POLITECHNIKA WARSZAWSKA
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

Rok akademicki 2012/2013

PRACA DYPLOMOWA INŻYNIERSKA

Michał Aniserowicz

Platforma aplikacji klient-serwer wykorzystujących rzeczywistość rozszerzoną dla systemu Android

Praca wykonana pod kierunkiem
dra inż. Jakuba Koperwasa

Ocena: .....

.....

Podpis Przewodniczącego Komisji
Egzaminu Dyplomowego

Kierunek: Informatyka
Specjalność: Inżynieria Systemów Informatycznych
Data urodzenia: 1990.02.14
Data rozpoczęcia studiów: 2009.10.01

Życiorys

Urodziłem się 14.02.1990 w Białymstoku. Wykształcenie podstawowe odebrałem w latach 1997-2006 w Publicznej Szkole Podstawowej nr 9 w Białymstoku i Publicznym Gimnazjum nr 2 im. 42 Pułku Piechoty w Białymstoku. W latach 2006-2009 uczęszczałem do III Liceum Ogólnokształcącego im. K. K. Baczyńskiego w Białymstoku. Od roku 2009 jestem studentem studiów dziennych pierwszego stopnia na kierunku Informatyka na wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. W marcu 2012 roku podjąłem pracę jako programista w firmie Fun and Mobile, gdzie po kilku miesiącach zostałem zastępcą przywódcy zespołu, do którego należę również obecnie. Moją pasją jest programowanie aplikacji w technologii .NET Framework.

.....
Podpis studenta

Egzamin dyplomowy:
Złożył egzamin dyplomowy w dniu: .....
z wynikiem: .....
Ogólny wynik studiów: .....
Dodatkowe uwagi i wnioski Komisji: .....
.....

Streszczenie

Celem pracy jest stworzenie platformy wspomagającej tworzenie aplikacji wieloosobowych dla systemu Android, zawierającej szynę komunikacyjną klient-serwer, jak również komponenty realizujące koncepcję rzeczywistości rozszerzonej. Dodatkowym celem jest zbadanie możliwości wykorzystania w systemie Android praktyk programistycznych powszechnie stosowanych podczas pracy nad dużymi projektami. Postawione cele zostały w pełni zrealizowane - wynikiem prac jest implementacja szyny komunikacyjnej, a także komponentów umożliwiających odczytywanie współrzędnych geograficznych urządzenia oraz wyświetlanie na jego ekranie stabilizowanej grafiki trójwymiarowej. Możliwości platformy zostały zaprezentowane w przykładowej grze wieloosobowej. Praca zawiera opis działania wykonanych komponentów, przedstawia proces ich projektowania, a także wyjaśnia ich wewnętrzną strukturę.

Client-server Augmented Reality applications framework for Android system

Summary

The goal of this thesis is to create a framework supporting the development of multiuser applications for Android system. The framework should consist of a client-server bus, as well as several Augmented Reality components. An additional goal is to explore the possibility of adapting commonly used programming practises dedicated to large projects during Andorid applications development. All the goals have been fully accoplished - the output is the implementation of the client-server bus as well as AR components allowing to track the device's geographic coordinates and to render a stable three-dimensional graphics on the display of the device. In order to demonstrate the features of the framework, a sample multiplayer game has been created. The thesis includes a description of the created components' design process, as well as their functionality and structure.

Spis treści

Rozdział 1

Wstęp

Przedmiotem niniejszej pracy inżynierskiej jest platforma wspomagająca tworzenie gier wieloosobowych opartych na koncepcji rzeczywistości rozszerzonej. Praca omawia praktyczny rezultat procesu tworzenia platformy, prezentuje wgląd w architekturę kolejnych komponentów, a także zawiera rozważania na temat obecnych możliwości urządzeń klasy smarfton. Ponadto, w pracy zbadano i przedstawiono możliwości wykorzystania w aplikacjach dedykowanych systemowi Android popularnych praktyk stosowanych w dużych projektach informatycznych.

Podstawowe założenia dotyczące platformy są następujące:

1. Aplikacje klienckie stworzone przy użyciu platformy działają na urządzeniach wyposażonych w system Android.

2. Aplikacje serwerowe mogą zostać uruchomione na dowolnych urządzeniach wyposażonych w Wirtualną Maszynę Java (Java Virtual Machine, JVM) - w szczególności, serwer może działać na telefonie jednej z osób biorących udział w grze.

3. Komponenty rzeczywistości rozszerzonej są uniwersalne, tak aby mogły zostać wykorzystane w możliwie szerokim spektrum aplikacji.

4. Jak największa część przetwarzania odbywa się na serwerze. Ma to umożliwić wymianę algorytmów przetwarzających dane bez potrzeby aktualizacji aplikacji klienckich.

5. Budowa komponentów rzeczywistości rozszerzonej nie narzuca z góry architektury klient-serwer - korzystać z nich mogą także samodzielne aplikacje.

6. O ile to możliwe, kod platformy jest stworzony z zastosowaniem dobrych praktyk programistycznych.

1.1 Zakres pracy

Pracę nad platformą prowadziły równolegle dwie osoby. W wyniku tej współpracy powstały:

- szyna komunikacyjna klient-serwer,

- implementacja czterech mechanizmów rzeczywistości rozszerzonej,
- dwie przykładowe gry.

Zgodnie z uzgodnionym podziałem prac, każda z osób wykonała część szyny komunikacyjnej, dwa komponenty rzeczywistości rozszerzonej i jedną grę. Autor niniejszej pracy, oprócz części szyny komunikacyjnej, wykonał i omówi:

- komponent rzeczywistości rozszerzonej zajmujący się śledzeniem pozycji gracza za pomocą GPS i sygnałów sieci Wi-Fi,
- komponent rzeczywistości rozszerzonej umożliwiający wyświetlanie nieruchomej względem otoczenia grafiki trójwymiarowej na ekranie telefonu,
- grę bedącą przeniesieniem tradycyjnej gry w podchody w świat rzeczywistości rozszerzonej.

Ponadto, autor zbadał i opisze przystosowanie systemu Android do stosowania wzorców projektowych i architektonicznych.

1.2 Zawartość pracy

Rozdział 2 ("Podłoże pracy") krótko omawia przemiany, jakie zaszły na rynku urządzeń elektronicznych w ciągu ostatnich kilku lat, a także przybliża pojęcia takie jak Android czy rzeczywistość rozszerzona. Wyjaśnia również motywy, którymi kierowali się autorzy podczas wyboru tematyki pracy.

Rozdział 3 ("Struktura platformy") przedstawia budowę platformy oraz krótko omawia jej komponenty. Pokazuje także warianty wdrożenia przykładowej aplikacji.

Rozdziały 4-7 omawiają kolejne komponenty wchodzące w skład platformy:

- Rozdział 4 ("Realizacja szyny komunikacyjnej klient-serwer") zawiera wgląd w realizację komunikacji klient-serwer. Szczegółowo obrazuje proces przetwarzania nadawanych i odbieranych wiadomości.
- Rozdział 5 ("Odczytywanie pozycji telefonu") prezentuje komponent wykorzystujący odbiornik GPS lub analizę siły sygnałów widocznych sieci bezprzewodowych w celu odczytywania pozycji telefonu. Omawia również sposoby odczytu współrzędnych geograficznych udostępniane przez system Android.
- Rozdział 6 ("Wyświetlanie stabilizowanej grafiki trójwymiarowej na ekranie telefonu") poświęcony jest komponentowi zajmującemu się wyświetlaniem nieruchomej względem otoczenia grafiki trówymiarowej na ekranie urządzenia.
- Rozdział 7 ("Hare and Hounds - przykładowa gra oparta na platformie") przedstawia grę stworzoną w celu zaprezentowania możliwości platformy.

Rozdział 8 ("Praktyki programistyczne zastosowane w projekcie") omawia wzorce architektoniczne i projektowe oraz metodyki programistyczne zastosowane podczas pracy nad platformą. Podaje przykłady ich wykorzystania w wykonanym projekcie.

Rozdział 9 ("Podsumowanie") podsumowuje efekty pracy, a także formułuje wnioski dotyczące możliwości tworzenia aplikacji wykorzystujących sensory telefonu.

Rozdział 2

Podłoże pracy

W ciągu ostatnich kilku lat rynek urządzeń elektronicznych skierowanych do masowego odbiorcy bardzo się zmienił. Postęp technologiczny i trend miniaturyzacji pozwoliły na produkowanie niewielkich urządzeń posiadających moc obliczeniową porównywalną z możliwościami komputerów osobistych. Pojawiły się nieznane wcześniej kategorie urządzeń, takie jak smartfon czy tablet. Urządzenia te bez wątpienia zrewolucjonizowały rynek telefonów komórkowych, a także komputerów stacjonarnych i przenośnych.

