebne do tego

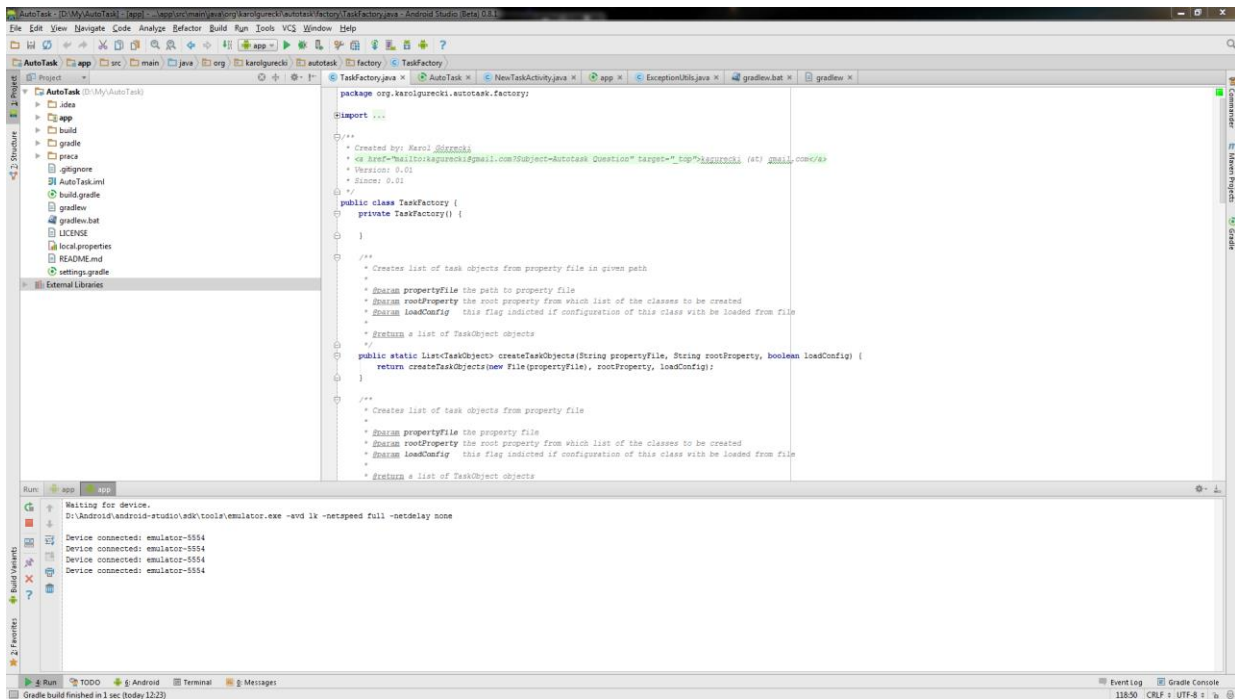
2.2. Android Studio

Android Studio to oficjalne IDE⁴ do tworzenia programów działających pod kontrolą Androida. Składa się z dwóch części:

- SDK⁵ dla systemu Android

⁴ **IDE** (skrót od *integrated development environment*, z ang. *zintegrowane środowisko programistyczne*) – jedna lub zestaw aplikacji pozwalających na edycję, budowanie oraz testowanie kodu programu przez programistę.

- IntelliJ IDEA Community Edition, zmodyfikowany przez firmę Google na potrzeby programowanie na ich system. Przykładowy zrzut ekranu przedstawiający okno tego programu widać na rysunku 1.



Rysunek 1. Przykładowy zrzut ekranu z IntelliJ Android Studio.

2.3. Gradle

Gradle jest to narzędzie do automatycznego budowania aplikacji. Dzięki mechanizmowi dodatków może także bez ingerencji użytkownika między innymi przetestować kod, wygenerować statystyki czy dokumentację. Jest połączeniem zalet dwóch największych konkurentów w swojej dziedzinie – Anta oraz Mavena. W drugim kwartale roku 2013 Google przedstawiło Gradle jako zalecane narzędzie do budowy projektów programów napisanych na Androida.

```
1  apply plugin: 'com.android.application'
2
3  android {
4      compileSdkVersion 19
5      buildToolsVersion "19.1.0"
6
7      defaultConfig {
8          applicationId "org.karolguerecki.autotask"
9          minSdkVersion 16
10         targetSdkVersion 16

```

⁵ **SDK** (skrót od *Software Development Kit*) – zestaw narzędzi ułatwiających tworzenie programów na dane środowisko.

```

11     }
12
13     buildTypes {
14         release {
15             runProguard false
16             proguardFiles getDefaultProguardFile(
17 'proguard-android.txt'), 'proguard-rules.txt'
18         }
19     }
20     compileOptions {
21         sourceCompatibility JavaVersion.VERSION_1_7
22         targetCompatibility JavaVersion.VERSION_1_7
23     }
24
25     packagingOptions {
26         exclude 'META-INF/NOTICE.txt'
27         exclude 'META-INF/LICENSE.txt'
28     }
29 }
30
31 dependencies {
32     compile 'com.google.guava:guava:16.0.1'
33     compile 'org.apache.commons:commons-collections4:4.0'
34     compile 'org.apache.commons:commons-lang3:3.3.2'
35     compile 'com.android.support:support-v13:18.0.+'
36 }

```

Listing 1. Przykłady plik build.gradle.

W pierwszej linii z powyższego listingu można zauważyć deklaracje używania dodatku do Gradle dodającego wsparcie dla projektów aplikacji na Androida. Następnie znajduje się konfiguracja tego pluginu. Ustawiane są takie parametry jak minimalna wersja Androida, identyfikator aplikacji w systemie czy pliki, które nie powinny być dołączone z programem. Na koniec (od linii 31) można zauważyć dodatkowe zależności wykorzystywane w projekcie.

Proces budowania projektów został opisany w rozdziale 4.2.

2.4. Android Manifest

Android Manifest to plik XML, który zawiera opis atrybutów aplikacji. W nim są zawarte informacje takie jak: nazwa aplikacji, uprawnienia jakie posiada aplikacja czy broadcast receivers. Tabela 1 przedstawia przykładowe znaczniki, które są dostępne oraz ich krótki opis.

Tabela 1. Przykładowe znaczniki, które są dostępne w Android Manifest.

Nazwa znacznika	Opis
<action>	Zawiera nazwę akcji, przy której będzie uruchamiany <i>BroadcastReceiver</i> . Używany w <intent-filter>
<activity>	Deklaruje aktywności dostępne w programie. Jeżeli aktywność nie zostanie opisana przez ten znacznik nie będzie widoczna dla systemu i nie zostanie wyświetlona.
<application>	Definiuje atrybuty aplikacji takie jak: jej nazwę, ikonę, czy może być stosowana do deklaracji aktywności oraz serwisów używane przez aplikację.
<intent-filter>	Może być zdefiniowany wewnątrz znaczników <activity>, <service> oraz <receiver>. Odpowiada za zdefiniowanie akcji, na które wcześniej wymienione komponenty będą mogły odpowiadać.
<manifest>	Główny element w pliku XML'owym.
<receiver>	Jeden z dwóch sposobów na zdefiniowanie obiektu typu <i>BroadcastReceiver</i> .
<service>	Używany go deklaracji obiektów klasy <i>Service</i> . Tak jak w przypadku <activity>, jeżeli serwis nieopisany tutaj, system nie będzie w wiedział o jego istnieniu.
<uses-permission>	Definiuje uprawnienia, które muszą być udzielone aplikacji, by mogła działać poprawnie.

Poniższy listing przedstawia przykładowy Android Manifest.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="org.karolgurecki.autotask"
4      android:versionCode="1"
5      android:versionName="1.0">
6      <uses-sdk android:minSdkVersion="16"/>
7      <application android:label="@string/app_name"
8          android:theme="@android:style/Theme.DeviceDefault">
9          <activity android:name=".ui.activities.Main"
10             android:label="@string/app_name">
11             <intent-filter>
12                 <action android:name="android.intent.action.MAIN"/>

```

```

13         <category android:name=
"android.intent.category.LAUNCHER"/>
14     </intent-filter>
15 </activity>
16 <activity android:name=".ui.activities.NewTaskActivity"
17         android:label="@string/newTaskButtonText"/>
18 <activity android:name=".ui.activities.TaskListActivity"
19         android:label="@string/taskListButtonText"/>
20 <activity android:name=".ui.activities.TaskList"
21         android:label="Task List"/>
22 <service android:name=".service.StartupService"
23         android:label="Autotask Service"/>
24 <receiver android:name="org.karolgurecki.autotask.
25         service.StartupBroadcastReceiver">
26     <intent-filter>
27         <action android:name="android.intent.action.
ACTION_EXTERNAL_APPLICATIONS_AVAILABLE"/>
28         <action android:name="android.intent.action.
BOOT_COMPLETED"/>
29     </intent-filter>
30 </receiver>
31 </application>
32 <uses-permission android:name=
"android.permission.READ_EXTERNAL_STORAGE"/>
33 <uses-permission android:name=
"android.permission.WRITE_EXTERNAL_STORAGE"/>
34 <uses-permission android:name=
"android.permission.RECEIVE_BOOT_COMPLETED"/>
35 <uses-permission android:name="android.permission.BLUETOOTH"/>
36 </manifest>

```

Listing 2. Przykładowy plik AndroidManifest.xml.

2.5. Aktywności

Za interfejs użytkownika aplikacji w Androidzie odpowiadają obiekty klasy *Activity*. Większość programów posiada wiele luźno ze sobą powiązanych aktywności. Zwykle aplikacje mają jeden główny obiekt *Activity*, który jest uruchamiany zaraz po jej włączeniu. Aby system wiedział, że dana aktywność jest „główna” musi posiadać *IntentFilter*, który zawiera *android.intent.action.MAIN* jako akcję oraz kategorie *android.intent.category.LAUNCHER* – przykład takiego zastosowania widać na listingu drugim w liniach 12 i 13.

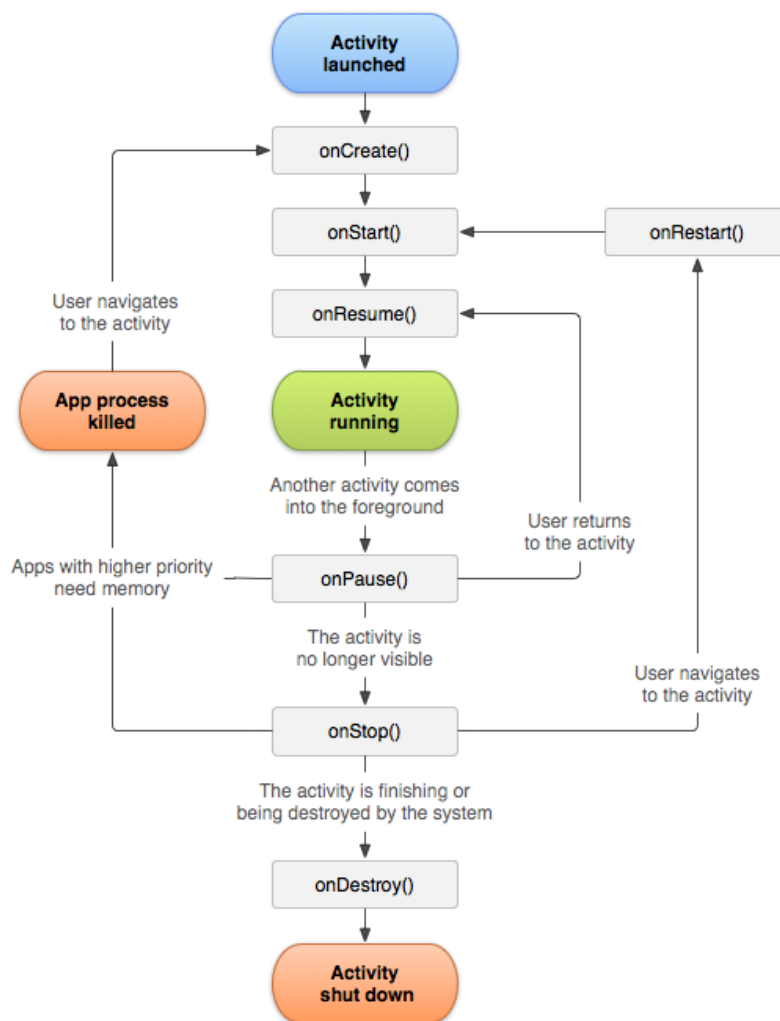
Powyższy sposób działa jedynie gdy chcemy, aby jakaś aktywność była uruchomiona jako pierwsza. Aby wyświetlić inne *Activity*, na przykład po naciśnięciu przycisku, trzeba utworzyć *Intent* używając obiektu *Class*

aktywności, którą chcemy pokazać. Listing 3 przedstawia przykład uruchomienia aktywności z kodu programu.

