|  |  |  |
| --- | --- | --- |
| wordml://75.png | **Politechnika Gdańska**  **Wydział Fizyki Technicznej i Matematyki Stosowanej** | wordml://76.png |
|  | | |
| |  |  | | --- | --- | | Katedra/Zakład: | Fizyki teoretycznej i informatyki kwantowej | | Kierunek studiów: | Fizyka Techniczna | | Specjalność: | Informatyka stosowana | | Rodzaj studiów: | stacjonarne | | Imię i nazwisko: | Maja Pyś | | Numer albumu: | 109769 | |  |  | | | |
| **PRACA DYPLOMOWA** | | |
| **INŻYNIERSKA** | | |
| Temat pracy: | | |
| Adaptacja aplikacji mobilnej "e-Doświadczenia w fizyce" do najnowszej wersji systemu Android | | |
| Zakres pracy: | | |
| ........................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................ | | |
|  | | |
| |  |  |  | | --- | --- | --- | | Potwierdzenie przyjęcia pracy: | |  | |  |  |  | | Opiekun pracy |  | Kierownik Katedry/Zakładu | | dr inż. Paweł Syty |  | prof. dr hab. Józef E. Sienkiewicz, prof. zw. PG | | ......................................... |  | ......................................... | | Tytuł, imię i nazwisko |  | Tytuł, imię i nazwisko | |  | Gdańsk, 18.12.2012r. |  | | | |

Spis treści

[Streszczenie 4](#_Toc346894557)

[Wstęp 4](#_Toc346894558)

[Rozdział 1 5](#_Toc346894559)

[Aplikacja z punktu widzenia systemu Android 5](#_Toc346894560)

[Architektura aplikacji w systemie Android 5](#_Toc346894561)

[Aktywności 6](#_Toc346894562)

[Usługi 8](#_Toc346894563)

[Dostawcy treści 12](#_Toc346894564)

[Odbiorcy komunikatów 14](#_Toc346894565)

[Struktura APK 16](#_Toc346894566)

[manifest aplikacji 16](#_Toc346894567)

[kod wykonywalny 17](#_Toc346894568)

[zasoby 17](#_Toc346894569)

[Rozdział 2 19](#_Toc346894570)

[aplikacja mobilna "e-Doświadczenia w fizyce" 19](#_Toc346894571)

[Rozdział 3 19](#_Toc346894572)

[Pobieranie i instalacja oprogramowania do obsługi technologii Adobe Flash 19](#_Toc346894573)

[Losowanie portu serwera WWW 20](#_Toc346894574)

[Mechanizm sprawdzania dostępnych aktualizacji e-doświadczeń 20](#_Toc346894575)

[Dodanie zrealizowanych e-d 22](#_Toc346894576)

[Uzupełnienie opisów oraz przykładowych ćwiczeń 22](#_Toc346894577)

[pomoc aplikacji 22](#_Toc346894578)

[pionowe przewijanie zdjęć w oknach popup wraz informacjami o realizatorach projektu 22](#_Toc346894579)

[Poprawienie grafiki aplikacji wyświetlanej podczas pracy z uruchomionym e-doświadczeniem. 22](#_Toc346894580)

[Zmiana wielkości wyświetlanego opisu doświadczenia 23](#_Toc346894581)

[Rozdział 4 24](#_Toc346894582)

[Analiza pojawiania się i przyczyny błędów związanych z Html.fromHtml() występujących na Androidzie 4.0 i wcześniejszych wersjach systemu. 24](#_Toc346894583)

[Analiza błędów zwiazane z obsługą tapnięć w komponencie webview. 24](#_Toc346894584)

[Analiza sposobów modyfikacji grafiki aplikacji dedykowanej dla tabletów o wysokiej rozdzielczości(powyżej 1280x800). 24](#_Toc346894585)

[Analiza sensowności umieszczania w aplikacji e-doświadczeń wymagających dużej mocy obliczeniowej (wahadło matematyczne, ruch ciał niebieskich). 24](#_Toc346894586)

[Zakończenie 25](#_Toc346894587)

[Dodatek A 25](#_Toc346894588)

[Spis rysunków 25](#_Toc346894589)

# Streszczenie

……

……

……

……

# Wstęp

Przedmiotem pracy inżynierskiej jest zmodyfikowanie istniejącej aplikacji mobilnej umożliwiającej przeglądanie oraz korzystanie z e-doświadczeń, produktu projektu "e-Doświadczenia w fizyce", realizowanego przez Politechnikę Gdańską, Wydział Fizyki Technicznej i Matematyki Stosowanej, we współpracy z firmą Young Digital Planet SA oraz LCG Malmberg BV.

Android to system operacyjny dla urządzeń mobilnych takich jak telefony komórkowe, smartfony, tablety i netbooki. Przeprowadzenie doświadczenia fizycznego na ekranie telefonu komórkowego jest niemożliwe lub bardzo utrudnione ze względu na niewielkie wymiary ekranu. Z tego względu aplikacja przeznaczona jest na urządzenia większe, czyli tablety i netbooki. Testy aplikacji zostały przeprowadzone tylko na tabletach.

Mobilna aplikacja "e-Doświadczenia w fizyce" jest przydatnym narzędziem zapewniającym szybki dostęp do tworzonych produktów. Ułatwia promocję i upowszechnianie e-doświadczeń w środowisku nauczycieli i uczniów. W połączeniu z postępem technologicznym, coraz szerzej wykorzystywanymi komputerami w nauczaniu(e-learningiem) tworzy narzędzie, które w sprzyjających okolicznościach można bezpośrednio wykorzystać w trakcie zajęć edukacyjnych w szkołach powszechnych. Zadania realizowane w ramach pracy inżynierskiej dzielą się na dwie grupy:

1. uzupełnienie funkcjonalności aplikacji,
2. naprawy błędów występujących na nowszych wersjach systemu android (od 4.03 wzwyż).

# Rozdział 1

Od 2005 roku rozwijany jest system operacyjny Android przeznaczony na urządzenia mobilne. Oparty został na Linuksie, którego warstwa jest dla użytkownika niedostępna oraz aplikacjach na licencji GNU. Cały system ….

## Aplikacja z punktu widzenia systemu Android

W systemie operacyjnym Android silny nacisk położony jest na maksymalne ograniczenie praw dostępu poszczególnych programów. Pojedyncza aplikacja instalowana na urządzeniu mobilnym postrzegana jest przez system jak pojedynczy użytkownik. Poszczególne uruchamiane aplikacje wyposażone zostają we własne procesy robocze niewspółdzielone z innymi aplikacjami. Proces aplikacji uruchamiany jest ze startem aplikacji i zabijany w przypadku zakończenia pracy z aplikacją lub gdy zachodzą inne okoliczności tego wymagające np.: potrzeba odzyskania zużywanej pamięci na potrzeby podstawowych zadań systemu – komunikacji z użytkownikiem. Każdy proces jest uruchamiany wraz z nową, dedykowaną maszyną wirtualną. W ten sposób w systemie Android uzyskano całkowitą izolację wykonywania poszczególnych aplikacji.