2.1 Rola smartfonów w życiu codziennym

Urządzenia typu smartfon diametralnie zmieniły także - i wciąż zmieniają - sposób życia ich posiadaczy. Zastąpiły tradycyjne telefony komórkowe, odtwarzacze multimediów, nawigatory GPS, elektroniczne notatniki i wiele innych. Znalazły również zastosowanie w obszarach, w których urządzenia elekronicznie nie były wcześniej szeroko stosowane: biegacze i cykliści używają ich do planowania i śledzenia przebywanych przez siebie tras, a turyści - jako interaktywnych przewodników, wyświetlających informacje o ważnych obiektach po wskazaniu ich obiektywem kamery. Te dwa przykłady łączy jedna wspólna cecha: przytoczone w nich aplikacje "obserwują" otaczający je świat - śledzą pozycję biegacza, czy analizują otoczenie turysty. Umożliwia im to jedna z głównych cech, która odróżnia smartfony od pozostałych kategorii urządzeń codziennego użytku. Mianowicie, są one wyposażone w sensory takie jak: akcelerometr, magnetometr i odbiornik GPS.

2.2 Augmented Reality

Wykorzystanie wspomnianych sensorów w połączeniu z parametrami technicznymi telefonu takimi jak moc obliczeniowa czy pojemność pamięci RAM oraz jego niewielkimi rozmiarami, a także prostotą i wygodą użytkowania oferowaną przez ekran dotykowy, umożliwia implementację aplikacji realizujących koncepcję rzeczywistości rozszerzonej (ang. Augmented Reality, AR). Pojęcie to oznacza system, który analizuje otoczenie urządzenia i rozszerza je o dodatkowe treści, pozwalające użytkownikowi na czerpanie pełniejszych doświadczeń z otaczającej go rzeczywistości [1].

Termin "Augmented Reality" pojawił się już w latach 90-tych XX wieku. W ten sposób określono system używany przez pracowników firmy Boeing, który wyświetlał wirtualną

grafikę na obrazie świata rzeczywistego [2]. Urządzenia zdolne używać aplikacji tego typu nie były jednak wtedy szeroko dostępne - dopiero pojawienie się smartfonów umożliwiło rozpowszechnienie koncepcji AR.

Rzeczywistość rozszerzona jest koncepcją zbliżoną do rzeczywistości wirtulanej (ang. Virtual Reality, VR [3]). Wspólną cechą obu rodzajów systemów jest to, że wszelkie przetwarzanie w nich zachodzące odbywa się w czasie rzeczywistym. Tym, co odróżnia Augmented Reality od rzeczywistości wirtualnej jest źródło przetwarzanych informacji - AR bazuje na świecie rzeczywistym, podczas gdy w przypadku Virtual Reality jest to jedynie wygenerowana przez system symulacja tego świata.

Przykładem wykorzystania rzeczywistości rozszerzonej może być przytoczona już aplikacja wyświetlająca informacje o danym zabytku po wskazaniu go obiektywem aparatu telefonu. Inne obszary jej zastosowania, to między innymi [4]:

- wszelkiego rodzaju aplikacje nawigacyjne, łączące w sobie kompas, GPS, inklinometr itd.,
- narzędzia wspomagające leczenie zaburzeń i niepełnosprawności, na przykład daltonizmu,
- gry, których polem jest świat rzeczywisty.

2.3 Platforma Android

Smartfony odróżniają się od tradycyjnych telefonów przede wszystkim dotykowym interfejsem i wsparciem dla aplikacji firm trzecich. Wsparcie to objawia się w bogatym interfejsie programowania aplikacji (ang. Application Programming Interface, API) udostępnianym przez systemy operacyjne przeznaczone na smartfony i tablety. Najpopularniejsze z tych systemów to iOS firmy Apple i Android, należący do konsorcjum Open Handset Alliance[1]. Niniejsza praca skupiać się będzie na drugim z wymienionych systemów.

Android jest oparty na jądrze Linux. Cechą odróżniającą go od pozostałych systemów jest otwartość i podatność na modyfikacje. Cecha ta jest widoczna na wielu poziomach wykorzystania tej platformy:

- wielu dystrybutorów smartfonów wyposaża swoje urządzenia we własne wersje tego systemu,
- kod źródłowy Android jest publicznie dostępny[2], czego wynikiem jest powstanie wielu niekomercyjnych wersji systemu,
- aplikacje firm trzecich przed publicznym udostępnieniem nie muszą przechodzić etapu certyfikacji, który jest konieczny na takich platformach jak iOS czy Windows Phone,

[1] Open Handset Alliance - konsorcjum rozwijające otwarte standardy dla urządzeń mobilnych, w skład którego wchodzą firmy takie jak Google, HTC, Motorola i T-mobile. Strona domowa Open Handset Alliance dostępna jest pod adresem: http://www.openhandsetalliance.com/. Rozwojem platformy Android zajmuje się głównie firma Google.

[2] Źródła systemu Android dostępne są pod adresem: http://source.android.com/.

- interfejs graficzny Android posiada szerokie możliwości personalizacji w porównaniu do innych systemów mobilnych - cecha ta bezpośrednio dotyczy użytkowników końcowych.

Z punktu widzenia programisty, Android jest atrakcyjną platformą z racji możliwości tworzenia aplikacji w sposób podobny do tych przeznaczonych na komputery osobiste. Standardowym językiem programowania dla tej platformy jest szeroko znany język Java, a środowiskiem deweloperskim - popularne oprogramowanie Eclipse. Ponadto, aplikacje dedykowane systemowi Android mają dużą szansę trafienia do szerokiego grona odbiorców. O popularności platformy świadczą następujące fakty:

- urządzenia wyposażone w Android są oferowane przez wielu producentów, takich jak Samsung, Sony, czy HTC,
- dostępnych jest wiele modeli smartfonów, plasujących się w różnych przedziałach cenowych - od przeciętnej jakości urządzeń z niższej półki do luksusowych telefonów charakteryzujących się znakomitymi parametrami technicznymi,
- platforma Android w chwili obecnej posiada ponad 70% rynku urządzeń mobilnych [5],
- oficjalny sklep Android[3] oferuje ponad 600 tys. aplikacji [6].

2.4 Gry przeznaczone na platformy mobilne

Popularność smartfonów skłoniła wielu producentów oprogramowania wszelkiego rodzaju do tworzenia mobilych wersji swoich aplikacji. Należą do nich również producenci gier komputerowych - w ciągu ostatnich kilku lat powstało wiele konwersji popularnych gier, dostosowanych do specyfiki interfejsu pozbawionego fizycznej klawiatury. Przykładem mogą być wyścigi samochodowe, w których gracz steruje pojazdem nie za pomocą przycisków, a przechylając telefon lub tablet w odpowiednią stronę.

Platformy mobilne udostępniają jednak możliwości znacznie większe niż tworzenie mobilnych odpowiedników tradycyjnych gier komputerowych. Po pierwsze, smartfon nie musi być stale podłączony do zewnętrzenego źródła zasilania - gracz nie musi więc przez cały czas gry znajdować się w jednym pomieszczeniu. Po drugie, gry tworzone na smartfony mają możliwość wykorzystania rzeczywistości rozszerzonej. Te dwa fakty stwarzają nieznany wcześniej potencjał tworzenia gier, których polem jest świat rzeczywisty, a nie wirtulany.

2.5 Uzasadnienie wyboru tematu pracy

O wyborze tematu niniejszej pracy inżynierskiej zdecydowały wszystkie wyżej przedstawione fakty. Autorzy pragną poznać specyfikę programowania nowoczesnych aplikacji mobilnych, a także spróbować własnych sił na polu rzeczywistości rozszerzonej. Augmented

[3]Google play: https://play.google.com/store

Reality jest obszarem jeszcze niezbadanym, jednakże w bardzo szybkim tempie zyskującym popularność - wydaje się, że jest to najlepszy moment na wzięcie udziału w procesie rozwoju i popularyzacji tej idei. Dodatkową motywacją jest fakt, że na chwilę obecną wybór platform wspomagających wykorzystanie AR jest bardzo ograniczony.

Wybór architektury klient-serwer jako domyślnego przeznaczenia platformy jest podyktowany chęcią zbadania potencjału wykorzystania obliczeń rozproszonych w aplikacjach mobilnych. Dodatkowo, architektura ta pozwala na tworzenie aplikacji i gier wieloosobwych, co rozszerza pole zastosowania stworzonego narzędzia.

Należy również wspomnieć o czysto programistycznym aspekcie pracy. Obaj autorzy traktują praktyki programistyczne nie tylko jako przedmiot edukacji, ale i zainteresowań. Z tego względu, podjęcie się pracy nad rozbudowaną platformą w dwuosobowym zespole jest nie tylko sprawdzianem umiejętności nabytych w toku studiów, ale i okazją do poszerzenia wiedzy i zyskania cennego doświadczenia w tej dziedzinie. Aby z tej okazji skorzystać, autorzy postanowili zbadać możliwość zastosowania w platformie Android popularnych praktyk programistycznych stosowanych podczas tworzenia aplikacji przeznaczonych na komputery osobiste.

Rozdział 3

Struktura platformy

W ramach niniejszej pracy inżynierskiej powstał *fARmework* - platforma (ang. framework) wspierająca tworzenie wieloosobowych aplikacji z wykorzystaniem rzeczywistości rozszerzonej. W jej skład wchodzą biblioteki realizujące połączenie klientów z serwerem, obsługujące sensory telefonu i ułatwiające tworzenie aplikacji klienckich, a także dwa praktyczne przykłady wykorzystania: gra w podchody i "kamień, papier, nożyce".