```
1 Intent intent = new Intent(Main.this, TaskListActivity.class);  
2 startActivity(intent);
```

Listing 3. Tworzenie Activity przy pomocy kodu.

Każde *Activity* w Androidzie ma kilka możliwych stanów, w których może się znaleźć. Sekwencje przechodzenia między nimi widać na rysunku 2.



Rysunek 2. Cykl życia aktywności. Źródło: [11].

Podczas tworzenia nowej klasy, które rozszerza *Activity* musimy napisać własną implementację metody *onCreate()*. Reszta stanów aktywności posiad swoje domyślne definicje. Jednakże nic nie stoi na przeszkodzie, aby programista napisał własne.

Trzeba także wspomnieć, że system posiada tak znany Back Stack, czyli stos aktywności, które zostały uruchomione przez użytkownika. Ta kolekcja jest kolejkowana przy pomocy algorytmu LIFO⁶, czyli ostatnia aktywność na stosie jest aktualnie wyświetlana przez system.

2.6. Uprawnienia

Z powodu chęci zapewnienia bezpieczeństwa danych umieszczonych na urządzeniu, w Androidzie zastosowano system uprawnień (ang. *Permissions*). Domyślnie aplikacja nie posiada żadnych przywilejów. Programista wiedząc, że będzie potrzebować dostępu na przykład do stanu modułu Bluetooth, musi powiadomić o tym fakcie system. Odbywa się to przy pomocy dodania znacznika `<use-permission>` w `AndroidManifest.xml`. Android posiada szereg wbudowanych uprawnień, których przykłady przedstawiono w tabeli 2.

Tabela 2. Przykładowe uprawnienie, które udostępnia Android.

Nazwa przywileju	Opis
android.permission.ACCESS_WIFI_STATE	Umożliwia aplikacji dostęp do informacji na temat sieci Wi-Fi
android.permission.BATTERY_STATS	Pozwala na odczyt stanu baterii.
android.permission.BLUETOOTH_ADMIN	Umożliwia odnajdywanie urządzeń przez protokół Bluetooth oraz łączenie z nimi.
Android.Permission.INTERNET	Potrzebne, aby program mógł korzystać z zasobów dostępnych w Internecie.
android.permission.VIBRATE	Umożliwia używanie funkcji wibracji w urządzeniu.
android.permission.RECEIVE_SMS	Pozwala na odczyt wiadomości SMS, które przychodzą do urządzenia.

⁶ **LIFO** (skrót od *last in, first out*) – algorytm kolejowania elementów na liście jednowymiarowej. Oznacza, że element, który jako ostatni został dodany do tej kolekcji będzie zdjęty jako pierwszy przy próbie pobrania z niej danych.

API Androida pozwala także na definiowanie własnych uprawnień przez programistę. Pozwala ono na zablokowanie dostępu do takich elementów aplikacji jak: serwisy, aktywności czy odbiorników przez inne aplikacje. Aby tego dokonać programista musi dodać znacznik `<permission>` w AndroidManifest.xml z definicją upoważnienia w deklaracji elementu, który chce zabezpieczyć.

2.7. Zasoby aplikacji

Oddzielenie zasobów używanych przez aplikację od jej kodu, jest uważane przez firmę Google jako jeden z best practice⁷ przy programowaniu na ich system. Z tego powodu Android został wyposażony w szereg udogodnień dla programistów. Te proces pozwala w prosty sposób przystosować program do między innymi do różnych języków czy wielkości ekranu.

Android wymaga, aby wszystkie zasoby były umieszczane w katalogu res/, który znajduje się w tym samym folderze co manifest aplikacji. Tutaj trzeba także nadmienić, że różne typy zasobów muszą znajdować się w różnych (określonych przez specyfikację) podfolderach.

Dokumentacja dostarczona przez firmę Google dzieli zasoby na dwa podstawowe rodzaje:

- Alternatywne, czyli zasoby, które są tworzone z myślą o określonej konfiguracji (na przykład plik z layoutem dla danej wielkości ekranu). System rozpoznaje, że dana grupa zasobów odnosi się do określonej konfiguracji poprzez odpowiedni modyfikator w nazwie podfolderu, w którym się ona znajduje (na przykład, jeżeli katalog nazywa się `values-pl` to Android wie, iż w nim są pliki odnoszące się do lokalizacji aplikacji w języku polskim)
- Domyślne – to zasoby, które system zacznie używać jeżeli nie znajdzie żadnych konfiguracji pod aktualną specyfikację

Każdy zasób może być użyty w kodzie aplikacji. Odbywa się to przy pomocy podklas klasy *R*. Ta klasa jest automatycznie generowana przez kompilator podczas kompilacji projektu. Kompilator czyta zawartość folderu `res/` wraz jego

⁷ **Best practice** (z ang. *Najlepszy sposób*) – zalecany sposób rozwiązywania często spotykanych problemów

podkatalogami i umieszcza wygenerowane numery seryjne znajdujących się w nich plików, w odpowiednich podklasach klasy *R*.

Dodatkowo powyższe rodzaje zasobów dzielą się na kilka typów:

- Animacje – zawiera informacje na temat animacji zdefiniowanych przez autora aplikacji. Poszczególne klatki powinny być umieszczone w katalogu *res/drawable/* oraz używane z Javy przy pomocy klasy *R.drawable*. Natomiast przejścia między poszczególnymi klatkami powinny się znaleźć w *res/anim/*, a korzystać można z nich przy pomocy *R.anim*.
- Listy stanów kolorów – definiuje informacje na temat kolorów używanych w widokach. Dane są zapisane w */res/color/* oraz dostępne są przy pomocy *R.color*.
- Grafiki – definiuje informacje na temat danych graficznych w postaci bitmap lub plików XML. Powinny być umieszczone w katalogu */res/drawable/*, a używane z poziomu kodu przy pomocy *R.drawable*.
- Layout – definiuje układ elementów na poszczególnych widokach. Pliki XML (przykład przedstawiono na listingu 4) z tymi informacjami umieszczone muszą być w */res/layout/*, a dostęp do nich zapewnia *R.layout*.

```
1 <LinearLayout xmlns:android=  
2     "http://schemas.android.com/apk/res/android"  
3     android:orientation="vertical" android:layout_width="fill_parent"  
4         android:layout_height="fill_parent">  
5     <Button android:layout_width="match_parent"  
6         android:layout_height="fill_parent"  
7         android:text="@string/newTaskButtonText"  
8         android:id="@+id/newTaskbutton"  
9         android:layout_gravity="left|center_vertical"  
10        android:layout_weight="0.5"/>  
11     <Button android:layout_width="match_parent"  
12         android:layout_height="fill_parent"  
13         android:text="@string/taskListButtonText"  
14         android:id="@+id/taskListButton"  
15         android:layout_gravity="left|center_vertical"  
16         android:layout_weight="0.5"/>  
17 </LinearLayout>
```

Listing 4. Przykładowy plik XML zawierający informacje na temat układu widoku.

- Menu – zawiera informacje o zawartości menu w aplikacji. Z kodu dostępne poprzez *R.menu*, a pliki znajdują się w */res/menu/*.
- Style (z ang. styl) – definiuje wygląd interfejsu użytkownika. Pliki należy umieszczać w */res/style*, a dostęp zapewnia *R.style*.
- Inne zasoby – zasoby takie jak typu *String*⁸, liczby całkowite, tablice muszą być umieszczone w */res/values/*. Każdy typ ma swoją podklasę (na przykład *R.string*, *R.integer*). Listing 5 przedstawia przykładowy plik *strings.xml*.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string name="app_name">AutoTask</string>
4      <string name="newTaskButtonText">New Task</string>
5      <string name="taskListButtonText">Task List</string>
6      <string name="newTaskNameEditText">Task name</string>
7      <string name="actions">Actions</string>
8      <string name="triggers">Triggers</string>
9      <string name="addNew">Add new...</string>
10     <string name="add_trigger">Add trigger</string>
11     <string name="add_action">Add action</string>
12     <string name="finish">Finish</string>
13     <string name="saveTaskTitleAlert">Save task</string>
14     <string name="doYouWantSave">Do you want to save this
15     task?</string>
16     <string name="yes">Yes</string>
17     <string name="no">No</string>
18     <string name="cancel">Cancel</string>
19     <string name="autotask">Autotask</string>
20     <string name="off">Off</string>
21     <string name="on">On</string>
22     <string name="confirm">Confirm</string>
23     <string name="bluetoothTriggerConfigTitle">Bluetooth trigger
24     configuration</string>
25     <string name="bluetoothTrigger">Bluetooth trigger</string>
26     <string name="ifString">If</string>
27     <string name="toastDialogTitle">Toast Text
28     Configuration</string>
29     <string name="display">Display</string>
30     <string name="toastActionName">Toast Action</string>
31 </resources>

```

Listing 5. Przykładowy plik *strings.xml*.

Google w API Androida umieściło także klasę *android.R* zawierającą zasoby, które często są wykorzystywane przez programistów. Klasa ta ma w sobie

⁸ **String** (z ang. *łańcuch znaków*) – typ zmiennej w języku Java, który może przechowywać całe ciągi znaków.

referencje do takich obiektów jak: domyślne style, ikony, czy często używane stringi.

2.8. Serwisy

Aktywności w Androidzie są aktywne tylko jak są wyświetlane na ekranie urządzenia. Każda akcja, która trwa długo powoduje, że widok, który obserwuje użytkownik nie odpowiada. Dlatego osoby, które projektowały ten system postanowiły wprowadzić mechanizmy do przetwarzania informacji w tle. Po za domyślnymi mechanizmami, które udostępnia Java oddano kilka, które zostały stworzone na potrzeby systemu od Google'a. Jednym z ich są serwisy.