## Architektura aplikacji w systemie Android

Aplikacje przeznaczone dla system Android konceptualnie budowane są z czterech podstawowych modułów o rozdzielnych funkcjonalnościach, są to:

* aktywności[[1]](#footnote-1),
* usługi[[2]](#footnote-2),
* dostawcy treści[[3]](#footnote-3),
* odbiorcy komunikatów[[4]](#footnote-4).

Moduły aplikacji: aktywności, usługi, odbiorcy komunikatów są aktywowane przez tzw. intencje[[5]](#footnote-5). Intencja (zamierzenie) jest abstrakcyjnym opisem operacji, która dopiero zostanie wykonana. Może być używana do wielu zadań, np.: do powiązania komponentu z usługą albo komunikacji z usługą działającą w tle, albo połączenia ze sobą aktywności w aplikacji.

Pośród dodatkowych informacji, które zawiera intencja może, ale nie musi znajdować się nazwa komponentu do którego jest skierowana. W aplikacji może znajdować się wiele komponentów odpowiadających wymaganiom intencji, ale też może się zdarzyć, że nie będzie żadnego. Dlatego, przewidziane w aplikacji komponenty są dodatkowo wyposażone we właściwości wymienione w tzw. manifeście aplikacji (por str. 15). Właściwości komponentu zwane filtrami zamierzeń uszczegóławiają warunki jakie intencja musi spełnić, aby dany komponent zajął się obsługą przyszłej akcji. Przy czym komponent może mieć dowolną liczbę filtrów, każdy opisujący inny warunek. Intencja, która jawnie wskazuje nazwę komponentu natychmiast go aktywuje, nieistotne są dodatkowe filtry. Inaczej dzieje się w przypadku, gdy intencja nie zawiera nazwy komponentu, wówczas aktywacja komponentu odbędzie się na podstawie dopasowania do filtrów. Opis operacji zawarty w intencji to zbiór informacji, w którego skład wchodzą: opis pożądanej przyszłej akcji, zestaw danych, który będzie obejmowała akcja, rodzaj komponentu, który powinien się zająć się obsługą akcji oraz inne instrukcje. System Android identyfikuje odpowiedni komponent albo, jeżeli zachodzi taka potrzeba, tworzy nowy i przekazuje mu cały obiekt intencji. Każdy komponent przeznaczony jest do wykonywania innych zadań i ma inny cykl życia, a intencja pełni rolę spoiwa łączącego porozrzucane komponenty w spójną aplikację.

### Aktywności

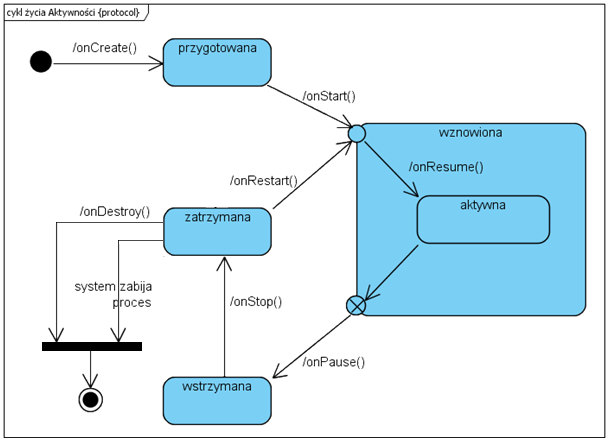
Moduł najlepiej znany użytkownikowi, to aktywność. W Androidzie aktywność jest komponentem udostępniającym pojedyncze okno przedstawiane użytkownikowi, wyposażone w narzędzia obsługi zdarzeń zachodzących pomiędzy tymże użytkownikiem a aplikacją. Zadaniem aktywności jest kontrolowanie logiki wyświetlanych treści na ekranie urządzenia. Aktywność wyświetla interfejs użytkownika złożony z widoków[[6]](#footnote-6). Zazwyczaj okno zajmuje całą przestrzeń ekranu, jednak zdarzają się okna mniejsze.

Aplikacje zazwyczaj składają się z luźno powiązanych ze sobą aktywności. Ta aktywność, która jako pierwsza tworzona jest podczas uruchomienia aplikacji, określana jest mianem głównej aktywności. Jeżeli działania użytkownika wymagają uruchomienia nowej aktywności, to jest to realizowane przez aktywną w tym czasie aktywność. Jako aktywną aktywność przyjęto określać pojedynczy ekran aplikacji mogący reagować na akcje użytkownika. Ponieważ w trakcie działania aplikacji aktywna może być tylko jedna aktywność, gdy nowa aktywność zostaje utworzona, to poprzednia przechodzi w stan wstrzymania (potem zostaje zatrzymana i zniszczona). Uruchomiona aktywność zostaje przez system operacyjny Android umieszczona na stosie, a jej okna są udostępniane użytkownikowi. Jeżeli użytkownik będzie chciał powrócić do wcześniej utworzonych aktywności, np.: naciśnie przycisk wstecz, to aktywność, z którą aktualnie pracuje, zostanie usunięta ze stosu i zniszczona, zaś poprzednia aktywność zostanie wznowiona. Najważniejsze stany, w których może znajdować się aktywność to:

* wznowiona – aktywność jest wyświetlana użytkownikowi i może reagować na akcje użytkownika,
* wstrzymana – aktywność jest wyświetlana użytkownikowi, ale nie reaguje na jego działania, np.: inna aktywność, której okno nie zajmuje całej dostępnej przestrzeni ekranu również jest wyświetlana użytkownikowi i częściowo zasłania wstrzymaną aktywność. Wstrzymana aktywność może zostać zabita przez system w celu zwolnienia zasobów pamięci operacyjnej.
* zatrzymana – dotychczasowa aktywność zostaje całkowicie zastąpiona przez inną. Nie jest już widoczna dla użytkownika i tak samo jak w stanie wstrzymania może zostać zabita przez system, jednak system Android nie musi jej powiadomić o tym, że zostanie zabita. Jeżeli system powiadomi aktywność o jej nadchodzącym zniszczeniu, to aktywność ma szansę wykonać operacje przed zniszczeniem swojego obiektu. W sytuacjach krytycznych system bez powiadomienia zabija proces przydzielony do obsługi aktywności (aktywność jest natychmiast niszczona). W obu przypadkach, jeżeli użytkownik powraca do zniszczonej aktywności, to musi ona zostać utworzona na nowo.