Pełna lista komponentów wchodzących w skład platformy przedstawia się następująco[1]:

1. *Core* - trzon platformy odpowiadający za komunikację klienta z serwerem. Umożliwia przesyłanie danych od klienta do serwera i vice-versa, a także sprawdza zgodność komunikatów przesyłanych przez nadawcę z komunikatami oczekiwanymi przez odbiorcę.

2. *Utils* - zawiera komponenty ułatwiające tworzenie aplikacji klienckich:

   - wspiera użycie podstawowych funkcji telefonu, takich jak odtwarzanie dźwięku i odczytywanie obrazu z kamery,
   - dostarcza klas bazowych ułatwiających implementację wzorca architektonicznego Model-View-ViewModel,
   - ułatwia zarządzanie widokami,
   - upraszcza odczytywanie i zapisywanie ustawień aplikacji.

3. *PositionTracking* - komponent rzeczywistości rozszerzonej odpowiedzialny za śledzenie pozycji użytkownika telefonu:

   - odczytuje współrzędne geograficzne urządzenia używając metod oferowanych przez system Android,
   - oblicza dystans i azymut pomiędzy dwoma punktami na kuli ziemskiej,
   - zawiera rozszerzenie *PositionTracking.WiFi* implementujące metodę odczytywania pozycji telefonu na podstawie siły sygnałów pobliskich sieci Wi-Fi.

[1]Komponenty przedstawione w punktach 5. i 7. nie zostały wykonane przez autora i nie zostaną omówione.

4. *SpaceGraphics* - komponent rzeczywistości rozszerzonej odpowiedzialny za renderowanie trójwymiarowego obiektu nieruchomego względem otoczenia telefonu:
   - wyświetla trójwymiarowy model na ekranie telefonu,
   - korzystając z odczytów akcelerometru i magnetometru obraca model tak, aby pozostawał on nieruchomy względem otoczenia telefonu niezależnie od jego położenia,
   - umożliwia dynamiczną zmianę koloru modelu,
   - zawiera aplikację pozwalającą symulować działanie komponentu na emulatorze systemu Android.
5. *ScreenGestures*, *SpaceGestures* - komponenty rzeczywistości rozszerzonej odpowiedzialne za odczytywanie i interpretowanie gestów rysowanych przez użytkownika na ekranie telefonu lub w powietrzu:
   - zawierają kontrolki interfejsu użytkownika umożliwiające wprowadzenie (narysowanie lub nagranie) gestu,
   - rozpoznają gest porównując go z osadzonymi w aplikacji szablonami (np. kwadrat, trójkąt),
   - umożliwiają definiowanie własnych szablonów.
6. *Hare and Hounds* - gra będąca przeniesieniem tradycyjnej gry w podchody w świat rzeczywistości rozszerzonej. Korzysta z komponentów *PositionTracking* i *SpaceGraphics*.
7. *Rock Paper Scissors* - gra będąca przeniesieniem tradycyjnej gry "kamień-papier-nożyce" w świat rzeczywistości rozszerzonej. Korzysta z komponentów *ScreenGestures* i *SpaceGestures*.

3.1 Wewnętrzna organizacja komponentów

Każdy komponent podzielony został na trzy moduły:

1. Kod specyficzny dla platformy Android, umieszczony w module o nazwie:
   - *Client* - dla komponentów *Core*, *Utils* i przykładowych gier,
   - *Android* - dla komponentów rzeczywistości rozszerzonej.
2. Kod w podstawowym środowisku Java, który może być wykorzystany na dowolnej platformie, umieszczony w module:
   - *Server* - dla komponentów *Core*, *Utils* i przykładowych gier,
   - *Java* - dla komponentów rzeczywistości rozszerzonej.
3. Kod wspólny dla obu powyższych części, umożliwiający ich współpracę - tj. zawierający klasy komunikatów przesyłanych pomiędzy częścią kliencką a serwerową - umieszczony w pakiecie *Data*.

Rozróżnienie pomiędzy *Client* i *Android* wynika stąd, że moduły *Client* stworzone zostały z myślą o architekturze klient-serwer. Moduły *Android* (komponenty rzeczywistości rozszerzonej) nie mają z góry narzuconej architektury - mogą zostać wykorzystane w samodzielnych aplikacjach.

Analogicznie, moduły *Server* dedykowane są stronie serwerowej aplikacji. Natomiast moduły *Java* są uniwersalne - korzystać z nich może zarówno serwer rezydujący na zewnętrznej maszynie, jak i strona kliencka lub samodzielna aplikacja przeznaczona na platformę Android.

Wyjątkiem od tej reguły jest komponent *SpaceGraphics*, który jest przeznaczony jedynie na system Android. Z tego względu zawiera tylko jeden moduł - *SpaceGraphics.Android*.

3.2 Warianty wdrożenia aplikacji

Jednym z założeń przyjętych podczas tworzenia platformy jest brak wymogów co do architektury aplikacji korzystających z komponentów rzeczywistości rozszerzonej. Komponenty te mogą być wykorzystane zarówno w aplikacjach o architekturze klient-serwer, jak i tzw. standalone (aplikacje samodzielne). Komponent *Core*, realizujący połączenie pomiędzy klientem a serwerem, z oczywistych względów może być wykorzystany jedynie w aplikacjach dwuwarstwowych.

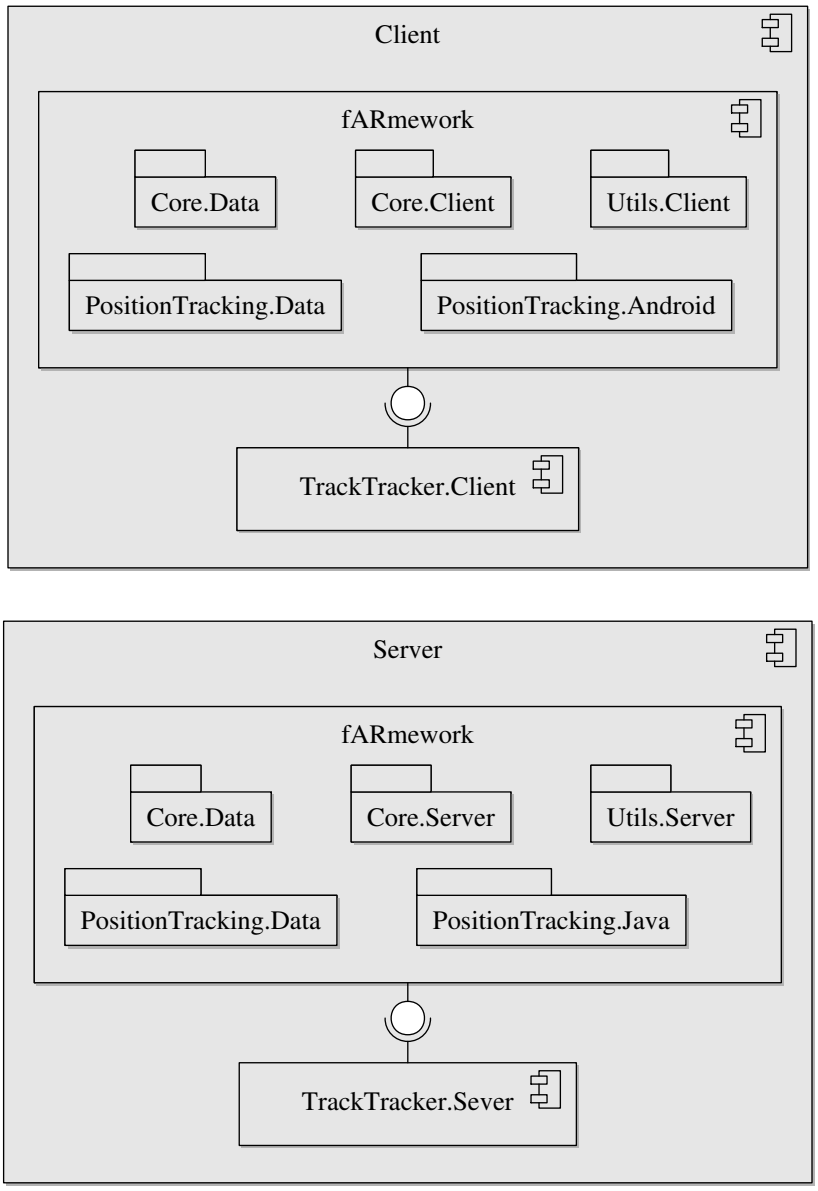
Oba warianty - klient-serwer oraz standalone - zostaną przedstawione na przykładzie hipotetycznej aplikacji, która pozwala użytkownikowi zapamiętać przebytą przez niego ścieżkę, a następnie wrócić po tej ścieżce do punktu startowego. W celu realizacji operacji związanych z pozycją urządzenia, aplikacja korzysta z komponentu *PositionTracking*. Po zapamiętaniu ścieżki i rozpoczęciu powrotu, użytkownik jest informowany o kierunku w jakim powinien się poruszać aby dotrzeć do kolejnego punktu ścieżki.