W odróżnieniu od aktywności – serwisy zostały stworzone z myślą o długo trwałej pracy. Posiadają wyższy priorytet niż zamrożone widoki, przez co zostaną usunięte z pamięci tylko, jeżeli systemowi (po uprzednim opróżnieniu stosu uśpionych aktywności) zabraknie pamięci operacyjnej na przykład na potrzeby utworzenia kolejnego widoku. Niemniej Android pozwala na skonfigurowanie systemu, aby ten mógł ponownie uruchomić nasz serwis, jeżeli ilość wolnego RAMu na to pozwala.

Aby utworzyć własny serwis programista musi rozszerzyć klasę *Service*. W celu odpowiedzenia na komendę uruchomienia komponentu trzeba nadpisać jedną w dwóch metod *onStartCommand* lub *onStart*. Z powodu, że ta druga funkcja została oznaczona jako przestarzała w chwili wydania Androida w wersji 2.0, Google zaleca używanie metody *onStartCommand*.

```
1 public class StartUpService extends Service {
2     public static final List<String> TASK_PROPERTIES_NAME_LIST
3         = new LinkedList<>();
4     private final static FileFilter FILE_FILTER =
5         new FileFilter() {
6         @Override
7         public boolean accept(File pathname) {
8             String name = pathname.getName().toLowerCase();
9             return name.endsWith(
10                 ConstanceFieldHolder.
11                     PROPERTIES_FILE_EXTENTION.toLowerCase())
12                 && !TASK_PROPERTIES_NAME_LIST.contains(name);
13         }
14     };
15     public static TaskHolderMap TASK_HOLDER_MAP;
16 }
```

```

17  @Override
18  public void onCreate() {
19      super.onCreate();
20  }
21
22  @Override
23  public int onStartCommand(Intent intent, int flags, int startId){
24      super.onStartCommand(intent, flags, startId);
25      if (TASK HOLDER_MAP == null) {
26          TASK HOLDER_MAP = new TaskHolderMap();
27          registerReceiver(TASK HOLDER_MAP, new
28              IntentFilter(TaskHolderMap.TASK HOLDER_MAP_ACTION));
29      }
30      File autoTaskFolder = getFilesDir();
31      if (autoTaskFolder.exists()) {
32          File[] fileArray =
33              autoTaskFolder.listFiles(FILE_FILTER);
34          for (File file : fileArray) {
35              // ...
36          }
37      } else if (!autoTaskFolder.mkdirs()) {
38          Log.e(ConstanceFieldHolder.AUTOTASK_TAG,
39              "Can't create a AutoTask folder");
40      }
41      return Service.START_STICKY;
42  }
43
44  public IBinder onBind(Intent intent) {
45      return null;
46  }
47
48  @Override
49  public boolean stopService(Intent name) {
50      TASK HOLDER_MAP.destroyAll();
51      unregisterReceiver(TASK HOLDER_MAP);
52      return super.stopService(name);
53  }
54}

```

Listing 6. Fragment StartUpService.java.

Na powyższym listingu widzimy fragment serwisu, będącego częścią praktycznej części pracy dyplomowej. Ta implementacja klasy *Service* zajmuje się odczytywaniem folderu lokalnego aplikacji w poszukiwaniu plików .properties, które zawierają specyfikacje zadań wykonywanych przez nią (więcej na ten temat zawiera rozdział 4 pracy).

Na koniec trzeba wspomnieć, że jak w przypadku aktywności, wszystkie używane serwisy muszą być zdefiniowane w manifeście aplikacji. Definicje StartUpService można zobaczyć na poniższym listingu.

```
1 <service android:name=".service.StartUpService"  
2     android:label="Autotask Service"/>
```

Listing 7. Definicja StartUpService w AndroidManifest.xml.

2.9. Android Support Library

Wraz ze powstawaniem nowych wersji Androida, firma Google postanowiła wydać bibliotekę pozwalającą w łatwy sposób zapewnić wsteczną kompatybilność ze starszymi wersjami systemu. Dlatego przy premierze Androida w wersji 1.6 oddano w ręce developerów Android Support Library. Pozwala ona na nie tylko łatwe pisanie aplikacji wspierających wcześniejsze wydania Androida, ale także dodającą nowe funkcjonalności takie jak opisany w późniejszym rozdziale *LocalBroadcastManager*.

Warto zaznaczyć, że pobierając tę bibliotekę przy użyciu menadżera Android SDK, a następnie dodając ją do projektu aplikacji, trzeba mieć na uwadze fakt, iż każda jej wersja jest kompatybilna (z punktu budowy metod, nie logiki) z określoną wersją Androida.

3. Intencje oraz broadcast receivers

3.1. Intencje

3.1.1. Wstęp

Intencje w Androidzie służą jako mechanizm komunikacyjny między aplikacjami jak i ich komponentami. Stosuje się je do:

- Uruchamiania aktywności oraz serwisów
- Nadawania informacji o zdarzeniach, które pojawiły się na urządzeniu

Są uważane jako jeden z wielu best practice przy programowaniu na Androida. Google zaleca używanie ich nie tylko przy komunikacji system – aplikacja, a także w jej wnętrzu. Pozwalają na bezproblemową modyfikację jednego komponentu programu bez zmiany referencji do niego.

Android powstał jako system, który można łatwo dostosować do własnych potrzeb. Jednym z owoców tego postanowienia jest mechanizm intencji. Używając go w łatwy sposób można napisać aplikacje, którą nadpisuje funkcjonalności już w nim dostępną, dzięki oczekiwaniu na te same akcje, co programy systemowe.

Dzięki intencjom system jest w stanie z na pozór niepołączonych ze sobą komponentów z różnych aplikacji (niepisanych przez jednego autora) – stworzyć jedną całość połączoną przy pomocy systemowi nadajników i odbiorników.

3.1.2. Uruchamianie aktywności i serwisów przy pomocy intencji

Jak zostało wspomniane w poprzednim podrozdziale, intencje między innymi służą do tworzenia nowych aktywności oraz serwisów podczas działania programu. Mamy dwie możliwości ich uruchamiania:

- Bezpośrednio (z ang. *explicitly*), przy pomocy *Class*

- Pośrednio (z ang. *implicitly*), przy użyciu akcji z odpowiednimi danymi (ten sposób może być użyty w miejscu, gdzie nie znamy nazwy widoku, który chcemy wyświetlić)

Pierwszy z wymienionych wcześniej sposobów jest najczęściej używany przy startowaniu aktywności lub serwisu, która znajduje się w tej samym programie. System przy pomocy obiektu typu *Class* tego elementu tworzy go, uruchamia i przenosi go na początek stosu widoków. Nowa aktywność jest obsługiwana przez ten sam wątek, który ją otwierał. Natomiast nowo uruchomiony serwis jest tworzony w tym samym wątku co aplikacja, która go wywołuje, ale po jego poprawnym uruchomieniu działa on w oddzielnym wątku. Poniższy listing zawiera przykładowe jednoznaczne wystartowanie aktywności.

```
1 Intent intent = new Intent(Main.this, TaskListActivity.class);
2 startActivity(intent);
```

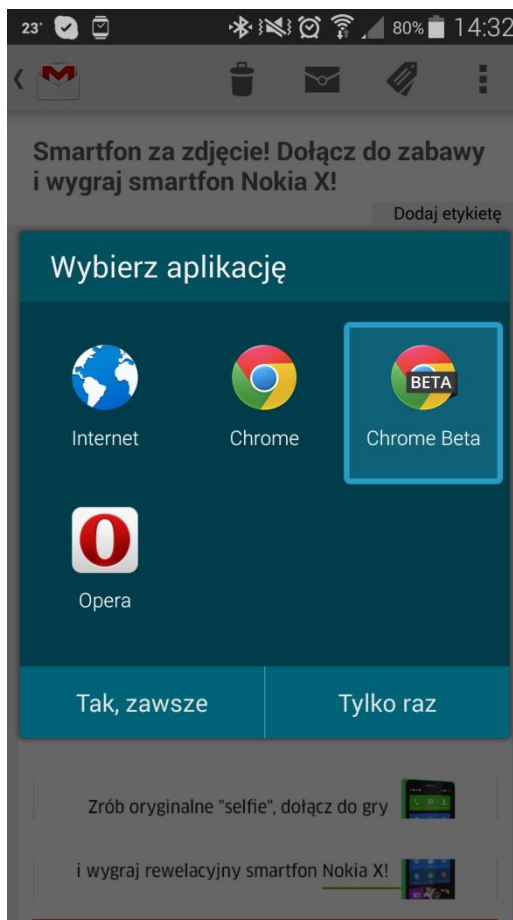
Listing 8. Przykładowe jednoznaczne uruchomienie aktywności.

Innym podejściem do startowania aktywności jest użycie akcji podczas tworzenia intencji. Dzięki temu mechanizmowi programista może powiedzieć systemowi, że potrzebuje uruchomić aplikację, która obsłuży określoną czynność, na przykład zadzwoni pod określony numer telefonu, co można zaobserwować na listingu 9.

```
1 String taskName = intent.getStringExtra(
2     ConstanceFieldHolder.TASK HOLDER_NAME EXTRA);
3 String number = STRING STRING MAP.get(taskName);
4 Uri telUri=Uri.parse(String.format("tel:%s",number));
5 Intent dialIntent=new Intent(Intent.ACTION_DIAL, telUri);
6 context.startActivity(dialIntent);
```

Listing 9. Wybranie numeru telefonu przy użyciu intencji.

Na powyższym listingu można zauważyć, że nie podajemy dokładnej informacji na temat widoku, który chcemy wyświetlić. System po przeparsowaniu zmiennej typu *URI*, którą podajemy jako jeden z paramentów przy tworzeniu intencji wie, iż chcemy wykonać połączenie telefoniczne pod podanym numer telefonu. Z tego powodu uruchomi domyślny dialer lub okno wyboru programu, który ma być odpowiedzialny za wsparcie tego typu treści. Przykładowe okno z wyborem aplikacji jest ukazane na rysunku 3.



Rysunek 3. Okno wyboru domyślnej przeglądarki w systemie.