Zarówno w stanie wstrzymania, jak i zatrzymania aktywność jest przechowywana w pamięci wraz ze wszystkimi informacjami o elementach składowych oraz stanie w jakim się znajdują. Różnica pomiędzy w.w. stanami tkwi zaś w utrzymaniu powiązania z zarządcą okien[[7]](#footnote-7): aktywność, która przeszła w stan zatrzymania zostaje wyrejestrowana z zarządcy okien. [1]

Niektóre intencje aktywują przejścia aktywności pomiędzy różnymi stanami. Po otrzymaniu intencji aktywność może przed osiągnięciem kolejnego stanu wykonać pewne, specyficzne dla siebie zadania np.: zwolnić wszystkie używane zasoby systemowe, gdy przekazana intencja informuje o przyszłym zamknięciu aktywności.



Rysunek . Cykl życia aktywności

### Usługi

Usługa jest komponentem uruchamianym w tle na poczet wykonywania długotrwałych operacji, np.: w celu zaspokojenia określonej potrzeby aplikacji lub systemu. Usługi nie udostępniają interfejsu użytkownika. Po uruchomieniu przez dowolny komponent aplikacji usługa będzie pracowała w tle, nawet jeśli użytkownik przejdzie do innej aktywności lub aplikacji. Przykładowo, usługa może zostać stworzona i oddelegowana do pobierania danych z internetu, podczas gdy użytkownik kontynuuje pracę z aktywnością. Komponenty mogą wiązać się z usługą, aby z nią współdziałać, a także komunikować się na poziomie IPC[[8]](#footnote-8).

Domyślnie usługi uruchamiane są w głównym wątku aplikacji, więc w przypadku wykonywania intensywnych lub blokujących operacji mogą doprowadzić do ANR[[9]](#footnote-9). W celu minimalizacji tego problemu należy dopilnować utworzenia osobnego wątku na potrzeby tworzonej usługi.

Usługi mogą być wykorzystywane w obrębie jednej aplikacji do wykonywania operacji albo w celu dostarczenia jakiejś funkcjonalności innym aplikacjom. Bez względu na to czy usługi są związane czy nie (aspekt ten omówiony jest w dalszej części pracy), każdy komponent dowolnej[[10]](#footnote-10) aplikacji może z nich korzystać. Korzystanie z usług przebiega podobnie jak w przypadku aktywności: uruchamia się odpowiednio opisaną intencję, skutkującą zmianą stanu usługi lub przekazaniem jej informacji o zadaniu do wykonania.

Tak samo jak aktywności, usługi posiadają swój cykl życia (Rysunek 2), który jest znacznie mniej skomplikowany od np. cyklu życia aktywności. W przypadku usług o wiele ważniejsze jest zadbanie o właściwy sposób uruchomienia i zakończenia usługi ze względu na możliwość jej pracy w nieskończoność.

Usługi możemy podzielić na trzy typy:

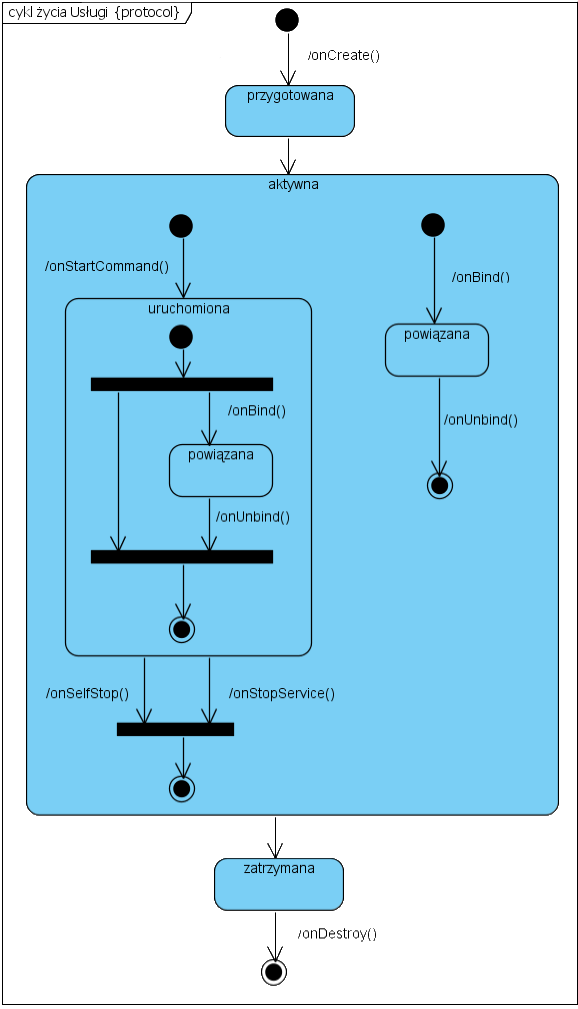
* uruchomione (niezwiązane),
* związane,
* uruchomione a następnie związane.

Usługę uznajemy za uruchomioną, gdy inny komponent aplikacji wywoła metodę startService(). Po uruchomieniu, usługa może działać w tle nieskończenie długo, nawet jeżeli komponent, który ją uruchomił został zniszczony (zakończył cykl życia). Zazwyczaj usługa zostaje powołana do wykonania indywidualnej operacji i nie zwraca wyniku swojego działania do komponentu, który ją utworzył (np.: pobieranie pliku z chmury do wskazanej lokalizacji). Komponent usługi zostanie zniszczony, gdy operacja zostanie wykonana. Niszczenie usługi może zostać rozpoczęte dwojako:

1. komponent, który ją utworzył, wysyła żądanie zatrzymania usługi (stopService()),
2. usługa sama się zatrzymuje (wywołanie metody stopSelf()).

Usługę uznajemy za związaną, gdy komponent aplikacji utworzy ją wywołując metodę bindService(). Usługa związana z komponentem pozwala klinetowi na interakcję (wysyłanie zapytań do usługi, pobieranie wyniku operacji, IPC). Usługa związana ma odmienny czas życia niż usługa, która związana nie jest. Działanie usługi związanej kończy się w momencie zerwania powiązania przez klienta. Różne komponenty mogą wiązać się z jedną usługą w tym samym czasie, ale gdy tylko ostatni z klientów zostanie odpięty od usługi, ta jest niszczona przez system. Cyklem życiowym związanej usługi nie można zarządzać, zajmuje się tym w pełni system Android.