3.2.1 Architektura klient-serwer

Jak podaje temat niniejszej pracy, docelową architekturą dla platformy fARmework jest architektura klient-serwer, gdzie niektóre komponenty znajdują się tylko na serwerze, a inne - tylko w aplikacji klienckiej. W tej wersji:

- odczyt pozycji użytkownika odbywał się będzie po stronie klienta,
- serwer będzie zapamiętywał pozycje i obliczał właściwy kierunek poruszania,
- wymagane będzie stałe połączenie internetowe pomiędzy klientem a serwerem,
- strona kliencka korzystać będzie z modułów o przyrostkach *Client* i *Android*, a strona serwerowa - *Server* i *Java*.

Diagram komponentów warstwy klienckiej i serwerowej tego wariantu przedstawia Rysunek 3.1.

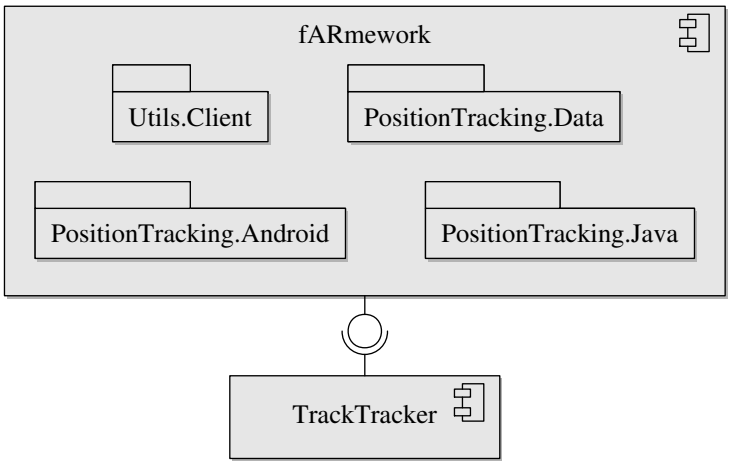


Rysunek 3.1: Diagram komponentów aplikacji klient-serwer korzystającej z *fARmework*.

3.2.2 Architektura standalone

W przypadku aplikacji standalone wszystkie operacje będą realizowane przez urządzenie użytkownika, a połączenie internetowe nie będzie wymagane. Wykorzystane zostaną zatem wszystkie moduły komponentu *PositionTracking*, a komponent *Core* nie znajdzie zastosowania.

Diagram komponentów samodzielnej aplikacji zawiera Rysunek 3.2.



Rysunek 3.2: Diagram komponentów samodzielnej aplikacji korzystającej z *fARmework*.

Rozdział 4

Realizacja szyny komunikacyjnej klient-serwer

Jak wyjaśniono w sekcji 3.2, platforma *fARmework* została przygotowana zarówno dla samodzielnych aplikacji przeznaczonych na system Android, jak również dla aplikacji klient-serwer. Niniejszy rozdział poświęcony jest komponentowi *Core*, który umożliwia tworzenie aplikacji drugiego z wymienionych typów.

Oparta na platformie aplikacja kliencka może w łatwy sposób wymieniać wiadomości z serwerem, o ile obie strony znają klasy przesyłanych wiadomości. Komponent *Core* realizuje zarówno wysyłanie i odbieranie danych, jak i walidację poprawności ich klas. Za zestawienie połączenia pomiędzy klientem a serwerem odpowiada zewnętrzna biblioteka netty [7]. Komponent *Core* przesyła za jej pomocą dane zserializowane do formatu JSON. Serializację i deserializację realizuje zewnętrzna biblioteka, google-gson [8].

4.1 Obiekty realizujące komunikację klient-serwer

Z punktu widzenia potencjalnego programisty korzystającego z *fARmework*, wymiana wiadomosci pomiędzy klientem a serwerem jest bardzo wygodna. Programista jedynie zleca nadawcy przesłanie dowolnych danych i dba o to, żeby odbiorca mogł je obsłużyć - proces serializacji, przesłania i walidacji obiektu danych odbywa się automatycznie.

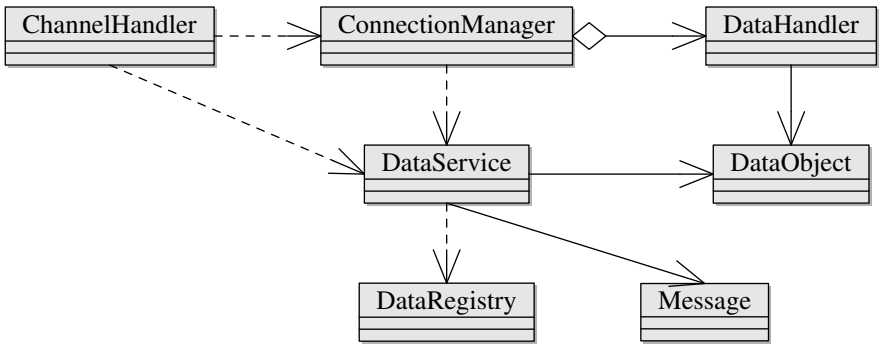
Aby umożliwić taki stopień automatyzacji przetwarzania, komponent *Core* zawiera zestaw klas realizujących proces komunikacji. Obiekty biorące udział w tym procesie, to:

- *ConnectionManager* - jest to główny obiekt zarządzający połączeniem. Realizuje nastepujące zadania:
  - otwiera i zamyka połączenie,
  - tworzy obiekty *ChannelHandler*,
  - rejestruje i przechowuje obiekty *DataHandler*.
- *ChannelHandler* - obsługuje zdarzenia występujące podczas trwania połączenia, takie jak:
  - zestawienie połączenia,

  - zakończenie połączenia,
  - nadejście wiadomości,
  - wystąpienie sytuacji wyjątkowej.

- Obiekty danych - obiekty przechowujące dane wymieniane pomiędzy klientem a serwerem.
- *Message* - reprezentuje pojedynczą wiadomość. Zawiera:
  - zserializowany obiekt danych (pole *data*),
  - nazwę typu tego obiektu (pole *type*).
- *DataRegistry* - przechowuje typy obiektów danych przesyłanych pomiędzy klientem a serwerem i nazwy tych typów. Umożliwia stwierdzenie, czy dany typ danych jest znany odbiorcy wiadomości.
- *DataService* - korzystając z obiektu *DataRegistry* i biblioteki gson, realizuje serializację i deserializację obiektów danych.
- *DataHandler* - obiekt implementujący procedurę obsługi danych konkretnego typu.

Organizację opisanych obiektów pokazuje Rysunek 4.1.



Rysunek 4.1: Organizacja obiektów biorących udział w procesie komunikacji.

Obiekty danych i obiekty *DataHandler* są dostarczane przez aplikacje korzystające z *fARmework*, a ich struktura i implementacja mogą być dowolne.

4.2 Format serializacji danych

Wybór formatu serializacji danych podyktowany był następującymi wymaganiami:

1. Format jest obsługiwany przez wiele platform. Ma to umożliwić stworzenie alternatywnych wersji *fARmework* dedykowanych innym systemom, ale kompatybilnych ze sobą.

2. Format obsługuje obiekty o rozbudowanej strukturze, takie jak obiekty zagnieżdżone i kolekcje obiektów.

3. Stosowanie formatu nie wymusza sposobu implementacji serializowanych obiektów, w szczególności nie wymaga stosowania adnotacji (ang. annotations). Ma to umożliwić transparentną dla systemu wymianę formatu.

4. Zserializowany obiekt zajmuje możliwie mało miejsca w pamięci. Ma to ograniczyć do minimum ilość przesyłanych danych.

Decyzję podjęto po rozpatrzeniu następujących popularnych sposobów serializacji:

1. Serializacja binarna zaimplementowana w platformie Java [9]. Umożliwia ona bardzo wydajne serializowanie obiektów o dowolnej strukturze, jednak nie spełnia pozostałych wymagań:

   - jest używana tylko w obrębie środowiska Java (wymaganie 1.),
   - wymaga, aby serializowane obiekty implementowały interfejs Serializable (wymaganie 3.).

2. Format XML [10]. Jest to uniwersalny format zapisu danych wspierany przez wiele platform. Nie nakłada wymagań co do budowy serializowanych obiektów, ani ich implementacji. Jego wadą jest jednak nieoptymalny rozmiar zserializowanego obiektu.

3. Format JSON. Posiada wszystkie zalety języka XML przy jednoczesnym zapewnieniu bardziej wydajnej serializacji.

Ostatecznie wybrany został format JSON, ponieważ spełnia wszystkie wymagania w zadowalającym stopniu.

4.2.1 Format JSON

JSON (JavaScript Object Notation) jest formatem reprezentacji obiektów łatwym do zrozumienia przez człowieka, jak również do przetworzenia przez komputer [11]. Przykładowy obiekt Message zserializowany do tego formatu ma następującą postać:

```
{
  "type": "com.fARmework.HareAndHounds.Data.GameStartInfo",
  "data": "{ "DemandedPositionUpdateInterval": 20 }"
}
```

Zaletą tego formatu jest całkowite uniezależnienie od platformy. Jako że dostępne są parsery JSON dla wielu popularnych języków programowania - w tym używanych podczas tworzenia aplikacji mobilnych (takich jak C# czy Objective-C) - możliwe jest przeniesienie klienckiej warstwy fARmework na inne platformy (np. Windows Phone, iOS) przy zachowaniu jednej implementacji warstwy serwerowej.