3.2. Natywne akcje Androida

Poniższa lista zawiera część intencji, które udostępnia klasa *Intent* (w postaci statycznego obiektu typu *String*) z API Androida:

- **ACTION_ALL_APPS** – wyświetla wszystkie zainstalowane aplikacje. Najczęściej używane przez programy typu Launcher
- **ACTION_ANSWER** – odpowiada za wyświetlenie widoku odpowiedzialnego za powiadomienie użytkownika o nadchodzącej rozmowie
- **ACTION_APP_ERROR** – ta akcja jest wysyłana po naciśnięciu przycisku **Report** z dialogu pozwalającego na zaraportowanie błędu w aplikacji
- **ACTION_ASSIST** – wyświetla aplikację typu Asystent, po naciśnięciu i przytrzymaniu przycisku domowego

- ACTION_BUG_REPORT – wyświetla widok odpowiedzialny raportowanie błędu w aplikacji
- ACTION_CALL – pozwala na wykonanie połączenia z określonym numerem telefonu
- ACTION_CALL_BUTTON – umożliwia pokazanie aplikacji pozwalającej na wybranie numeru telefonicznego
- ACTION_CHOOSER – najczęściej używana jako zamiennik do standardowego widoku wybierania aktywności (pokazanego na rysunku 3), która ma zostać wyświetlona dla danego typu akcji, kiedy system posiada wiele aplikacji z danej kategorii. Programista może nadać własny tytuł temu oknu dialogowemu, ale za to użytkownik nie ma możliwości ustawienia wybranego programu jako domyślnej aplikacji dla tej akcji
- ACTION_CREATE_SHORTCUT – tworzy skrót. Zwraca intencję, która musi zawierać informacje o:
 - Nazwie skrótu
 - Intencji, którą ma uruchamiać
 - Ikonicie, jaką ma on posiadać
 lub obiekt typu *Intent.ShortcutIconResource*
- ACTION_DELETE – przeprowadza operacje usunięcia na przekazanych danych
- ACTION_DIAL – działa podobnie do ACTION_CALL, czyli wyświetla element interfejsu użytkownika odpowiedzialny za wybieranie numeru telefonu. Programista ma możliwość podania numeru telefonu, na który ma zostać wykonane połączenie
- ACTION_EDIT – pozwala na edycję wybranego zasobu
- ACTION_GET_CONTENT – pozwala na poproszenie użytkownika o wybranie elementu wyszczególnionego typu danych (na przykład kontaktu) oraz przekazanie go z powrotem do aplikacji
- ACTION_INSERT – tworzy nowy element (na przykład plik, obraz, kontakt) w przekazanej przez programistę ścieżce

- ACTION_INSERT_OR_EDIT – pozwala na stworzenie nowego elementu lub edycję istniejącego
- ACTION_MAIN – jedna z najważniejszych akcji opisanych tutaj. Jest wykorzystywana w większości aplikacji działających na systemie Android. Przy jej pomocy definiuje się główny widok programu
- ACTION_OPEN_DOCUMENT - pozwala za otworzenie jednego lub wielu dokumentów używając zainstalowanych na urządzeniu DocumentProvider
- ACTION_PASTE – przy użyciu danych znajdujących się w schowku podręcznym, tworzy nowy element w wybranym kontenerze
- ACTION_PICK – zwraca ścieżkę do wybranego elementu
- ACTION_POWER_USAGE_SUMMARY – wyświetla widok odpowiedzialny za podanie sumarycznych informacji o zużyciu energii
- ACTION_SEACH – uruchamia widok odpowiedzialny za wyszukiwanie informacji
- ACTION_SEND – wysyła otrzymane dane przy pomocy *EXTRA_TEXT* lub *EXTRA_STREAM* do wybranej przez użytkownika aplikacji z danej kategorii
- ACTION_SEND_MULTIPLE – ta akcja w odróżnieniu od ACTION_SEND pozwala wysłać więcej niż jeden element przy jednorazowym wywołaniu
- ACTION_SET_WALLPAPER – wyświetla widoku, który pozwala na zmianę tła ekranu blokady oraz startowego
- ACTION_WEB_SEARCH – w odróżnieniu od ACTION_SEARCH wyszukuje informacje dostępne tylko w Internecie

Na koniec tego podrozdziału trzeba wspomnieć, że klasa *Intent* nie jest jedynym miejscem w API Androida gdzie możemy spotkać akcje. Inne klasy gdzie mamy je dostępne to między innymi Settings, gdzie są w niej zawarte akcje do widoków dla wszelakich ustawień na urządzeniu.

3.3. Dodawanie dodatkowych informacji do intencji

Jak w poprzednim podrozdziale zostało wspomniane, wiele intencji dostępnych w klasie *Intent* podczas tworzenia, prócz podania ich nazwy komponentu czy akcji, posiada także możliwość dodania dodatkowych danych potrzebnych podczas startu albo działania widoku lub broadcast receivera. Można to wykonać przy pomocy kilku sposobów – przy pomocy obiektu klasy *Uri*, używając mapy zwanej EXTRA lub korzystając z flag.

3.3.1. Uri

Pierwszy typ danych, z którego można korzystać przy tworzeniu intencji jest obiekt klasy *Uri*⁹. Jak sama nazwa klasy wskazuje jej obiekty nie zawierają samej informacji, jedynie odnośnik do niej. W trakcie konstrukcji elementu tego typu trzeba podać obiekty typu *String*, który zawiera informacje o typie oraz miejscu ich przechowywania. Danych w postaci obiektu *Uri* stosuje się na przykład jak chcemy wyświetlić kontakt przy pomocy ACTION_VIEW o nazwie *mama*, taki odnośnik wygląda następująco – `content://contacts/people/mama`.

Warto tutaj zaznaczyć, że jeżeli programista chce wykorzystać ten typ danych podczas tworzenia intencji musi skorzystać z odpowiedniego konstruktora lub jednego z trzech udostępnionych do tego setterów¹⁰. Jeżeli zostanie wybrany drugi sposób oraz zajdzie potrzeba ustawienia jednocześnie typu danych oraz ścieżki do elementu, powinna być użyta funkcja *setDataAndType()*. Jest to spowodowane faktem, iż metody nadające wartość tych zmiennych w klasie *Intent* wzajemnie się wykluczają.

3.3.2. Mapa dodatkowych danych

Innym sposobem dodawania informacji do intencji jest dodawanie ich do mapy reprezentowanej przez obiekt klasy *Bundle*. Zmienna ta może przechowywać element każdego typu łącznie kluczem, który jest stringiem. Klasa

⁹ **URI** (skrót od *Uniform Resource Identifier*) – standard zapisu odnośników, w postaci ciągu znaków, do zasobów w celu ich łatwej identyfikacji. Zostaw stworzony przez Internet Society [14].

¹⁰ **Setter** – zwyczajowo przyjęty typ metody w języku Java pozwalający na ustawianie wartości zmiennych w obiekcie.

Bundle jak i *Intent* posiada szereg metod, które pozwalają zapisać oraz odczytać informacje w nich zawarte. Jednym z best practice stosowanym przy tym podejściu jest zapis kluczy jako statycznych publicznych zmiennych typu `string` w celu ułatwienia korzystania z nich przez inne komponenty.

3.3.3. Flagi

Dzięki flagom w intencji Android może określić jak ma się zachować podczas uruchamiania oraz po zamknięciu widoku. Poprzez ten mechanizm programista może powiedzieć systemowi między innymi, aby ten uruchomił aktywność w tle lub nie zostawił informacji o niej w liście ostatnio uruchamianych aplikacji.

3.4. Podstawowe informacje o *IntentFilters*

W poprzednim podrozdziale pracy zostało pokazane jak użyć intencji do startowania widoków oraz serwisów. Klasa *Intent* ma jeszcze jedno przeznaczenie, mianowicie można przy jej pomocy rozsyłać informacje o wydarzeniach dziejących się na urządzeniu (zostanie to opisane w późniejszym rozdziale pracy). Ale jak Android wie, którego komponentu ma użyć, jeżeli dostanie takie żądanie?

Klasa *IntentFilter* powstała, aby zaradzić temu problemowi. Jej obiekty zawierają informacje mówiące o akcji, które dany komponent może wykonać i na jakich typach danych. Jest także stosowana do przechowywania nazw akcji, o których odbiornik chce otrzymywać powiadomienia.

3.4.1. Jak system rozwiązuje filtry intencji

Intent resolution to proces, który ma na celu znalezienie najlepiej pasującego komponentu do danych otrzymanych z intencji przekazanej poprzez metodę *startActivity()*. Przebiega on następująco:

1. System tworzy listę wszystkich dostępnych obiektów *IntentFilter*
2. Jeżeli jakiś filtr nie pasuje do akcji lub kategorii otrzymanej z intencji jest usuwany z listy. Ten proces wyrzuca te obiekty, które:

- Nie posiadają przynajmniej jednej akcji zdefiniowanej przez intencje
 - Nie posiadają jakiegokolwiek kategorii zdefiniowanej przez intencje. Innymi słowy filtr musi posiadać wszystkie kategorie zawarte w obiekcie klasy *Intent*
3. Jeżeli intencja posiada dane dodatkowe w postaci URI są one porównywane ze schematem danych z filtru intencji. Jeśli przynajmniej jedna z części schematu się nie zgadza z tym co zostało przekazane przez intencje – filtr zostanie odrzucony
 4. Jeśli po zakończeniu tego procesu lista stworzona w pierwszym kroku nie jest pusta, to:
 - Dla widoków, jeżeli będzie istnieć więcej niż jedna aktywność pasująca do otrzymanej intencji, system zaprezentuje użytkownikowi ich listę z możliwością wyboru
 - W przypadku odbiorników – wszystkie otrzymają tą intencje

Warto zaznaczyć, że wszystkie aplikacje (włącznie z systemowymi) przechodzą ten sam proces oraz wszystkie posiadają ten sam priorytet.

3.5. Odbiorniki

Jak zostało napisane w pierwszym rozdziale pracy, osoby odpowiedzialne za stworzenie Androida wiedząc, że ich system będzie działał na urządzeniach z ograniczoną mocą obliczeniową oraz działających na baterii, chcieli w jak największym stopniu ograniczyć wielokrotne przetwarzanie tych samych danych przez różne komponenty w nim zainstalowane. Dziekiem tego podejścia jest mechanizm komunikacji między zainstalowanymi aplikacjami na urządzeniu. Działa on na zasadzie sieci nadajników oraz odbiorników. Broadcast receivers pełnią w tym systemie tą drugą rolę.

3.5.1. Implementacja klasy **BroadcastReceiver**

Jak można zauważyć na poniższym kodzie źródłowym podczas rozszerzania klasy *BroadcastReceiver* programista musi napisać własną implementację metody *onReceive()*. Funkcja ta przyjmuje dwa parametry:

- Obiekt klasy *Context* – będący reprezentacją widoku lub serwisu na rzecz, którego działa odbiornik.
- Intencje - zawierają akcję oraz dane dodatkowe, dzięki którym został uruchomiony.

```
1  public abstract class AbstractBroadcastReceiverTaskObject
2      extends BroadcastReceiver implements TaskObject {
3      protected static Intent responseIntent;
4      protected Boolean activated;
5      @Override
6      public void onReceive(Context context, Intent intent) {
7          Map<Boolean, Set<Intent>> activationSet =
8              receive(context, intent);
9          for (Map.Entry<Boolean, Set<Intent>> activationEntry :
10              activationSet.entrySet()) {
11              for (Intent intentFromEntry :
12                  activationEntry.getValue()) {
13                  intentFromEntry.putExtra(
14                      ConstanceFieldHolder.EXTRA_TRIGGER_ACTIVATED,
15                      activationEntry.getKey());
16                  intentFromEntry.putExtra(
17                      ConstanceFieldHolder.EXTRA_CLASS_NAME,
18                      getClass().getName());
19                  context.sendBroadcast(intentFromEntry);
20              }
21          }
22      }
23      protected abstract Map<Boolean, Set<Intent>> receive(
24          Context context, Intent intent);
```

Listing 10. Przykładowa implementacja broadcast receivera.