Należy zaznaczyć, że każda usługa, bez względu na sposób jej utworzenia, może zostać powiązana z klientem (pierwszym lub kolejnym). Dopuszczalne jest powiązanie się komponentu z istniejącą usługą, która wcześniej została uruchomiona (ilustruje to diagram stanów przedstawiony na rysunku 2), w takim przypadku wywołanie stopService() lub stopSelf() nie zatrzyma usługi, dopóki wszyscy klienci nie zakończą powiązań. [3]



Rysunek . Cykl życia usługi

### Dostawcy treści

Dostawca treści odpowiedzialny jest za udostępnianie repozytorium danych, jego interfejs umożliwia spójny dostęp do danych (odczyt, zapis i ich modyfikację). Dostawcy treści opcjonalnie mogą obsługiwać IPC, jednak wtedy odpowiedzialni są również za odpowiednie zabezpieczanie dostępu do danych. Udostępniany zestaw danych może być więc zarówno współdzielony pomiędzy aplikacjami, jak i zastrzeżony tylko dla jednej z nich[[11]](#footnote-11). Dane prezentowane są zewnętrznym aplikacjom jako zestaw tabel tworzących bazę relacyjną.

Poniższe 2 akapity do przeformułowania (na razie nie mam pomysłu):

Wykorzystywanie dostawcy treści jest potrzebne i wymagane, w przypadku współdzielenia danych pomiędzy apliakacjami. Aby w kontrolowany sposób udostępniać dane tj. przy pomocy mechanizmu dostawcy treści, należy zapisać je w wewnętrznych plikach, bazie danych SQL, lub w chmurze, np.: na zdalnym serwerze. Należy również określić czy dane te są prywatne dla macierzystej aplikacji. Gdy wysyłane jest zapytanie poprzez dostawcę treści system Android sprawdza uprawnienia dostępu do danej lokalizacji i przekazuje zapytanie do dostawcy treści razem z kluczem uprawnień.

Istotną kwestią dot. przechowywania danych aplikacji jest rozróżnienie danych prywatnych i publicznych. Pliki danych przechowywane w pamięci wewnętrznej urządzenia oraz tworzone bazy danych SQL są prywatnymi danymi aplikacji oraz dostawcy treści. Zaś pliki danych zapisywane w zewnętrznej pamięci są publiczne i nie mają nałożonych ograniczeń dot. odczytu. Nie można wykorzystywać dostawcy treści, aby ograniczyć dostęp do plików przechowywanych w zewnętrznej pamięci, bo inne aplikacje mogą korzystać z innych wywołań API do odczytu i zapisu w tych pamięciach.

W tabeli 1 (możliwe przeniesienie tabeli do dodatku A) wymienione zostały rodzaje ograniczeń nakładanych na aplikacje komunikujące się z dostawcami treąci poczynając od tych o największym obszarze i najmniejszym priorytecie. W przypadku praw dostępu do dostawcy treści, im większy zasięg uprawnień, tym niższy status uprawnienia.

Tabela 1 Rodzaje ograniczeń dostępu do dostawcy treści

|  |  |  |
| --- | --- | --- |
| Poziom uprawnień | typ uprawnień | opis |
| cały content provider | Prawo do odczytu i zapisu | Uprawnienie zezwalające na modyfikacje całego c.p. |
| Prawo tylko do odczytu | Uprawnienie zezwalające na odczyt danych z c.p. |
| Prawo tylko do zapisu | Uprawnienie zezwalające na zapis danych w c.p. |
| Uri | Prawo do odczytu i zapisu | Uprawnienie zezwalające na odczyt/zapis dla określonego URI |
| Prawo tylko do odczytu | Uprawnienie zezwalające na odczyt dla określonego URI |
| Prawo tylko do zapisu | Uprawnienie zezwalające na zapis dla określonego URI |
| tymczasowe | Prawo do odczytu i zapisu | tymczasowe uprawnienie zezwalające na odczyt/zapis dla określonego URI |
| Prawo tylko do odczytu | tymczasowe uprawnienie zezwalające na odczyt dla określonego URI |
| Prawo tylko do zapisu | tymczasowe uprawnienie zezwalające na zapis dla określonego URI |

Poziom uprawnień określany jako „tymczasowe” przyznaje chwilowy dostęp do danych aplikacji, nawet jeśli ubiegająca się o dana aplikacja nie ma uprawnień, które są zazwyczaj wymagane. Prawo dostępu tymczasowego zmniejsza liczbę uprawnień wymienianych w manifeście aplikacji.

Dostęp do dostawców treści zapewniony jest pośrednio poprzez intencje. Aplikacja nie posługuje się metodami dostarczanymi z interfejsem dostawcy treści, lecz wysyła intencję, która rozpoczyna aktywność. Nowa aktywność ma za zadanie pobieranie i wyświetlanie danych w swoim UI. W zależności od rodzaju intencji, aktywność może umożliwić użytkownikowi wprowadzenie zmian w danych, które mają zostać przekazane do dostawcy treści. Wiadomość może zawierać dodatkowe dane, które aktywność również wyświetli użytkownikowi do modyfikacji. Wykorzystywanie mechanizmu intencji zapewnia integralność danych zależącą od operacji wykonywanych na przechowywanych danych (wstawiane, aktualizowane i usuwane zgodnie z ściśle określoną logiką aplikacji).

Cykl życia dostawcy treścinie jest jawnie określony w dokumentacji SDK[[12]](#footnote-12) Androida. Wspomniano jedynie o konstruowaniu komponentu dostawcy treści, ale nie napisano nic o tym kiedy jest niszczony. Wg Dianne Hackborn*[[13]](#footnote-13)* komponenty dostawców treści nigdy nie są niszczone, istnieją razem z procesami, w których zostały utworzone [5]. Innymi słowy, cykl życia dostawcy treści trwa od momentu wywołania metody onCreate() do momentu, w którym proces go zawierający zakończy się lub zostanie zabity.

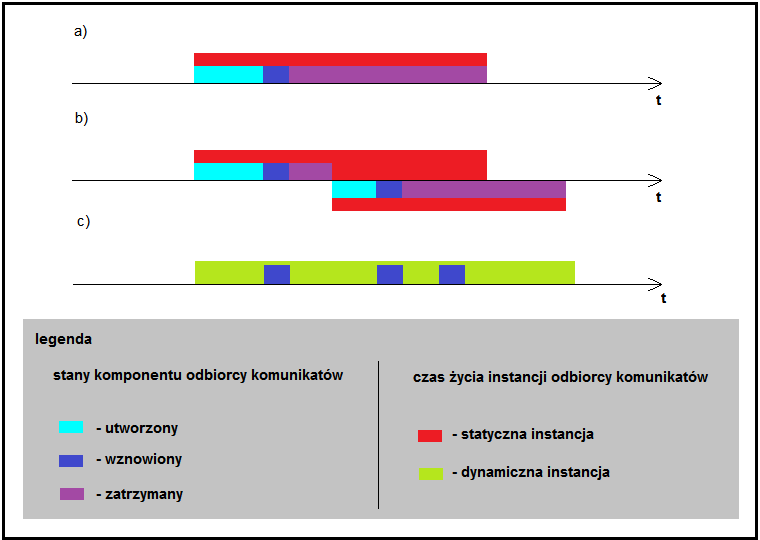
### Odbiorcy komunikatów

Odbiorca komunikatów jest komponentem, który po utworzeniu oczekuje na powiadomienia rozgłaszane przez system lub aplikacje, np.: powiadomienie o niskim poziomie energii baterii, „o nadejściu nowych wiadomości, wydarzeniach kalendarza i alarmach oraz trwających zdarzeniach, na przykład ustawieniu tabletu jako punktu dostępu Wi-Fi”.