Biblioteka google-gson wybrana została z tego względu, że nie narzuca żadnych wymagań i ograniczeń co do obiektów, które serializuje i deserializuje. Oznacza to, że serializowane obiekty nie muszą mieć określonej klasy bazowej, a pola ich klas nie muszą być oznaczone adnotacjami określającymi sposób serializacji - w takim przypadku mówi się, że biblioteka współpracuje z obiektami POJO (Plain Old Java Object).

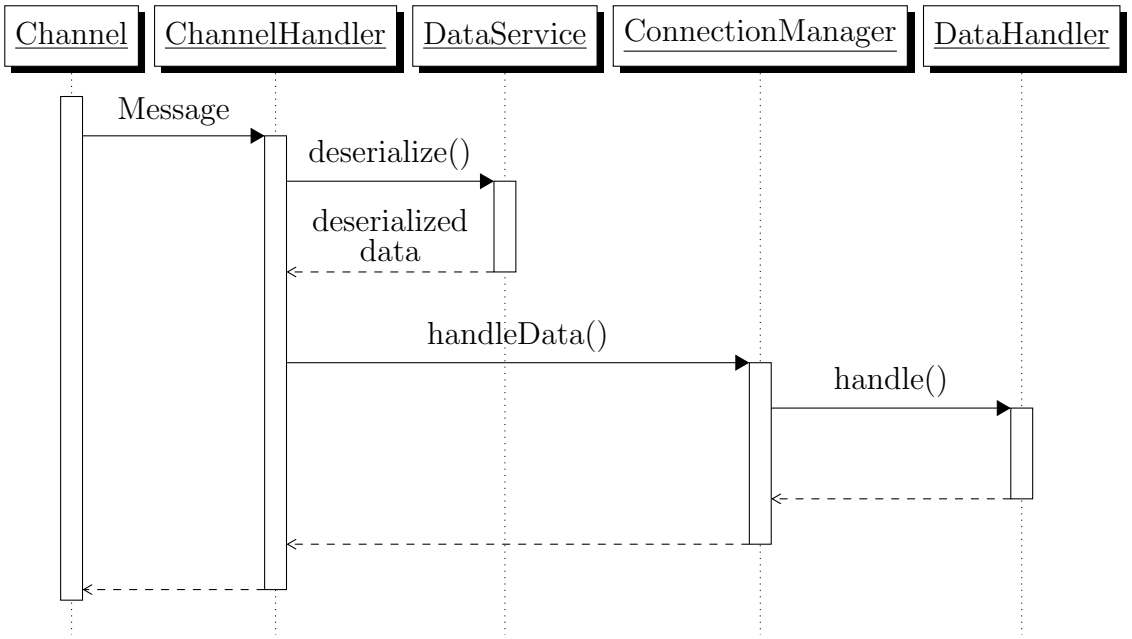
Sposób serializacji danych jest znany jedynie klasie DataService. Format serializacji można więc wymienić w całkowicie transparentny dla systemu sposób, dostarczając alternatywnej implementacji tej klasy.

4.3 Proces przetwarzania wiadomości

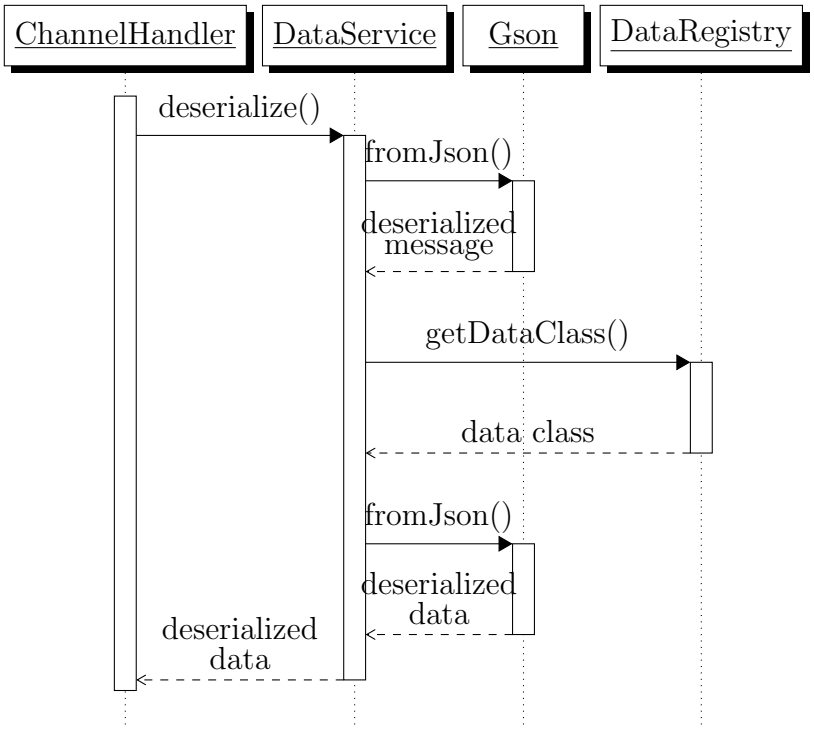
Opisane wcześniej obiekty współpracują ze sobą zarówno podczas odbierania, jak i nadawania danych. Przetwarzanie odebranej wiadomości przebiega następująco:

1. Następuje zdarzenie *MessageReceived* - obiekt *ChannelHandler* otrzymuje ciąg znaków reprezentujący zserializowaną wiadomość.
2. *ChannelHandler* zleca obiektowi *DataService* deserializację wiadomości do obiektu danych.
3. *DataService* korzysta z biblioteki gson w celu deserializacji otrzymanego ciągu znaków do obiektu *Message*.
    - Jeśli deserializacja się nie powiedzie, wiadomość jest ignorowana.
4. Po otrzymaniu obiektu *Message*, *DataService* próbuje pobrać klasę (obiekt *Class*) zserializowanego obiektu danych zawartego w wiadomości - w tym celu przekazuje typ danych (pole *type* obiektu *Message*) rejestrowi danych (*DataRegistry*).
    - Jeśli typ nie jest zarejestrowany, wiadomość jest ignorowana.
5. *DataService* ponownie korzysta z gson, tym razem próbując zdeserializować dane (pole *data* obiektu *Message*) do obiektu klasy otrzymanej z *DataRegistry*.
    - Niepowodzenie deserializacji skutkuje zignorowaniem wiadomości.
    - Pomyślnie zdeserializowany obiekt danych jest zwracany obiektowi *ChannelHandler*.
6. *ChannelHandler* przekazuje otrzymany obiekt danych obiektowi *ConnectionManager* do dalszego obsłużenia.
7. *ConnectionManager* sprawdza, czy dla typu reprezontawego przez obiekt danych zarejestrowana jest procedura obsługi (*DataHandler*). Następnie dane przekazywane są odpowiedniej procedurze.
    - Jeśli nie zostanie znaleziony *DataHandler* obsługujący dany typ, to wiadomość jest ignorowana.

Uproszczony przebieg powyższego procesu przedstawia Rysunek 4.2. Deserializacja danych została wyszczególniona na Rysunku 4.3.



Rysunek 4.2: Przebieg procesu przetwarzania odebranej wiadomości.