3.5.2. Rejestracja odbiornika

Tak jak aktywności oraz serwisy, obiekty typu *BroadcastReceiver* muszą zostać zarejestrowane przez system. Niemniej jednak w odróżnieniu od dwóch pierwszych komponentów Androida, odbiornik używany przez program nie musi powiadamiać o swoim istnieniu w manifeście aplikacji (statycznie), ale także można to zrobić w samym jej kodzie (dynamicznie).

```

1      <receiver android:name=
"org.karolgurecki.autotask.service.StartupBroadcastReceiver">
2          <intent-filter>
3              <action android:name=
"android.intent.action.ACTION_EXTERNAL_APPLICATIONS_AVAILABLE"/>
4              <action android:name=
"android.intent.action.BOOT_COMPLETED"/>
5          </intent-filter>
6      </receiver>

```

Listing 11. Przykład rejestracji broadcast receivera z użyciem AndroidManifest.xml.

Listing 11 przedstawia fragment AndroidManifest.xml odpowiedzialny za rejestrację odbiornika w systemie. Dzięki zdefiniowaniu broadcast receivera w tym pliku, można powiadomić system o istnieniu tego broadcast receivera bez uruchamiania kontekstu aplikacji na rzecz, której jest tworzony. Przy pomocy tego podejścia są one rejestrowane od razu po pomyślnym uruchomieniu Androida, czyli istnieje możliwość stworzenia odbiornika, który będzie czekał na to wydarzenie, co przedstawiono na listingu 11.

Drugim sposobem powiadamiania Androida o istnieniu odbiornika jest użycie do tego kodu aplikacji. Aby go użyć trzeba wykonać następujące kroki:

1. Stworzyć *IntentFilter* zawierający akcję, które chcemy wygłosić
2. Na koniec używając metody *registerReceiver()* z obiektu typu Context

Powyższe kroki można zaobserwować na listingu 11.

```

1 context.registerReceiver(
2     taskObject.getBroadcastReceiver(),
3     taskObject.getIntentFilter());

```

Listing 12. Rejestracja nowego odbiornika w kodzie aplikacji.

Przy dynamicznej rejestracji odbiornika trzeba pamiętać o jego odrejestrowaniu. Jak to nie zostanie wykonane może dojść do wycieku pamięci na urządzeniu. Służy do tego metoda *unregisterReceiver()* pobierająca referencje do broadcast receivera, którego chcemy dezaktywować.

3.5.3. Wysyłanie wiadomości oraz cykl życia odbiornika

Do nadawania wiadomości służy metoda *sendBroadcast()* z klasy *Context*. Poniższy listing prezentuje budowanie intencji wraz z dodatkowymi danymi oraz rozsyłanie jej go przy pomocy powyższej funkcji.

```

1 Intent taskObjectIntent = TaskObject.getIntent();
2 taskObjectIntent.putExtra(

```

```
3 ConstanceFieldHolder.TASK HOLDER_NAME_EXTRA, name);  
4 context.sendBroadcast(taskObjectIntent);
```

Listing 13. Wysyłanie wiadomości o zdarzeniu.

Podczas używania broadcast receiverów warto wiedzieć jak one się zachowują po zakończeniu wykonywania się metody *onReceiver()*. Domyślnie odbiorniki nie mogą wykonywać obliczeń w innym wątku niż ten, z którego są uruchamiane, a po ich zakończeniu system może uwolnić zasoby przez nią używane. W Androidzie 3.0 wprowadzono metodę *goAsync()* pozwalającą na kontynuowanie obliczeń asynchronicznie. Dzięki tej funkcji system będzie mógł usunąć danego broadcast receivera z pamięci niejedynie, kiedy zostanie wywołana metoda *finish()* z obiektu klasy *PendingResult* otrzymanego po wywołaniu funkcji *goAsync()*.

3.6. Lokalne broadcast receivers

Podczas tworzenia aplikacji zdarza się czasami, że programista nie chce lub nie powinien rozgłaszać albo odbierać pewnych informacji z całego urządzenia. Najczęściej jest to spowodowane względami bezpieczeństwa. Wysyłanie wrażliwych informacji w kierunku wszystkich komponentów w systemie może doprowadzić do ich wycieku, a odbieranie wszystkich transmisji danego wydarzenia do nieautoryzowanego sterowania określonego komponentu przez zewnętrzną aplikację.

Z powodu przedstawionego powyżej firma Google postanowiła w Android Support Library dodać klasę *LocalBroadcastManager*. Jej zadaniem jest zarządzanie nadawaniem oraz odbieraniem informacji przekazywanych wewnątrz aplikacji.

Sam proces rejestrowania oraz wysyłania informacji odbywa się bardzo podobnie do korzystania z wersji tego mechanizmu udostępnionej przez klasę *Context*. Trzeba pobrać nową instancję *LocalBroadcastManager* poprzez *getInstance()*, następnie używając otrzymanego obiektu można tak jak w przypadku globalnych zarządzać odbiornikami oraz wysyłać wiadomości. Nazwy metod oraz ich parametry są identyczne do dostępnych w klasie *Context*. Poniższy kod źródłowy przedstawia otrzymanie instancji klasy

LocalBroadcastManager oraz zarejestrowanie odbiornika, natomiast listing 15 ukazuje wysyłanie wiadomości przy użyciu tej klasy.

```

1     private LocalBroadcastManager localBroadcastManager =
2         LocalBroadcastManager.getInstance(this);
3
4     @Override
5     protected void onResume() {
6         super.onResume();
7         localBroadcastManager.registerReceiver(receiver,
8             new IntentFilter(ConstanceFieldHolder.
9                 INTERNAL_ADD_TASK_OBJECT_ACTION));
10    }
11
12    @Override
13    protected void onPause() {
14        super.onPause();
15        localBroadcastManager.unregisterReceiver(receiver);
16    }

```

Listing 14. Otrzymanie nowej instancji *LocalBroadcastManager* oraz rejestracja lokalnego odbiornika wraz z jego odrejestrowaniem.

```

1    Intent intent =
2        new Intent("org.karolgurecki.autotask.addTaskObject");
3    intent.putExtra("INDEX", position);
4    intent.putExtra("TYPE", type);
5    LocalBroadcastManager.getInstance(context).sendBroadcast(intent);

```

Listing 15. Nadanie lokalnej wiadomości.

3.7. Broadcast receivers udostępniane przez Androida

3.7.1. Wydarzenia dostępne w klasie *Intent* do powszechnego użytkowania

Poniższa tabela przedstawia wybrane akcje z klasy *Intent*. Wszystkie nazwy akcji reprezentują nazwy zmiennych typu *String*, którą trzeba podać przy stworzeniu obiektu *IntentFilter* potrzebnego do zarejestrowania odbiornika.

Tabela 3. Wydarzenia z klasy *Intent* dostępne do korzystania w aplikacjach innych niż systemowe.

Nazwa akcji	Opis
ACTION_AIRPLANE_MODE_CHANGED	Wysyłana, kiedy użytkownik włączył lub wyłączył tryb samolotowy. Posiada informacje o stanie, w którym znajduje się ten tryb.
ACTION_BATTERY_CHANGED	Zawiera informacje o stanie baterii. Ten Intent nie może być użyty w <i>AndroidManifest.xml</i>

ACTION_BATTERY_LOW	Wysyłana, kiedy bateria w urządzeniu osiągnęła niski poziom naładowania.
ACTION_BOOT_COMPLETED	Aplikacja otrzymuje tę transmisję, kiedy system zakończył uruchamianie się. Musi posiadać uprawnienie <u>RECEIVE_BOOT_COMPLETED</u> .
ACTION_CONFIGURATION_CHANGED	Wysyłana, kiedy na urządzeniu zmieniła się konfiguracja jak na przykład zmiana języka czy orientacji. Większość programów nie musi go implementować, ponieważ system sam zastosuje zmiany poprzez restart aplikacji.
ACTION_HEADSET_PLUG	Informuje, że słuchawki przewodowe zostały podpięte lub odłączone od urządzenia. Zawiera dane o ich stanie, nazwie oraz czy posiadają mikrofon.
ACTION_NEW_OUTGOING_CALL	Wysyłana, kiedy z urządzenia wychodzi połączenie głosowe. Zawiera informacje o numerze, który został wybrany
ACTION_POWER_CONNECTED	Informuje, że ładowarka została podłączona do urządzenia.
ACTION_POWER_DISCONNECTED	Wysyłana, kiedy ładowarka została odłączona od urządzenia.
ACTION_REBOOT	Oznacza, że urządzenie zostanie zrestartowane. Jest używany tylko przez komponenty systemowe.
ACTION_SCREEN_OFF	Oznacza, że urządzenie jest w stanie uśpienia. Warto zaznaczyć, iż nie koniecznie informuje to o stanie działania ekranu urządzenia. Aby sprawdzić stan wyświetlacza trzeba użyć metody <i>getState()</i> z klasy <i>Display</i> .

ACTION_SHUTDOWN	Wysyłana, kiedy urządzenie się wyłącza.
ACTION_TIME_CHANGED	Informuje, że czas został zmieniony w ustawieniach urządzenia.
ACTION_TIME_TICK	Wysyłana co minutę. Informuje o zmianie czasu.

3.7.2. Przykłady wydarzeń z innych klas dostępnych w API Androida

Poniższa tabela przedstawia wybrane akcje z innych klas API Androida. Tak jak w poprzednim podrozdziale wszystkie nazwy akcji reprezentują nazwy zmiennych typu *String* (wraz z nazwą klasy, z której pochodzi), którą trzeba podać przy stworzeniu obiektu *IntentFilter* potrzebnego do zarejestrowania odbiornika.

Tabela 4. Wybrane wydarzenia z innych klasy Android API.

Nazwa akcji	Opis
BluetoothDevice.ACTION_BOND_STAGE_CHANGED	Informuje, że stan połączenia z urządzeniem zewnętrznym przez interfejs Bluetooth się zmienił.
BluetoothDevice.ACTION_FOUND	Wysyłana, kiedy nowe urządzenia zostały wykryte.
BluetoothDevice.ACTION_PAIRING_REQUEST	Używana do wysłania prośby o połączenie się z innym urządzeniem przez interfejs Bluetooth. Wymaga, aby aplikacja posiadała uprawnienie <u>BLUETOOTH_ADMIN</u> .
BluetoothAdapter.ACTION_LOCAL_NAME_CHANGED	Informuje, że nazwa urządzenia widziana przez inne urządzenia Bluetooth została zmieniona.
BluetoothAdapter.ACTION_SCAN_MODE_CHANGED	Wysyłana, kiedy tryb wyszukiwania nowych urządzeń Bluetooth się zmienił.