Odbiorca komunikatów nie jest wyposażony w interfejs użytkownika. Jeżeli wystąpi zdarzenie, o którym powinien zostać poinformowany użytkownik, to odbiorca komunikatów przechwyci rozgłoszone powiadomienie i umieści je w pasku systemowym znajdującym się na ekranie urządzenia. „W zależności od ustawień i modelu tabletu możesz również usłyszeć dźwięk powiadomienia, może zacząć mrugać dioda LED itp” .

Odbiorcy komunikatów, mają nieskomplikowaną definicję cyklu życia: po utworzeniu i wykonaniu zadań umieszczonych w metodzie onReceive() zostają zniszczeni.[[14]](#footnote-14) Ze względu na krótki czas życia istnieje szereg obostrzeżeń dotyczących wykorzystania odbiorców komunikatów, jednym z nich jest wykonywanie zadań krótkotrwałych, aby nie dopuścić do ANR.

Warto zauważyć, że po wykonaniu metody onReceive() odbiorca komunikatów nie jest już aktywny, a priorytet procesu go utrzymującego wyznaczają pozostałe komponenty istniejące w tym procesie. Załóżmy, że w procesie istnieje tylko komponent odbiorcy komunikatów. W momencie wstrzymania odbiorcy komunikatów system rozważy jego proces jako pusty i agresywnie zabije ten proces, potencjalnie niszcząc zasoby udostępniane innym procesom. To oznacza, że dla operacji długotrwających lepszym rozwiązaniem jest połączenie funkcjonalności usługi i odbiorcy komunikatów, aby proces zawierający odbiorcę utrzymać aktywnym przez długi czas.



Rysunek . Porównanie czasu życia komponentu odbiorcy komunikatów i jego obiektu[[15]](#footnote-15)

Istnieją dwa rodzaje odbiorców komunikatów: statyczni (zarejestrowany w manifeście aplikacji) lub dynamiczni (niezarejestrowany w manifeście aplikacji, ale konstruoowany na bieżąco kodzie i rejestrowany za pomocą registerReceiver()). Tworzenie wystąpień statycznych obiektów odbiorcy komunikatów odbywa się, gdy zostanie odebrane rozgłoszenie, natomiast po wykonaniu metody onReceive() obiekt jest porzucany. Jeżeli nowe zgłoszenie jest odbierane, tworzony jest nowy obiekt i metoda onReceive() jest wywoływana w nowej instancji, po czym obiekt również jest porzucany. Odmiennie niż w przypadku pozostałych omówionych modułów jeden cykl życia komponentu odpowiada jednemu cyklowi życia obiektu Java[[16]](#footnote-16). Jednocześnie obiekty dynamiczne odbiorcy komunikatów nie są kontrolowane przez Android. Oznacza to, że wystąpienie dynamicznego odbiorcy komunikatów może zostać stworzone przez aplikację w dowolnym momencie przed wywołaniem metody registerReceiver() i że nie jest ono automatycznie porzucane po wykonaniu metody onReceive(). W rzeczywistości obiekty dynamiczne odbiorcy komunikatów mogą przejść kilka cykli życia komponentu (przypadek c zilustrowany na Rysunek 3). Należy pamiętać, że nawet jeśli obiekty dynamiczne odbiorcy komunikatów żyją dłużej, komponenty, które reprezentują, nadal mają bardzo krótki czas życia. Ponieważ każdy typ odbiorcy komunikatów może zostać zarejestrowany zarówno jako statyczny, jak i dynamiczny, lepiej nie powoływać się na fakt, że obiekt może lub nie może przetrwać kilka cykli życia komponentów, ale użyć najprostszego założenia (jeden komponent odbiorcy odpowiada jednemu obiektowi Java) i delegować cięższe zadania do innych komponentów aplikacji, np.: usług.

## Struktura APK[[17]](#footnote-17)

Oprogramowanie przeznaczone na platformę Android fizycznie podzielone jest na trzy odrębne sekcje:

* manifest aplikacji,
* kod wykonywalny,
* zasoby.

### manifest aplikacji

Manifest zawiera podstawowe, najważniejszy zbiór informacji o aplikacji, który jest potrzebny platformie Android, aby tą aplikację uruchomić. Autor oprogramowania musi zadbać, aby w manifeście aplikacji zostały wskazane główne komponenty składające się na aplikację oraz różne jej wymagania uruchomieniowe, np.: zestaw wymaganych uprawnień, minimalne cechy sprzętowe, minimalna wersja systemu operacyjnego, itp. . Dzięki temu system operacyjny dowiaduje się jak uruchomić aplikację i może przydzielić jej odpowiednie zasoby, a także zweryfikować wymagane uprawnienia. Jeżeli w/w informacji zabraknie w manifeście, aplikacja nie będzie oferowała pełnej funkcjonalności lub w najgorszym wypadku, system w ogóle nie pozwoli na jej uruchomienie.

Wymienione w manifeście pożądane uprawnienia ustalane są przez autora aplikacji. Zalecane jest, aby tworzona aplikacja nie ubiegała się o nadmiarowe prawa dostępu i ograniczała się tylko do tych elementów systemu, które w danym momencie są wymagane do jej poprawnej pracy. Podyktowane jest to względami bezpieczeństwa. Przykładowo, potencjalnie niebezpieczne jest API umożliwiające współdzielenie danych pomiędzy aplikacjami. W opracowywanej aplikacji nie jest ono wykorzystywane, więc nie wymaga ona praw dostępu do niego i nie należy jej go przyznawać. W naturalny sposób zabezpiecza to tworzoną aplikację przed zewnętrznymi, złośliwymi aplikacjami, które mogłyby próbować wykorzystywać nadmiarowe uprawnienia, np.: w celu dostępu do jej prywatnych danych, czy też wypaczenia jej funkcjonalności.