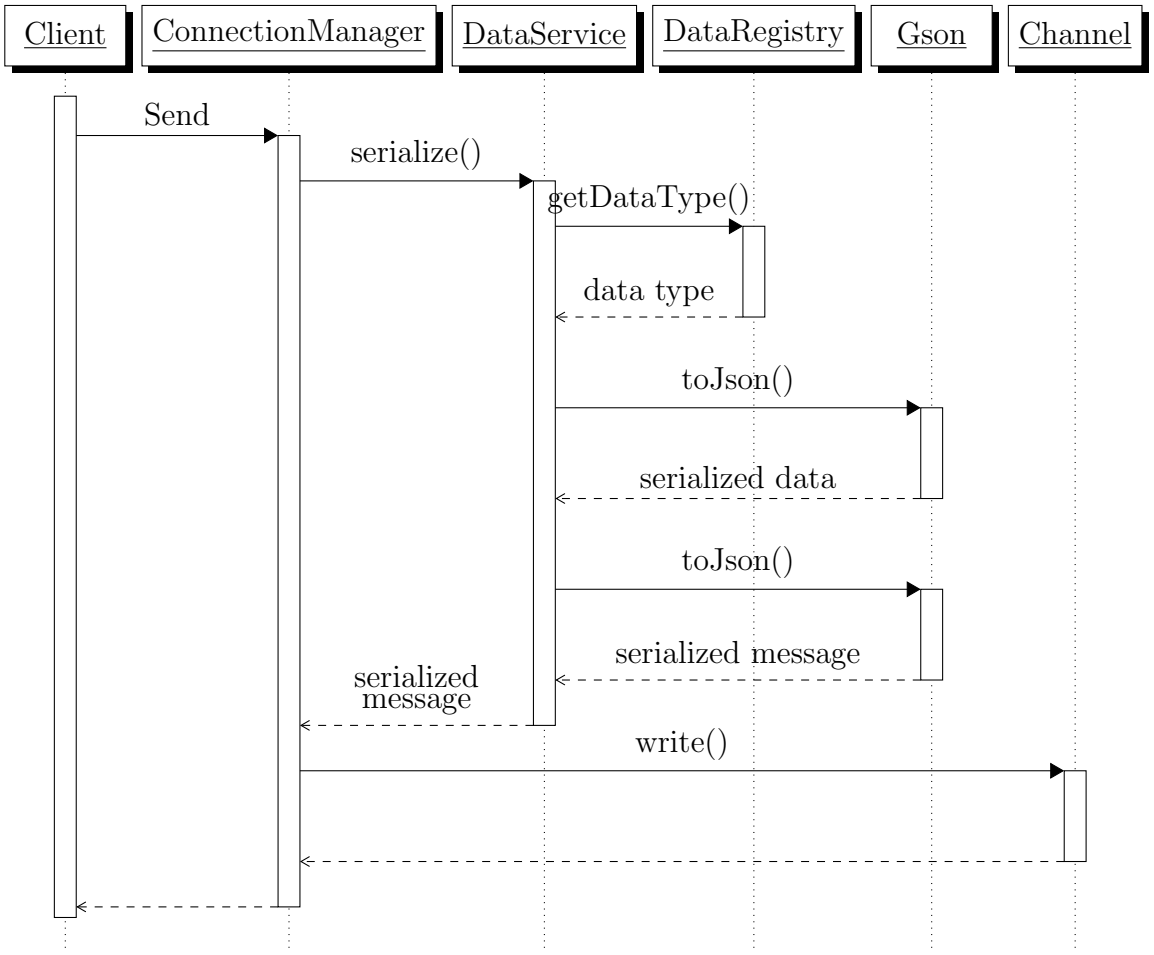


Rysunek 4.3: Przebieg procesu deserializacji odebranej wiadomości.

Przetwarzanie danych wysyłanych jest lustrzanym odbiciem procesu odbierania danych. Różnice są następujące:

- Jako że wysłanie danych odbywa się na żądanie aplikacji a nie w wyniku zdarzenia, *ChannelHandler* nie bierze w nim udziału.
- Dane są serializowane, a nie deserializowane:
  1. Na podstawie klasy obiektu danych, pobierany jest jego typ.
  2. Obiekt danych jest serializowany.
  3. Tworzony jest obiekt *Message* zawierający typ i zserializowane dane.
  4. Obiekt *Message* podlega serializacji.
- *ConnectionManager* manager zleca bibliotece netty wysłanie wiadomości - obiekty *DataHandler* nie są potrzebne.

Przebieg procesu wysyłania został zobrazowany na Rysunku 4.4.



Rysunek 4.4: Przebieg procesu przetwarzania wysyłanych danych.

4.4 Rejestracja typów obiektów danych i procedur ich obsługi

Zadaniem użytkownika *fARmework* w kontekście komunikacji klient-serwer jest właściwe rejestrowanie typów przesyłanych danych i procedur obsługi tych danych. O ile w przypadku typów danych wystarczy pojedyncza rejestracja każdego typu, o tyle rejestracja procedur obsługi może być realizowana na kilka sposobów, a jedne procedury mogą zastępować inne.

4.4.1 Rejestracja obiektów danych

Rejestracja typów wszystkich obiektów danych przesyłanych od klienta do serwera i viceversa, powinna odbyć się zarówno po stronie klienta, jak i serwera (jeśli którakolwiek ze stron nie zna danego typu, obiekty tego typu nie zostaną przetworzone). Typowo, rejestracja ta jest dokonywana podczas inicjalizowania aplikacji, a typy danych pozostają zarejestowane przez cały czas jej działania.

Aby ułatwić i zautomatyzować rejestrację, wszystkie komponenty rzeczywistości rozszerzonej wchodzące w skład *fARmework* zawierają obiekt *DataRegistrar*. *DataRegistrar* implementuje operację przyjmującą obiekt *DataRegistry* i rejestrującą w nim wszystkie typy obiektów danych występujących w danym komponencie.

4.4.2 Rejestracja procedur obsługi danych

W przeciwieństwie do typów danych, rejestracja procedur obsługi odbywa się niezależnie w każdej z warstw - serwer nie musi rejestrować procedur obsługi danych, których jest nadawcą. Kolejną różnicą, znacząco wpływającą na implementację, jest potrzeba umożliwienia wymiany procedur w trakcie działania aplikacji. W celu spełnienia tego wymagania, mechanizm rejestracji stworzono w oparciu o wzorzec projektowy Strategia [12], który pozwala na dynamiczną zmianę zachowania programu w trakcie jego działania. Rozwiązanie to zostanie szczegółowo opisane w Rozdziale 8.

Jednocześnie zarejestrowanych może być wiele procedur obsługi dla różnych typów danych. Procedurę dla danego typu można zarejestrować na dwa sposoby:

- tak, aby obsługiwała dane nadawane przez konkretnego klienta,
- tak, aby obsługiwała dane pochodzące od wszystkich klientów.

Procedura obsługująca wiadomości konkretnego klienta ma pierwszeństwo przed procedurą obsługującą wiadomości od wszystkich klientów. Oznacza to, że jeśli dla danego typu danych zarejestrowana jest procedura pierwszego rodzaju dla danego klienta i, oprócz niej, procedura drugiego rodzaju, to jeśli ten klient przyśle dane tego typu, to użyta zostanie tylko zarejestrowana dla niego procedura pierwszego rodzaju.

Procedury należy wyrejestrować możliwie szybko po nastąpieniu zdarzenia, które kończy ich przydatność. W przypadku gry wieloosobowej przykładowym czasem życia procedury po stronie serwera może być pojedyncza rozgrywka, a w przypadku klienta - pojedyncza aktywność lub etap rozgrywki.

Rozdział 5

Odczytywanie pozycji telefonu

Głównym celem powstania *fARmework* jest stworzenie uniwersalnych komponentów rzeczywistości rozszerzonej. Pierwszym z komponentów powstałych na potrzeby niniejszej pracy inżynierskiej jest *PositionTracking* - zbiór trzech modułów ułatwiających wykorzystanie wbudowanego w telefon odbiornika GPS (a także innych metod odczytywania współrzędnych geograficznych urządzenia):

- *PositionTracking.Android* - zawiera klasy zajmujące się odczytywaniem pozycji telefonu na kuli ziemskiej,
- *PositionTracking.Java* - dostarcza klas realizujących podstawowe operacje z użyciem współrzędnych geograficznych,
- *PositionTracking.Data* - zawiera używaną w obu powyższych modułach klasę danych reprezentującą parę współrzędnych geograficznych.

W wyniku dalszych prac opracowane zostało rozszerzenie tego *PositionTracking* - komponent *PositionTracking.WiFi*. Implementuje on algorytm wyznaczania lokalizacji telefonu na podstawie siły sygnałów sieci bezprzewodowych znajdujących się w jego pobliżu. Rozszerzenie składa się z następujących modułów:

- *PositionTracking.WiFi.Android* - zawiera klasy realizujące odczyt siły sygnałów sieci Wi-Fi widocznych dla telefonu,
- *PositionTracking.WiFi.Java* - implementuje algorytm wyznaczania położenia telefonu na podstawie odczytów,
- *PositionTracking.WiFi.Data* - wprowadza klasę danych reprezentującą odczyty siły sygnałów sieci.

5.1 Metody odczytywania pozycji telefonu oferowane przez system Android

Komponent *PositionTracking* korzysta z metod odczytywania pozycji telefonu na kuli ziemskiej zaimplementowanych w API platformy Android. System udostępnia możliwość odczytywania pozycji z następujących źródeł [13]:

5.1.1 GPS

Odczyt bezpośrednio z odbiornika GPS zamontowanego w urządzeniu. Jest to najdokładniejszy sposób określania pozycji telefonu - jego dokładność wynosi od kilku do kilkunastu metrów. Ma jednak znaczące ograniczenia:

- ze względu na konstrukcję systemu GPS, do poprawnego działania wymaga otwartej przestrzeni - używanie go w budynkach lub w ich pobliżu skutkuje niedokładnymi odczytami,
- przed uzyskaniem pierwszego odczytu musi nawiązać kontakt z kilkoma (ok. 7) satelitami GPS, co może potrwać nawet do kilku minut,
- pobiera najwięcej energii spośród wszystkich źródeł.

5.1.2 Assisted GPS

Metoda Assisted GPS (AGPS, wspomagany GPS) oprócz odbiornika GPS wykorzystuje sygnał sieci komórkowej. Telefon otrzymuje od pobliskich stacji bazowych informacje na temat dostępności satelitów GPS oraz przybliżonych pozycji tych satelitów. Znacząco przyśpiesza to uzyskanie pierwszego odczytu, a także eliminuje wymóg otwartej przestrzeni. Co więcej, AGPS pobiera mniej energii niż bezpośredni odczyt GPS. Jego wadą jest jednak dokładność - posiłkowanie się przybliżonymi danymi pochodzącymi ze stacji bazowych powoduje jej spadek do kilkudziesięciu metrów.