CHANGED	Zawiera dane o aktualnym stanie oraz poprzednim.
BluetoothAdapter. ACTION_STATE_CHANGED	Informuje, że moduł Bluetooth na urządzeniu został włączony lub wyłączony. Zawiera dane o aktualnym stanie oraz poprzednim.
NfcAdapter. ACTION_ADAPTER_STATE_CHANGED	Informuje, że moduł NFC na urządzeniu został włączony lub wyłączony.
Telephony.Sms.Intents. SIM_FULL_ACTION	Informuje, że pamięć SIM dla krótkich wiadomości jest pełna.
Telephony.Sms.Intents. SMS_DELIVER_ACTION	Wysyłana, kiedy wiadomość SMS została dostarczona do nadawcy.
Telephony.Sms.Intents. WAP_PUSH_RECEIVED_ACTION	Oznacza, że urządzenie otrzymało wiadomość typu <u>WAP PUSH</u> .
WifiManager. NETWORK_STATE_CHANGED_ACTION	Informuje o zmianie stanu połączenia z punktem dostępowym Wi-Fi.
WifiManager. WIFI_STATE_CHANGED_ACTION	Informuje, że moduł Wi-Fi na urządzeniu został włączony lub wyłączony. Zawiera dane o aktualnym stanie oraz poprzednim.
UsbManager. ACTION_USB_ACCESSORY_ATTACHED	Wysyłana, kiedy urządzenie zostało podłączone z innym przez interfejs USB w trybie <u>accessory</u> .
UsbManager. ACTION_USB_DEVICE_ATTACHED	Wysyłana, kiedy urządzenie zostało podłączone z innym przez interfejs USB w trybie <i>host</i> .

4. Opis praktycznej części pracy

4.1. Opis aplikacji

Program będący praktyczną częścią mojej pracy, ma za zadanie ukazanie możliwości mechanizmu intencji oraz odbiorników. Działa on na systemie Android w wersji przynajmniej 4.1 „Jelly Bean”.

4.1.1. Funkcjonalność aplikacji

Funkcjonalność programu to przede wszystkim tworzenie oraz usuwanie zadań zdefiniowanych przez użytkownika. Każde zadanie reprezentowane jest w kodzie poprzez klasę *TaskHolder*. Jeden taki obiekt składa się z trzech głównych elementów: nazwy zdania oraz dwóch list – wyzwalaczy (triggers) oraz akcji (actions). Użytkownik używając interfejsu aplikacji wybiera, jakie wyzwalacze oraz akcje mają się znaleźć w zadaniu. Ich ilość oraz kolejność może być dowolna. Lista dostępnych obiektów do wyboru wygląda następująco:

- Wyzwalacze:
 - Zmiana stanu modułu Wi-Fi
 - Zmiana stanu modułu Bluetooth
 - Zmiana stanu modułu NFC
 - Połączenie głosowe wychodzące
 - Połączenie głosowe przychodzące
- Akcje:
 - Wyświetlenie wiadomości na ekranie w postaci toastu
 - Wybranie podanego numeru telefonu
 - Otworzenie określonej strony WWW w przeglądarce
 - Odrzucenie połączenia
 - Włączenie wibracji na urządzeniu

4.1.2. Architektura programu

Architektura aplikacji opiera się o wzorzec projektowy fabryki abstrakcyjnej. Klasa odpowiedzialna za tworzenie zadań używa fabryki *TaskFactory* do utworzenia obiektów implementujących interfejsu *TaskObject*, według specyfikacji odczytanej z pliku .properties. Zawiera on:

- Nazwę zadania
- Listę wyzwalaczy z konfiguracją
- Listę akcji wraz z ich ustawieniami.

Przykładowy plik przedstawia poniższy listing.

```
1  #This file is auto generated by AutoTask DO NOT CHANGE IT!
2  #Mon Aug 04 22:18:38 CEST 2014
3  name=Testowe zadanie
4  action.classes=org.karolgurecki.autotask.tasks.actions.ToastAction
5  org.karolgurecki.autotask.tasks.actions.ToastAction.config=
Modu\u0142y wifi oraz bluetooth zosta\u0142y w\u0142\u0105czone\!
6  org.karolgurecki.autotask.tasks.triggers.WifiTrigger.config=3
7  trigger.classes=org.karolgurecki.autotask.tasks.triggers.Bluetooth
Trigger,org.karolgurecki.autotask.tasks.triggers.WifiTrigger
8  org.karolgurecki.autotask.tasks.triggers.BluetoothTrigger.config=12
```

Listing 16. przykład pliku properties zawierający specyfikację zadania.

To co jest opisane w pliku pokazanym powyżej w kodzie aplikacji jest reprezentowane przez klasę *TaskHolder*.

Interfejs *TaskObject* posiada dwie podstawowe implementacje, różniące się sposobem pozyskiwania informacji o zdarzeniach – *AbstractBroadcastReceiverTaskObject* oraz *AbstractThreadTaskObject*. Wyzwalacze mogą być dziećmi jednej z powyższych klas abstrakcyjnych, natomiast akcie muszą dziedziczyć po tej pierwszej. Spowodowane jest to użyciem *sendBroadcast()* do aktywacji akcji przez klasę *TaskHolder*.

Poniższy schemat UML przedstawia relacje między klasami w projekcie. Z diagramu zostały wykluczone klasy stworzone na potrzeby interfejsu użytkownika.

4.2. Opis procesu budowania i instalacji aplikacji na urządzeniu

4.2.1. Wymagania

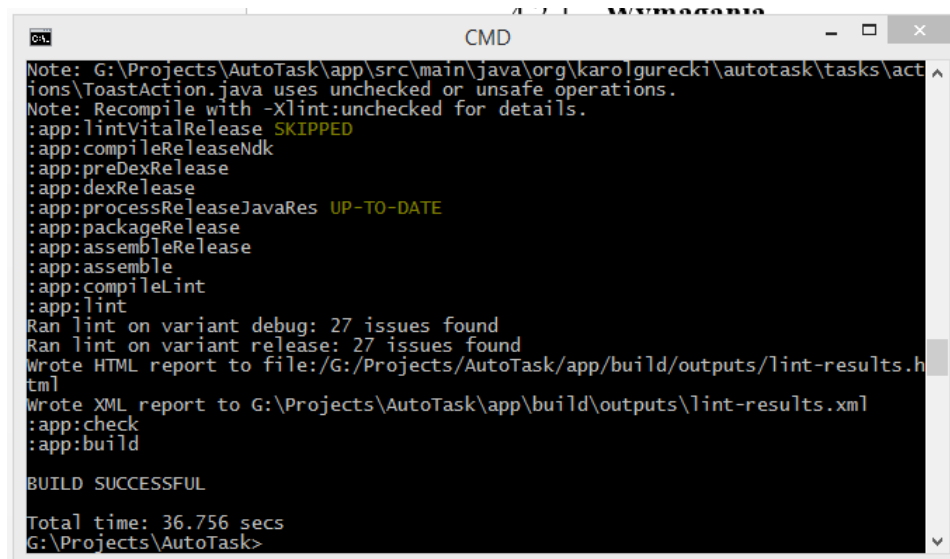
Aby zbudować projekt aplikacji na komputerze trzeba posiadać zainstalowane następujące oprogramowanie:

- Android SDK Build tools w wersji 19.1
- Gradle w wersji 1.12
- Java Standard Edition Developer Kit 7u51

Przed rozpoczęciem budowy programu trzeba upewnić się, że Gradle jest dostępne z wiersza poleceń oraz ścieżka do katalogu z Android SDK jest przypisana do zmiennej środowiskowej `ANDROID_HOME`.

4.2.2. Proces budowy projektu

Aby zbudować aplikację wystarczy przejść do folderu głównego projektu używając wiersza poleceń. Następnie wpisać komendę `gradle build`. Po pomyślnym wykonaniu procedury zostanie wyświetlony komunikat to potwierdzający – jak na rysunku 5.



```
CMD
Note: G:\Projects\AutoTask\app\src\main\java\org\karolgurecki\autotask\tasks\actions\ToastAction.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
:app:lintVitalRelease SKIPPED
:app:compileReleaseNdk
:app:preDexRelease
:app:dexRelease
:app:processReleaseJavaRes UP-TO-DATE
:app:packageRelease
:app:assembleRelease
:app:assemble
:app:compileLint
:app:lint
Ran lint on variant debug: 27 issues found
Ran lint on variant release: 27 issues found
Wrote HTML report to file:/G:/Projects/AutoTask/app/build/outputs/lint-results.html
Wrote XML report to G:\Projects\AutoTask\app\build\outputs\lint-results.xml
:app:check
:app:build

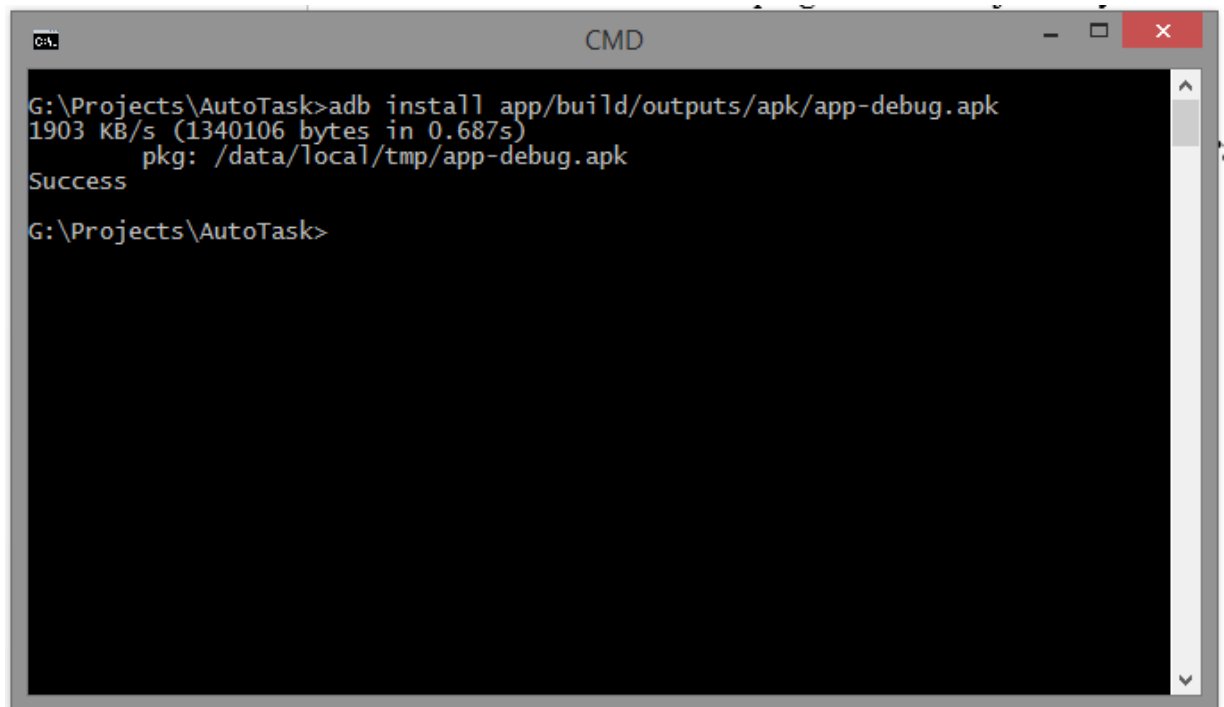
BUILD SUCCESSFUL
Total time: 36.756 secs
G:\Projects\AutoTask>
```

Rysunek 5. Potwierdzenie pomyślnego zbudowania projektu.