### kod wykonywalny

W systemie Android aplikacje są tworzone przy użyciu obiektowych języków programowania. W znakomitej większości aplikacji wykorzystywany jest język Java, jednakże istnieją biblioteki umożliwiające włączanie kodu źródłowego pisanego w innych językach (C++). Z uwagi na fakt, iż w aplikacja będąca przedmiotem dyskusji nie wymaga użycia NDK[[18]](#footnote-18) wątek ten zostanie pominięty.

Maszyna wirtualna Dalvik nie wykonuje bezpośrednio kodu źródłowego, tylko powstały z  niego w procesie kompilacji kod wykonywalny, który umieszczany jest jako pliki binarne w archiwum APK. Kod źródłowy nie jest dołączany do pakietu instalacyjnego, a umieszczony w niej kod wykonywalny jest zabezpieczony przed zapisem, co podnosi bezpieczeństwo aplikacji. Aczkolwiek zgodnie z ruchem OpenSource aplikacje dedykowane na system Android upubliczniają swoje źródła innymi kanałami.[[19]](#footnote-19)

### zasoby

Na zasoby aplikacji składają się wszystkie treści statyczne aplikacji, tj.: łańcuchy znakowe wyświetlane użytkownikowi, pliki graficzne, pliki wykorzystywane do tworzenia i te zawierające informacje o rozmieszczeniu elementów szaty graficznej aplikacji. Fizyczne pliki zebrane są w jednym miejscu, a odniesienia do nich utrzymywane są w plikach .xml o odpowiedniej strukturze tagów w odpowiednio oznaczonych katalogach. Pozwala to na łatwe i szybkie dostosowywanie aplikacji do urządzeń skonfigurowanych odmiennie bez  dodatkowego komplikowania i zaciemniania kodu źródłowego. W plikach .xml umieszczane są informacje o:

* wielojęzyczności lub jej braku,
* domyślnym trybie pracy aplikacji (z którego zestawu łańcuchów znakowych ma korzystać, jaki jest preferowany język, czy aplikacja udostępnia inne)
* preferowanym rozmiarze ekranu i wiele innych;

Istotną kwestią jest zapewnienie tworzonej aplikacji zasobów domyślnych i alternatywnych np.: osobne zestawy napisów przeznaczone dla różnych języków albo odmienne układy graficzne aplikacji zależne od orientacji ekranu urządzenia (pionowy lub poziomy).

Rozważmy aplikację zawierającą różne wersje językowe zasobów, ale nieposiadającą zasobów domyślnych. W tym przypadku każdy zestaw napisów wyświetlanych przez aplikację umieszczony jest w katalogu, którego nazwa zawiera sufiks będący kwalifikatorem konfiguracji (tutaj kodem języka). Uruchomienie aplikacji na urządzeniu pracującym w języku niewspieranym przez tą aplikację nie powiedzie się. Wynika to ze sposobu, w jaki Android dokonuje wyboru najlepiej pasujących ustawień aplikacji. W omawianym przypadku niepasujące kwalifikatory kodu języka będą kolejno odrzucane i nie pozostanie żaden spełniający wymagania, więc system operacyjny nie pozwoli na uruchomienie aplikacji.

Określenie zasobów domyślnych pozwala systemowi operacyjnemu na uruchomienie aplikacji dla konfiguracji urządzenia nieprzewidzianych w trakcie jej tworzenia. Wówczas, ponownie, każdy zestaw napisów umieszczony jest w katalogu, którego nazwa zawiera sufiks będący kodem języka. Dodatkowo wybrano jeden język jako preferowany i umieszczono jego zawartość w katalogu, którego nazwa nie zawiera żadnego sufiksu.

Uruchomienie aplikacji na urządzeniu pracującym w języku niewspieranym przez aplikację powiedzie się, ponieważ system Android nie będąc w stanie dopasować żadnego języka wybierze zestaw domyślny, preferowany przez autora aplikacji. Aplikacja nie zakończy swojego działania z powodu błędu (braku wsparcia dla danego języka), ale poprawnie się uruchomi – jedynie użytkownik może nie zrozumieć wyświetlanych treści.

Zasoby domyślne używane są również, gdy zasoby alternatywne są niedostępne lub niezdefiniowane. Zapewnienie domyślnych zasobów jest ważne nie tylko dlatego, że aplikacja może zostać uruchomiona na nieprzewidzianej konfiguracji, ale również dlatego, że nowsze wersje Androida mogą obsługiwać kwalifikatory konfiguracji, których starsze wersje nie obsługują. [9]

# Rozdział 2

## aplikacja mobilna "e-Doświadczenia w fizyce"

to do: opis aplikacji

porównanie wersji podstawowej aplikacji do nowych:

<http://developer.android.com/about/versions/android-3.0-highlights.html>

<http://developer.android.com/about/versions/android-4.0-highlights.html>

<http://developer.android.com/about/versions/jelly-bean.html>

# Rozdział 3

## Pobieranie i instalacja oprogramowania do obsługi technologii Adobe Flash

Wspomniana w rozdziale 1 izolacja na poziomie procesów i instancji maszyn wirtualnych, w połączeniu z rozbudowaną kontrolą uprawnień aplikacji tworzą hermetyczne środowisko uruchomieniowe, potocznie nazywane sandbox’em. Charakterystyczny dla tego modelu jest domyślny brak dostępu do zasobów i operacji chronionych. Z tego właśnie powodu w manifeście aplikacji musiały być wymienione wszystkie potrzebne jej do pracy uprawnienia – i tym samym system zazwyczaj automatycznie je przyznaje – jednak nie o wszystkie uprawnienia można ubiegać się w ten sposób. Operacje względnie bezpieczne, takie jak automatyczny zapis / odczyt z prywatnej pamięci, czy łączenie się z internetem itd. nie wymagają informowania o nich użytkownika i wystarczy wymienienie ich przez autora w manifeście aplikacji.

Jednak część z nich, zwłaszcza dostęp do krytycznych operacji systemowych, wymaga znacznie większego nadzoru i uzyskania zgody od właściciela urządzenia (czyli rzeczywistego użytkownika) na ich przeprowadzenie w trakcie pracy aplikacji. Zabezpieczenia wprowadzone na platformie nie pozwalają na ominięcie tej procedury, co powoduje, że użytkownik musi zostać każdorazowo poinformowany np.: o pobraniu, czy instalacji nowego oprogramowania. Z tego względu przeprowadzenie instalacji brakującego oprogramowania wymaganego przez działającą aplikację nie może odbywać się w tle. W aplikacji mobilnej "e-Doświadczenia w fizyce" użytkownik musi zostać poinformowany o wykorzystywaniu pakietu Adobe Flash Player i potrzebie jego pobrania z internetu.