5.1.3 Odczyt pasywny

Metoda odczytu pasywnego nie korzysta z odbiornika GPS - jest więc najmniej dokładna (jej dokładność wynosi od kilkudziesięciu do kilkuset metrów). Polega na porównaniu siły sygnału odbieranego przez urządzenie od pobliskich stacji bazowych. Znając lokalizajcję stacji bazowych możliwe jest przybliżone określenie pozycji urządzenia. Dodatkowym źródłem informacji jest siła sygnału pochodzącego od sieci Wi-Fi wykrywanych przez urządzenie[1].

Wyboru źródła odczytu dokonuje programista, poprzez jawne wskazanie, którego dostawcy lokalizacji (ang. *LocationProvider*) chce użyć. Może również zdać się na system, formułując jedynie kryteria wyboru, takie jak wymagana dokładność odczytu, możliwość uzyskania informacji o prędkości poruszania się urządzenia i kierunku jego ruchu, czy zezwolenie na użycie metody, która wiąże się z naliczeniem opłat przez operatora sieci. PositionTracking używa domyślnego sposobu odczytu, podając domyślne kryteria wyboru. Pozwala to systemowi wybrać najdokładniejszy w danych warunkach bezpłatny *LocationProvider*.

Użytkownik komponentu może zlecić pojedynczy odczyt lub odczyt cykliczny z zadanym interwałem czasowym. W przypadku wyboru drugiej możliwości, powinien pamiętać o jawnym zakończeniu odczytu, gdy przestanie go potrzebować.

[1]Firma Google, podobnie jak inne firmy działające w sektorze mobilnym, zbiera i przechowuje informacje o lokalizacji sieci bezprzewodowych [14].

5.2 Problem niedokładności odczytów

Błąd odczytu odbiornika GPS zaobserwowany w trakcie testowania *PositionTracking* w optymalnych warunkach, tj. na otwartym terenie, wahał się od kilku do kilkunastu metrów, w zależności od miejsca odczytu. O ile błąd na poziomie dwóch-trzech metrów można uznać za akceptowalny, o tyle niedokładność rzędu dwunastu-piętnastu metrów może bardzo niekorzystanie wpływać na doświadczenia użytkowników aplikacji stworzonych przy pomocy *fARmework*.

Za przykład posłużyć może gra w "policjantów i złodziei", w której osoba goniąca przez cały czas jest informowana o tym, jaki dystans powinna przebyć i w jakim kierunku się poruszać, aby dotrzeć do osoby uciekającej. Dystans ten typowo wynosił będzie od zera do kilkuset metrów.

5.2.1 Przykład

W danym momencie policjant znajduje się w okolicach wydziału Elektroniki i Technik Informacyjnych (punkt A(52,219505° N, 21,012028° E)), a złodziej - na ulicy Lwowskiej (w punkcie B(52,220458° N, 21,012281° E)). Odległość między punktami wynosi 103 m, a azymut - 9,24°. Następują odczyty pozycji policjanta i złodzieja, oba z niedokładnością około dziesięciu metrów. Według odczytów, policjant znajduje się w punkcie A'(52,219518° N, 21,011892° E), a złodziej w punkcie B'(52,220466° N, 21,012431° E).

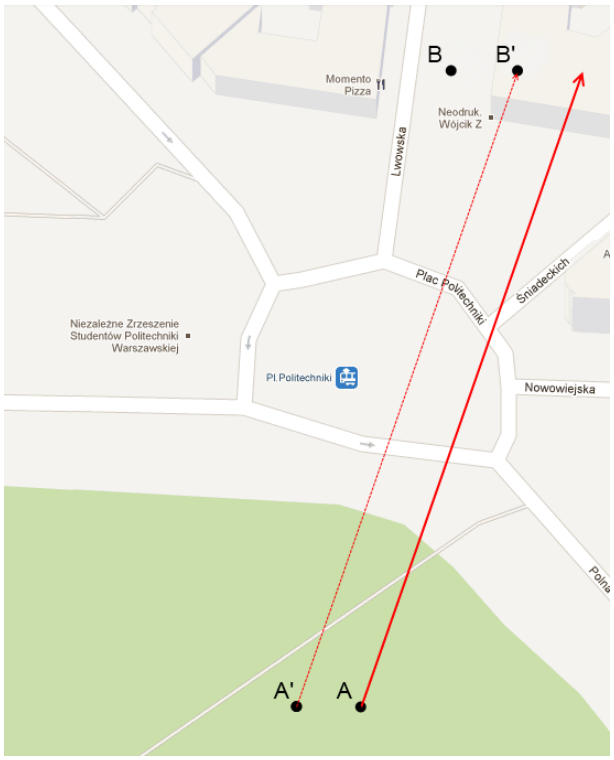
Obliczenia dla tych punktów wskazują, że policjant powinien przebyć 115,3 m w kierunku 19,2°. Niedokładność wskazań jest na tyle duża, że może zmylić policjanta - zamiast na ulicę Lwowską, prawdopodobnie pobiegnie on w kierunku ulicy Śniadeckich (sytuację obrazuje Rysunek 5.1).

Jak widać, dokładność jest istotna nawet w sytuacjach, w których odległość pomiędzy telefonami wynosi ponad sto metrów. W przypadku metod odczytu udostępnianych przez system Android nie jest możliwe całkowite usunięcie błędu odczytu - wynika on z jakości odbiorników GPS montowanych w smartfonach. Można jednakże próbować go zmniejszyć poprzez rozszerzenie istniejących metod o operacje korygujące takie jak filtrowanie lub uśrednianie kolejnych odczytów. Alternatywnym podejściem jest zaimplementowanie nowej metody dającej w określonych warunkach lepsze wyniki niż metody podstawowe. To właśnie podejście zastosowano po stworzeniu podstawowej wersji komponentu, czego wynikiem było opracowanie i zaimplementowanie dodatkowej metody odczytu pozycji.

5.3 Odczyt na podstawie siły sygnału pobliskich sieci WiFi

Problem niedokładności odczytów dokonywanych wewnątrz budynku jest zdecydowanie bardziej poważny niż w przypadku odczytów w otwartym terenie. Gra w "policjantów i złodziei" zrealizowana z wykorzystaniem bazowego komponentu *PositionTracking* prawdopodobnie nie nadawałaby się do rozgrywania w pomieszczeniach. Składają się na to dwa czynniki:

- z racji ograniczonego pola gry, odległości pomiędzy graczami byłyby zazwyczaj



Rysunek 5.1: Pesymistyczny przykład konsekwencji błędu odczytu pozycji. Linią przerywaną zaznaczono wynik dokonanego odczytu, a linią ciagłą - to, jak wynik zostanie zinterpretowany przez użytkownika. [Źródło mapy: https://maps.google.pl/]

znacznie mniejsze niż w otwartym terenie - wymagana byłaby większa dokładność odczytów pozycji,

- odczyty dokonywane wewnątrz budynku przy użyciu metod dostarczanych przez system Android są znacznie mniej dokładne niż w otwartym terenie.

Wymagana byłaby dokładność odczytów, której system Android nie jest w stanie zapewnić. W celu wypełnienia tej luki, komponent *PositionTracking* został rozszerzony o dodatkową metodę odczytu, bazującą na sile sygnałów sieci bezprzewodowych "widzianych" przez urządzenie.

5.3.1 Opis metody odczytu

Metoda implementowana przez komponent *PositionTracking.WiFi* przypomina przytoczoną wyżej metodę odczytu pasywnego realizowaną przez system Android. Jednakże zamiast opierać się na sile sygnału pochodzącego od stacji bazowych, analizuje sygnał odbierany przez urządzenie od punktów dostępowych[2] sieci Wi-Fi znajdujących się w jego pobliżu. Znając dokładną pozycję tych punktów i odległości dzielące je od urządzenia możliwe jest obliczenie jego lokalizacji.

[2]Punkt dostępowy (ang. Access Point, AP) - urządzenie (zazwyczaj router) zapewniające innym urządzeniom bezprzewodowe połączenie z siecią Internet.

Metoda częściowo bazuje na informacjach zawartych w dokumencie "Wi-Fi Location-Based Services - Design and Deployment Considerations" opracowanym przez firmę Cisco Systems [15]. Dokument ten rozważa realizowalność podobnej metody, podaje możliwe warianty wykorzystania sygnału sieci WiFi, a także przybliża dokładność takiego systemu (ok. 5 metrów). Spośród wymienionych w dokumencie sposobów wykorzystania sygnału sieci WiFi wybrano sposób oparty na wartości wskaźnika RSSI[3] sieci.

Aby zapewnić możliwie dokładny odczyt, metoda wymaga spełnienia następujących warunków:

1. Telefon wykrywa przynajmniej trzy punkty dostępowe.
   - Mniejsza liczba punktów uniemożliwi obliczenie pozycji urządzenia.
2. Znane są dokładne współrzędne geograficzne punktów dostępowych.
   - Współrzędne powinny zostać wyznaczone przez użytkownika aplikacji korzystającej z PositionTracking.WiFi.
   - Niespełnienie tego warunku jest jednym z powodów niskiej dokładności metody odczytu pasywnego wspomaganego analizą sygnałów sieci Wi-Fi.
3. Znane są zasięgi sieci udostępnianych przez punkty dostępowe.
   - Zasięgi powinny zostać wyznaczone przez użytkownika aplikacji.
4. Punkty dostępowe są od siebie oddzielone możliwie małą liczbą ścian.
   - Metoda nie koryguje osłabienia sygnału powodowanego przez przeszkody takie jak ściany i drzwi.