4.2.3. Instalacja zbudowanej aplikacji przy użyciu narzędzia adb

Aby zainstalować zbudowaną aplikację na urządzeniu, należy:

1. Podłączyć to urządzenie do komputera używając kabla USB.
2. W wierszu poleceń, będąc w główny folderze projektu, wpisać komendę `adb install app/build/outputs/apk/app-debug.apk` (rysunek 6).
3. Ikona aplikacji powinna się pojawić na liście zainstalowanych programów –jak na rysunku 7.



```
CMD
G:\Projects\AutoTask>adb install app/build/outputs/apk/app-debug.apk
1903 KB/s (1340106 bytes in 0.687s)
pkg: /data/local/tmp/app-debug.apk
Success
G:\Projects\AutoTask>
```

Rysunek 6. Instalacja aplikacji przy użyciu adb.



Rysunek 7. AutoTask wśród zainstalowanych aplikacji.

4.3. Przykładowe użycie programu

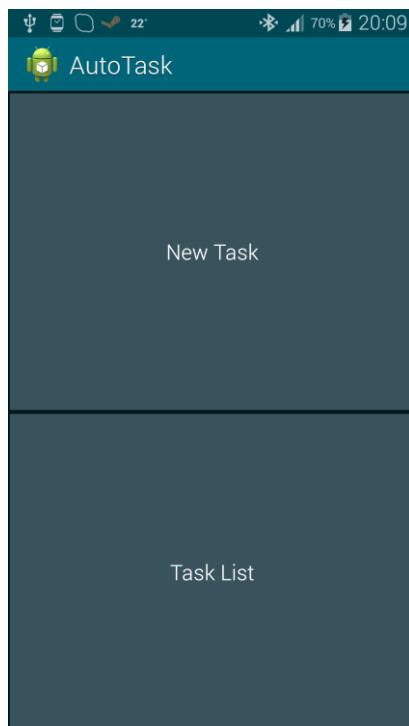
4.3.1. Tworzenie zadania

Proces tworzenia nowego zadania przebiega następująco:

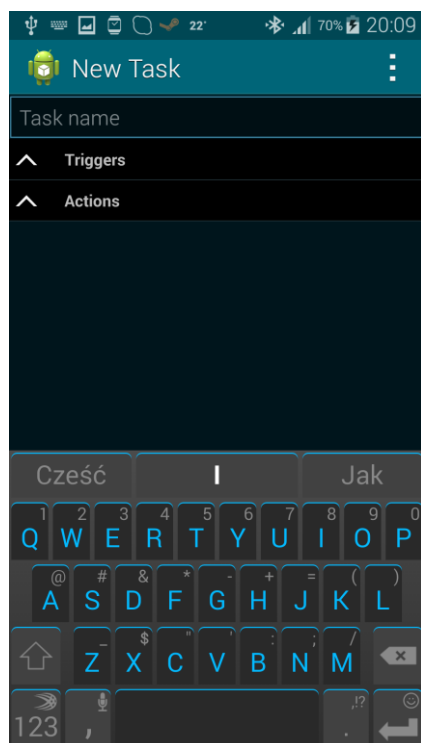
1. Z ekranu głównego aplikacji (rysunek 8) wybieramy przycisk New Task. Otworzy nam się okno wyglądające podobnie do rysunku 9.
2. Żeby wybrać wyzwalacz, z menu wybieramy Add trigger. Otworzy nam się dialog z listą dostępnych obiektów do wyboru (rysunek 10).
3. Po wybraniu triggera zostaniemy poproszeni o jego skonfigurowanie. W przypadku rysunku 11 jest to pytanie, kiedy ten wyzwalacz ma się uaktywnić – przy włączonym module Bluetooth czy przy wyłączonym.

4. Przy akcjach postępujemy podobnie. Jedyną różnicą jest wybór w menu opcji Add action. Rysunek 12 przedstawia listę dostępnych akcji, natomiast rysunek 13 ukazuje dialog konfiguracyjny Toast Action.
5. Po wybraniu wszystkich potrzebnych akcji oraz wyzwalaczy wpisać nazwę zadania w pole znajdujące się na górze widoku – rysunek 14.
6. Na koniec zapisujemy to zadanie używając przycisku wstecz (lub opcji Finish z menu) i zatwierdzając zapis (rysunek 15).

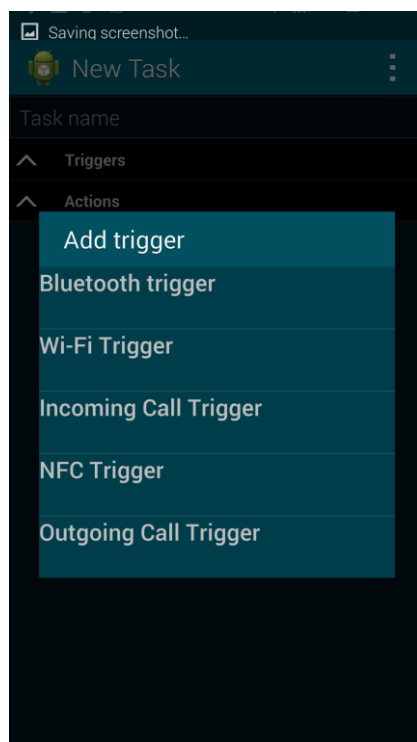
Po wykonaniu się wszystkich triggerów (w przykładzie jest to uruchomienie modułów Wi-Fi oraz Bluetooth) zostaną wykonane wybrane akcje. Przedstawia to rysunek 16.



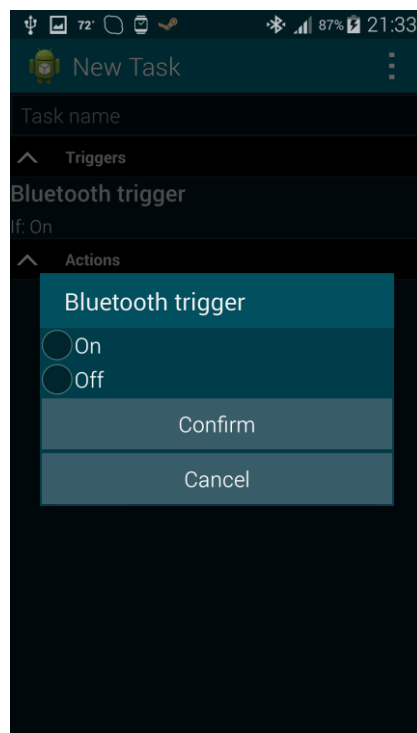
Rysunek 8. Główna aktywność aplikacji.



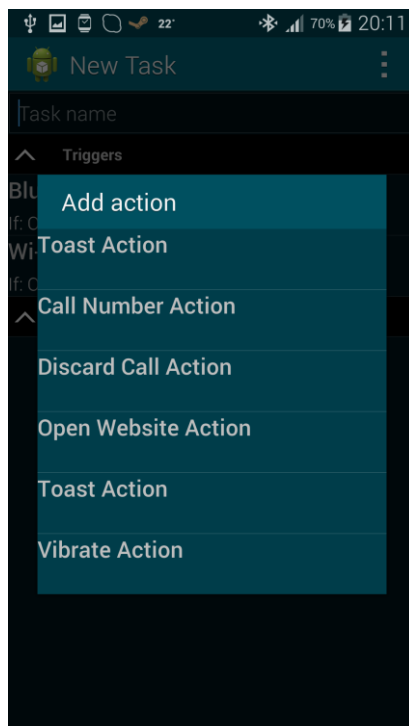
Rysunek 9. Wygląd widoku przy pomocy, którego tworzy się zadania.



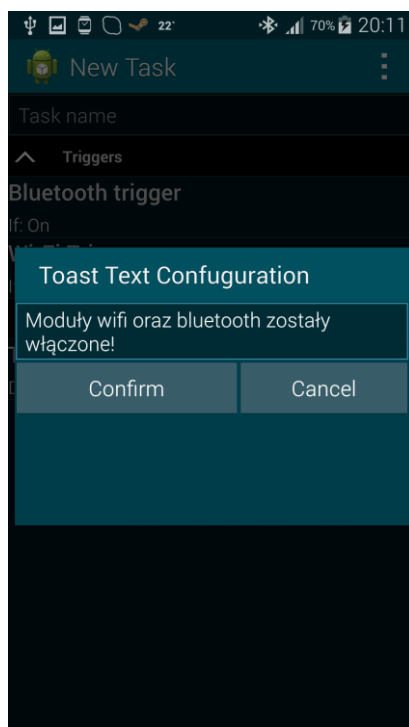
Rysunek 10. Dialog z listą dostępnych wyzwalaczy.



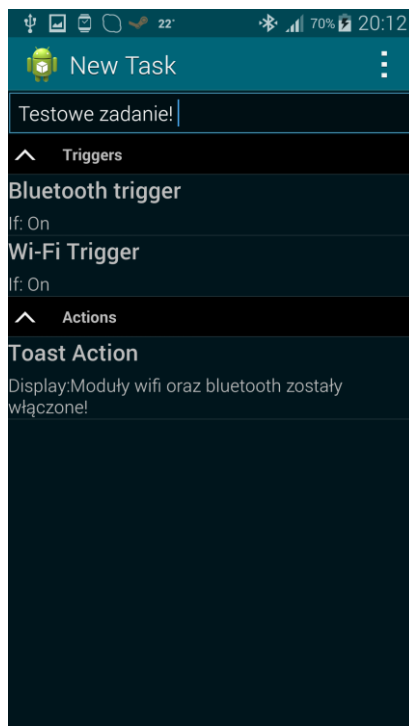
Rysunek 11. Konfiguracja wyzwalacza stanu modułu Bluetooth.



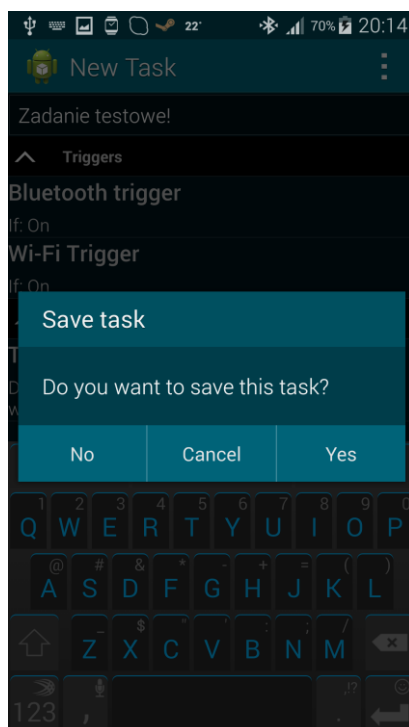
Rysunek 12. Lista dostępnych akcji.



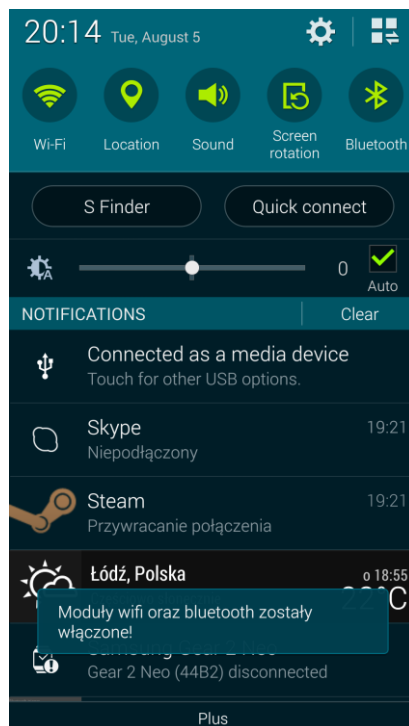
Rysunek 13. Konfiguracja "Toast Action".