Ponieważ sens istnienia aplikacji zasadza się na wykorzystaniu technologii Adobe Flash użytkownik musi zdawać sobie sprawę z instalacji oprogramowania nie pochodzącego z *AndroidMarket*. Podstawowym problemem stało się wycofanie się firmy Adobe z kontynuowania produkcji oprogramowania umożliwiającego pracę z aplikacjami flash na urządzeniach mobilnych. W udoskonalanej aplikacji „e-Doświadczenia w fizyce” wykorzystywane są archiwalne zasoby firmy Adobe udostępnione przez w/w w internecie[[20]](#footnote-20). Po zainstalowaniu dodatkowego oprogramowania na urządzeniu mobilnym aplikacja ”e-Doświadczenia wfizyce" zyskuje pełną funkcjonalność.

## Losowanie portu serwera WWW

zaimplementowane – opisać

## Mechanizm sprawdzania dostępnych aktualizacji e-doświadczeń

Aplikacje mobilne mogą zostać zaimplementowane jako jedno lub wielowątkowe. Pierwszy model wymusza wykonanie wszystkich operacji w głównym wątku aplikacji (interfejsu użytkownika). Podczas wykonywania pracy w odpowiedzi na działanie użytkownika model jednowątkowy nie będzie dawał zadowalających rezultatów, szczególnie gdy wykonywane operacje są intensywne, blokujące itp. Wszystko dzieje się w wątku interfejsu użytkownika, więc długotrwałe operacje będą blokować całe UI. Gdy wątek jest zablokowany aplikacja nie jest w stanie reagować na pojawiające się zdarzenia. Z punktu widzenia użytkownika aplikacja zawiesza się. Po dłuższym okresie bezczynności użytkownik automatycznie zostaje poinformowany przez system operacyjny o braku reaktywności danej aplikacji i może zdecydować się na przerwanie jej pracy. Przy niecierpliwym użytkowniku aplikacja nigdy nie wykona kolejnych zadań (aktualnie dopuszczalny czas „zawieszenia się” aplikacji wynosi 5 sek.).

Drugi model wymusza rozdzielenie operacji na wykonywane przez wątek główny oraz przez wątki dodatkowe. Wątek główny zajmuje się ciągłą obsługą zdarzeń związanych z działaniami użytkownika, a pozostałe wątki wykonują inne zadania np.: pobieranie danych z Internetu, wysyłanie zapytań do bazy danych itp. Przeniesienie zadań długotrwałych i blokujących do osobnego wątku działającego w tle zabezpiecza przed „zawieszeniem się” aplikacji. Wynika to z ograniczenia narzucanego na wątki niebędące głównym wątkiem aplikacji – nie mogą one modyfikować interfejsu użytkownika – nawet, jeżeli wykonanie wątku nie powiedzie się lub zostanie wstrzymane wątek główny będzie płynnie pracował. [10]

W aplikacji mobilnej "e-Doświadczenia w fizyce" sprawdzanie aktualizacji uruchamiane jest wraz ze startem aplikacji (w głównej aktywności). Zadanie to jest wykonywane przez wątek pracujący w tle, aby zapobiec przeciążeniu głównego wątku aplikacji. Obarczenie głównego wątku aplikacji sprawdzaniem czy na serwerze zamieszczono nową wersję e-doświadczenia mogłoby doprowadzić do krytycznego spowolnienia aplikacji. Jeżeli np.: aplikacja nie uzyskałaby połączenia z internetem mogłaby przestać reagować na działania użytkownika.

Wykorzystano mechanizm „AsyncTask”, który pozwala na wykonywanie asynchronicznych operacji blokujących w wątku głównym. Operacje blokujące wykonywane są automatycznie w wątku roboczym, a ich wynik udostępniany jest w wątku głównym bez potrzeby pisania dodatkowego kodu źródłowego obsługującego zdarzenia zachodzące w tym wątku.

Główna aktywność aplikacji została wyposażona w zagnieżdżoną klasę CheckForEDUpdates będące podklasą AsyncTask[[21]](#footnote-21). Ze względu na konstrukcję aplikacji zaimplementowano jedynie metodę doInBackground(), która dla każdej paczki zawierającej e-doświadczenie dokonuje na podstawie daty modyfikacji porównania pakietu znajdującego się na serwerze WWW oraz zapisanego na urządzeniu. Jeżeli pliki na serwerze zewnętrznym są nowsze, to odpowiednia informacja zostaje zapisana w „shared preferences” głównej aktywności [odnośnik do rozdziału 1, bibliografia].

...

SharedPreferences.Editor edLocalDataEditor = edSavedData.edit();

edLocalDataEditor.putBoolean(edName + *PREFS\_UPDATE\_SUFFIX*, **true**);

edLocalDataEditor.apply();

...

Ponieważ użytkownik jest informowany o wyniku pracy wątku dopiero, gdy spróbuje otworzyć ekran konkretnego doświadczenia (nie jest to główna aktywność) metoda onPostExecute() nie została zaimplementowana. onPostExecute() dostarcza rezultat pracy metody doInBackground () i pozwala na bezpieczne uaktualnienie interfejsu użytkownika.

## Dodanie zrealizowanych e-d

Zrobione – opisać

Lista dodanych eD.

### Uzupełnienie opisów oraz przykładowych ćwiczeń

Lista ed dla których wstawiono opisy.

W trakcie

### pomoc aplikacji

……

### pionowe przewijanie zdjęć w oknach popup wraz informacjami o realizatorach projektu

……

### Poprawienie grafiki aplikacji wyświetlanej podczas pracy z uruchomionym e-doświadczeniem.

……

……

……

## Zmiana wielkości wyświetlanego opisu doświadczenia

Głównym zadaniem odbiorcy komunikatów jest przechwycenie informacji o zdarzeniu i możliwie najszybszym oddelegowaniu tej informacji do przystosowanego do jego obsługi elementu aplikacji (jeżeli zachodzi potrzeba, to odbiorca komunikatów może inicjalizować usługi). Przyjrzyjmy się dokładniej dwóm głównym typom rozgłoszeń, które mogą być odbierane:

* Rozgłoszenia zwykłe (asynchroniczne). Wszyscy odbiorcy komunikatów są uruchamiani w nieokreślonej kolejności, często w tym samym czasie. Rozwiązanie to jest bardzo wydajne, ale oznacza również, że odbiorca komunikatów nie może zwracać wyniku po ukończeniu pracy ani skonsumować zgłoszenia.
* Rozgłoszenia uporządkowane dostarczane są do jednego odbiorcy komunikatów na raz. Wszyscy odbiorcy wykonują czynności po kolei, ponieważ każdy odbiorca komunikatów po zapoznaniu się z rozgłoszeniem może przekazać wynik do następnego w kolejce. Odbiorca komunikatów może również skonsumować rozgłoszenie, wówczas nie zostanie ono przekazane dalej. Kolejność odbiorców komunikatów można kontrolować poprzez ustawienie odpowiedniego filtru (android:priority attribute). Odbiorcy o tych samych priorytetach będą odpowiadali na zgłoszenie w nieustalonej kolejności, jednak wciąż jeden na raz.

Nawet podczas rozsyłania zwykłych rozgłoszeń system może w pewnych specjalnych sytuacjach dostarczać komunikaty pojedynczo do kolejnych odbiorców komunikatów. W szczególności, gdy odbiorca komunikatów może wymagać utworzenia nowego procesu (w danej chwili uruchamiany będzie tylko jeden, aby uniknąć przeciążenia systemu mnóstwem nowych procesów). Pozostają jednak utrzymane ograniczenia dot. zwykłych rozgłoszeń tj.: niedopuszczalne jest zwracanie wyniku po ukończeniu pracy ani konsumowanie zgłoszenia.

Należy zauważyć, że chociaż klasa Intencji jest używana do wysyłania i odbierania tych zgłoszeń, to mechanizm transmisji Intencji jest tutaj całkowicie oddzielony od intencji, które są używane do rozpoczęcia aktywności. Niemożliwe jest, aby odbiorca komunikatów zobaczył lub przechwycił intencję używaną do uruchomiania aktywności; podobnie, kiedy intencja jest rozgłaszana jako powiadomienie, to nigdy nie będzie zawierała opcji uruchomienia aktywności. Te dwie operacje są semantycznie bardzo różne: rozpoczynanie aktywności poprzez intencję jest operacją wykonywaną pierwszoplanowo, która reaguje, na to co użytkownik aktualnie modyfikuje, natomiast rozgłaszanie intencji jest operacją wykonywaną w tle, której użytkownik zwykle nie jest świadomy.

Istotną kwestią jest wyrejestrowywanie odbiorcy komunikatów, jeżeli został on zarejestrowany w metodzie Activity.onResume(). Odpowiednim do jego wyrejestrowania miejscem jest metoda Activity.onPause() – unika się, wtedy niepotrzebnego obciążenia systemu. Dajeto dodatkowe zabezpieczenie, ponieważ wstrzymana aktywność nie odbiera intencji. Proces, który aktualnie wykonuje odbiorcę komunikatów traktowany jest przez system jako pierwszoplanowy i będzie utrzymywany jak najdłużej (wyjątkiem są krytyczne braki pamięci).

zaimplementowane – opisać

zoom na samym layoucie tekstview.

### Analiza pojawiania się i przyczyny błędów związanych z Html.fromHtml() występujących na Androidzie 4.0 i wcześniejszych wersjach systemu.

……

### Analiza błędów zwiazane z obsługą tapnięć w komponencie webview.

……

### Analiza sposobów modyfikacji grafiki aplikacji dedykowanej dla tabletów o wysokiej rozdzielczości(powyżej 1280x800).

……

### Analiza sensowności umieszczania w aplikacji e-doświadczeń wymagających dużej mocy obliczeniowej (wahadło matematyczne, ruch ciał niebieskich).

……

# Zakończenie

Podsumowanie i prezentacja wyników

Możliwości testowania przy pomocy emulatora.

Przeprowadzone testy. Rodzaje i wyniki.

Podsumowanie wykonanych zadań i wnioski.

Dyskusja nad wycofaniem flasha z aplikacji mobilnych.

## Dodatek A

## Dodatek B

## Dodatek C

## Dodatek D

## Spis rysunków

[Rysunek 1. Cykl życia activity 8](#_Toc345843765)

[Rysunek 2. Cykl życia usługi 10](#_Toc345843766)

[Rysunek 3. Porównanie czasu życia komponentu odbiorcy komunikatów i jego obiektu 14](#_Toc345843767)

Bibliografia

|  |  |
| --- | --- |
| [1] | „http://developer.android.com/reference/android/app/Activity.html,” [Online]. |
| [2] | „http://developer.android.com/reference/android/app/Service.html,” [Online]. |
| [3] | „http://developer.android.com/reference/android/app/Service.html#ServiceLifecycle,” [Online]. |
| [4] | „http://developer.android.com/reference/android/content/ContentProvider.html,” [Online]. |
| [5] | „https://groups.google.com/forum/?fromgroups=#!topic/android-developers/PD-XxFn1hvI,” [Online]. |
| [6] | „http://devmaze.wordpress.com/2011/07/17/android-components-lifetime,” [Online]. |
| [7] | „http://support.google.com/android/bin/answer.py?hl=pl&answer=168913,” [Online]. |
| [8] | „http://developer.android.com/reference/android/content/BroadcastReceiver.html,” [Online]. |
| [9] | „http://developer.android.com/guide/components/fundamentals.html,” [Online]. |
| [10] | „http://developer.android.com/guide/components/processes-and-threads.html,” [Online]. |

1. ang. activities [↑](#footnote-ref-1)
2. ang. services [↑](#footnote-ref-2)
3. ang. content providers [↑](#footnote-ref-3)
4. ang. broadcast receivers [↑](#footnote-ref-4)
5. ang. intents [↑](#footnote-ref-5)
6. ang. views [↑](#footnote-ref-6)
7. ang. window manager [↑](#footnote-ref-7)
8. Komunikacja międzyprocesowa (ang. *Inter-Process Communication* — IPC) – umowna nazwa zbioru sposobów komunikacji pomiędzy procesami systemu operacyjnego. [↑](#footnote-ref-8)
9. Application Not Responding [↑](#footnote-ref-9)
10. “However, you can declare the service as private, in the manifest file, and block access from other applications.” [2] [↑](#footnote-ref-10)
11. “However, you can declare the service as private, in the manifest file, and block access from other applications.” [2] [↑](#footnote-ref-11)
12. ang. Software Development Kit [↑](#footnote-ref-12)
13. “a Software Engineer who sits very near the exact center of everything Android” [↑](#footnote-ref-13)
14. Na rysunku 3 przedstawiono graficzną interpretację tego faktu (przypadek a) oraz b)). Do obsługi nowego rozgłoszenia tworzony jest nowy komponent odbiorcy oraz nowy statyczny obiekt. [↑](#footnote-ref-14)
15. Wykonany na podstawie dokumentacji [↑](#footnote-ref-15)
16. [↑](#footnote-ref-16)
17. Android application package file [↑](#footnote-ref-17)
18. # [native development kit](http://stackoverflow.com/questions/6660621/what-is-native-development-kit-ndkandroid)

    [↑](#footnote-ref-18)
19. Np.: http://code.google.com [↑](#footnote-ref-19)
20. http://helpx.adobe.com/flash-player/kb/archived-flash-player-versions.html [↑](#footnote-ref-20)
21. http://developer.android.com/reference/android/os/AsyncTask.html [↑](#footnote-ref-21)