Rysunek 14. Zadanie z wybraną akcją i wyzwalaczami oraz wpisaną nazwą.



Rysunek 15. Potwierdzenie zapisania zadania.



Rysunek 16. Wykonanie akcji po aktywacji wszystkich triggerów.

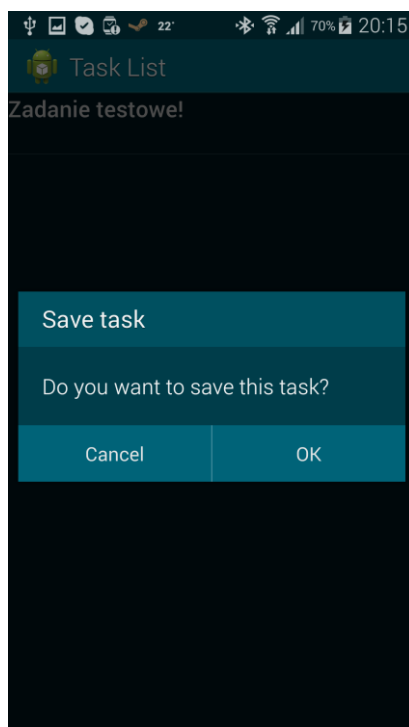
4.3.2. Usuwanie zadania

By usunąć akcje, trzeba wykonać następujące kroki:

1. Z głównej aktywność wybrać przycisk *Task List* (rysunek 17).
2. Po odnalezieniu zadania na liście, trzeba nacisnąć oraz przytrzymać go do momentu ukazania się komunikatu potwierdzającego usunięcie.
3. Zatwierdzić usunięcie zadania – jak na rysunku 18.



Rysunek 17. Lista aktywnych zadań.



Rysunek 18. Dialog z potwierdzeniem usunięcia zadania.

5. Podsumowanie pracy

W powyższej pracy zaprezentowałem możliwości klas *Intent* oraz *BroadcastReceiver*. Na podstawie przedstawionej wiedzy można stwierdzić, że brak tych klas w API systemu Android mógłby spowodować wielkie szkody. Sprawne działanie wielu komponentów systemowych jak i napisanych przez innych developerów jest możliwa dzięki tylko i wyłącznie zastosowaniu tych mechanizmów.

Dzięki intencjom możemy otworzyć stronę WWW w przeglądarce, nie wiedząc jaka jest domyślna w systemie.. Wystarczy, że uruchomimy widok pośrednio, korzystając z obiektu *Intent* stworzonego korzystając z akcji *Intent.ACTION_VIEW* wraz z odpowiednio utworzonym *Uri*.

Także korzystanie z klasy *BroadcastReceiver* ułatwia pisanie programów pod Androida oraz zmniejsza ich zapotrzebowanie na procesor i pamięć operacyjną. W systemie od firmy Google nie trzeba tworzyć nowych wątków do sprawdzenia takich rzeczy jak: stan modułu Wi-Fi, oczekiwanie na przychodzące połączenia głosowe.

Mechanizm stworzony przez twórców Androida posiada wiele zalet, ale także i wad. Największą z nich jest tworzenie nowego obiektu danego odbiornika za każdym razem, kiedy nastąpi akcja, na którą wyczekuje. Dużo czasu spędziłem nad szukaniem błędu w mojej aplikacji, kiedy wartości zmiennych ustawione po otrzymaniu transmisji nie były dostępne na obiekcie przy pomocy, którego zarejestrowałem odbiornik. Rozwiązaniem tego było uczynienie tych zmiennych – zmiennymi statycznymi, które mają taką samą wartość we wszystkich instancjach.

Bibliografia

- [1] *Worldwide Smartphone Market Grows 28.6% Year Over Year in the First Quarter of 2014, According to IDC* [online]. [Framingham]: IDC Corporate USA, 2014 [dostęp na dzień 25.06.2014], dostępny w Internecie: <http://www.idc.com/getdoc.jsp?containerId=prUS24823414>
- [2] *Api Guides* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 25.06.2014], dostępny w Internecie: <http://developer.android.com/guide/index.html>
- [3] *Training* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 25.06.2014], dostępny w Internecie: <http://developer.android.com/training/index.html>
- [4] *Android BroadcastReceiver - Tutorial* [online]. [Hamburg]: Vogella, 2013 [dostęp na dzień 25.06.2014], dostępny w Internecie: http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html#resources_general
- [5] Meier, R. *Professional Android 4 Application Development*, Wyd. 3, Indianapolis, 2013, ISBN 978-1118102275, s. 53-200 oraz 665-737
- [6] Conder Shane, Darcey Lauren. *Android. Programowanie aplikacji na urządzenia przenośne*, Wyd. 2, Helion, Gliwice, 2012, ISBN 978-83-246-3349-4, s 121-122
- [7] *Google Android prototypes debut at MWC* [online]. [Nowy York]: CBS Interactive Inc., 2008 [dostęp na dzień 01.07.2014], dostępny w Internecie: <http://www.cnet.com/news/google-android-prototypes-debut-at-mwc/>
- [8] *A brief history of Android phones* [online]. [San Francisco]: CBS Interactive Inc., 2011 [dostęp na dzień 01.07.2014], dostępny w Internecie: <http://www.cnet.com/news/a-brief-history-of-android-phones/>
- [9] *Google I/O: Android stands at one billion active users and counting* [online]. [San Francisco]: CBS Interactive Inc., 2014 [dostęp na dzień 01.07.2014], dostępny w Internecie:

<http://www.zdnet.com/google-io-android-stands-at-one-billion-active-users-and-counting-7000030881/#ftag=RSSf468ffe?ref=synergymx>

- [10] *Q1 2014 Smartphone OS Results: Android Dominates High Growth Developing Markets* [online]. [Londyn]: Allied Business Intelligence, Inc. 2014 [dostęp na dzień 01.07.2014], dostępny w Internecie:
<https://www.abiresearch.com/press/q1-2014-smartphone-os-results-android-dominates-hi>
- [11] *Activities* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 2.07.2014], dostępny w Internecie:
<http://developer.android.com/guide/components/activities.html>
- [12] *Android Intents - Tutorial* [online]. [Hamburg]: Vogella, 2014 [dostęp na dzień 16.07.2014], dostępny w Internecie:
<http://www.vogella.com/tutorials/AndroidIntent/article.html>
- [13] *Intent* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:
<http://developer.android.com/reference/android/content/Intent.html>
- [14] *Uniform Resource Identifiers (URI): Generic Syntax* [online]. [Wiehle Avenue]: Internet Society, 1998 [dostęp na dzień 29.07.2014], dostępny w Internecie:
<http://tools.ietf.org/pdf/rfc2396.pdf>
- [15] *Android BroadcastReceiver - Tutorial* [online]. [Hamburg]: Vogella, 2013 [dostęp na dzień 30.07.2014], dostępny w Internecie:
<http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html>
- [16] *BluetoothAdapter* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:
<http://developer.android.com/reference/android/bluetooth/BluetoothAdapter>
- [17] *BluetoothManager* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:
<http://developer.android.com/reference/android/content/BluetoothManager>
- [18] *Telephony.Sms.Intents* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:

<https://developer.android.com/reference/android/provider/Telephony.Sms.Intents.html>

- [19] *WifiManager* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:

<http://developer.android.com/reference/android/net/wifi/WifiManager.html>

- [20] *UsbManager* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:

<http://developer.android.com/reference/android/hardware/usb/UsbManager.html>

- [21] *Gradle – build automation evolved.* [online]. [Nowy York]: Gradleware Inc., 2013 [dostęp na dzień 20.07.2014], dostępny w Internecie:

<http://gradle.org>

- [22] *Google positions Gradle as the build system of choice for Android* [online]. [San Francisco InfoWorld Media Group, 2013 [dostęp na dzień 01.07.2014], dostępny w Internecie:

<http://www.infoworld.com/t/development-tools/google-positions-gradle-the-build-system-of-choice-android-218852>

- [23] *Android Support Library Tutorial* [online]. [Hyderabad]: tutorialspoint, 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:

http://www.tutorialspoint.com/android/android_support_library.htm

Spis rysunków

1. Przykładowy zrzut ekranu z Intelij Android Studio.	9
2. Cykl życia aktywności. Źródło: [11].	13
3. Okno wyboru domyślnej przeglądarki w systemie.	23
4. Diagram UML klas związanych z tworzeniem i wykonywaniem zadań.	38
5. Potwierdzenie pomyślnego zbudowania projektu.	39
6. Instalacja aplikacji przy użyciu adb.	40
7. AutoTask wśród zainstalowanych aplikacji.	41
8. Główna aktywność aplikacji.	43
9. Wygląd widoku przy pomocy, którego tworzy się zadania.	43
10. Dialog z listą dostępnych wyzwalaczy.	44
11. Konfiguracja wyzwalacza stanu modułu Bluetooth.	44
12. Lista dostępnych akcji.	45
13. Konfiguracja "Toast Action".	45
14. Zadanie z wybraną akcją i wyzwalaczami oraz wpisaną nazwą.	46
15. Potwierdzenie zapisania zadania.	46
16. Wykonanie akcji po aktywacji wszystkich triggerów.	47
17. Lista aktywnych zadań.	48
18. Dialog z potwierdzeniem usunięcia zadania.	48

Spis tabel

1. Przykładowe znaczniki, które są dostępne w Android Manifest	11
2. Przykładowe uprawnienie, które udostępnia Android.....	14
3. Wydarzenia z klasy Intent dostępne do korzystania w aplikacjach innych niż systemowe.	32
4. Wybrane wydarzenia z innych klasy Android API.	34

Spis listingów zawierających kod źródłowy

1. Przykłady plik build.gradle.....	10
2. Przykładowy plik AndroidManifest.xml.	12
3. Tworzenie Activity przy pomocy kodu.	13
4. Przykładowy plik XML zawierający informacje na temat układu widoku.	16
5. Przykładowy plik strings.xml.	17
6. Fragment StartUpService.java.	19
7. Definicja StartUpService w AndroidManifest.xml.	20
8. Przykładowe jednoznaczne uruchomienie aktywności.	22
9. Wybór numeru telefonu przy użyciu intencji.	22
10. Przykładowa implementacja broadcast receivera.	29
11. Przykład rejestracji broadcast receivera z użyciem AndroidManifest.xml.	30
12. Rejestracja nowego odbiornika w kodzie aplikacji.	30
13. Wysyłanie wiadomości o zdarzeniu.	31
14. Otrzymanie nowej instancji LocalBroadcastManager oraz rejestracja lokalnego odbiornika wraz z jego odrejestrowaniem.	32
15. Nadanie lokalnej wiadomości.	32
16. przykład pliku properties zawierający specyfikację zadania.	37