Warunki te są trudne do spełnienia w przypadku gier takich jak "policjanci i złodzieje", które mogą rozgrywać się w dowolnym miejscu. Są natomiast osiągalne dla aplikacji, których polem działania jest ustalony obszar. Przykładową aplikacją tego rodzaju może być narzędzie wspomagające grę w paintball (aplikacja informowałaby gracza o pozycjach członków jego drużyny) odbywającą się w przystosowanej do tego hali. W takim przypadku wystarczy rozlokowanie w hali punktów dostępowych i zmierzenie ich pozycji i zasięgów.

Proces wyznaczania pozycji telefonu przebiega następująco:

1. Następuje skanowanie sygnałów wszystkich widocznych sieci Wi-Fi.
2. Odrzucane są sygnały pochodzące od nieznanych punktów dostępowych.
3. Wybierane są trzy odczyty o największej sile sygnału.
4. Wyznaczana jest odległość urządzenia od wybranych w poprzednim kroku punktów dostępowych.
5. Dokonywana jest trilateracja[4] pozycji telefonu.

[3]RSSI (Received Signal Strength Indication) - wskaźnik mocy odbieranego sygnału radiowego. Jest to wartość chwilowa, różniąca się w kolejnych odczytach.

[4]Trilateracja – wyznaczanie współrzędnych punktu wewnąrz trójkąta na podstawie jego odległosci od wierzchołków tego trójkąta.

5.3.2 Odczyt siły sygnału sieci bezprzewodowej

W celu odczytania siły sygnałów pobliskich sieci bezprzewodowych, moduł *PositionTracking.WiFi.Android* korzysta klasy *WifiManager* udostępnianej przez API systemu Android [16]. Metody tej klasy (*startScan* i *getScanResults*) umożliwiają skanowanie otoczenia i uzyskiwanie informacji takich jak SSID[5], BSSID[6] i RSSI pobliskich sieci bezprzewodowych.

Dla każdej z widocznych sieci pobierana jest para wartości: BSSID i RSSI. Na podstawie wartości RSSI danej sieci obliczana jest procentowa wartość określająca stosunek mocy sygnału odbieranego w danej chwili do mocy nominalnej tej sieci. W tym celu wykorzystywany jest wzór:

$$strength = \begin{cases} 0\% & : RSSI \leqslant RSSI_{min} \\ 100\% & : RSSI \geqslant RSSI_{max} \\ \frac{RSSI - RSSI_{min}}{RSSI_{max} - RSSI_{min}} * 100\% & : RSSI \in (RSSI_{min}, RSSI_{max}) \end{cases},$$

gdzie:

- $RSSI$ - wartość RSSI w danej chwili,
- $RSSI_{min}$ - ustalona z góry minimalna wartość RSSI, równa -100,
- $RSSI_{max}$ - ustalona z góry maksymalna wartość RSSI, równa -55.

Klasa *WifiManager* udostępnia tę operację (metoda *calculateSignalLevel*), jednak jej implementacja zawiera błąd - dla pewnych wartości występuje w niej dzielenie przez 0. W oparciu o kod źródłowy błędnej metody operacja ta została zaimplementowana poprawnie.

5.3.3 Wyznaczenie odległości dzielącej urządzenie od punktu dostępowego

Uzyskane pary BSSID-strength przekazywane są modułowi *PositionTracking.WiFi.Java* do dalszego przetworzenia. Po odfiltrowaniu nieznanych sieci i wybraniu trzech odczytów o największej sile sygnału następuje wyznaczenie odległości telefonu od wybranych punktów dostępowych.

Aby wyznaczyć zależność procentowej wartości siły sygnału od odległości dzielącej urządzenie od punktu dostępowego, przeprowadzono pomiar w dużym pomieszczeniu wolnym od przeszkód. Jako punktu dostępowego użyto rutera D-link Wireless N150 DIR-600[7]. Kolejne pomiary uzyskano pięciokrotnie mierząc siłę sygnału w różnych odległościach od punktu dostępowego aż do utraty widoczności sieci. Interwał czasowy pomiędzy kolejnymi pomiarami wynosił 10 sekund, a skok odległości - 5 metrów. Wyniki pomiarów przedstawia Tabela 5.1.

[5]SSID (Service Set IDentifier) - identyfikator sieci bezprzewodowej nadawany danej sieci przez jej administratora.

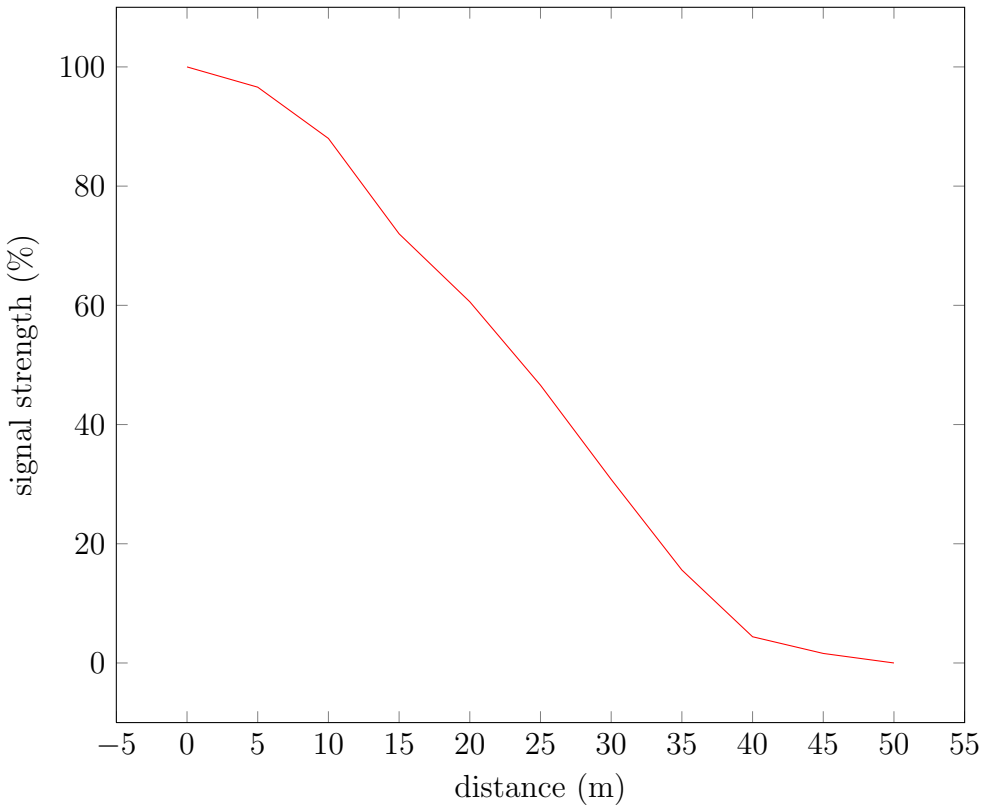
[6]BSSID (Basic Service Set IDentification) - numer identyfikacyjny punktu dostępowego, nadawny automatycznie; najczęściej jest tożsamy z adresem MAC punktu dostępowego.

[7]Opis i specyfikacja urządzenia:
http://www.dlink.com/uk/en/home-solutions/connect/routers/dir-600-wireless-n-150-home-router

odległość (m)	kolejne odczyty (%)	średnia (%)
0	100, 100, 100, 100, 100	100
5	97, 93, 100, 97, 96	96,6
10	95, 95, 82, 84, 84	88
15	73, 73, 73, 75, 66	72
20	62, 65, 58, 58, 60	60,6
25	48, 46, 51, 44, 44	46,6
30	31, 35, 29, 31, 28	30,8
35	15, 17, 13, 20, 13	15,6
40	2, 5, 7, 5, 3	4,4
45	1, 2, 4, 0, 1	1,6
50	0, 0, 0, 0, 0	0

Tabela 5.1: Zależność procentowej siły sygnału sieci WiFi docierającego do telefonu od odległości pomiędzy telefonem a ruterem. Wartość 0 oznacza brak widoczności sieci.

W celu analizy uzyskanych pomiarów sporządzono wykres uśrednionej wartości odczytów w funkcji odległości od punktu dostępowego. Wykres ten przedstawiony został na Rysunku 5.2.



Rysunek 5.2: Wykres zależności średniej procentowej wartości siły sygnału sieci WiFi docierającego do telefonu od odległości pomiędzy telefonem a ruterem.

Jak widać, dla przedziału $5-40$ metrów siła odbieranego sygnału maleje w przybliżeniu liniowo. W przedziale $0-10$ metrów zależność ta nie występuje prawdopodobnie z powodu braku zakłóceń w bezpośrednim sąsiedztwie źródła sygnału - w odległości kilku metrów od punktu dostępowego siła sygnału maleje więc wolniej. W przedziale $40-50$ metrów sygnał powoli zanika - jest on widoczny dla telefonu, ale ze względu na jego niską moc, system nie jest w stanie poprawnie wyznaczyć jego wskaźnika RSSI.

W celu sformułowania wzoru określającego odległość od punktu dostępowego na podstawie siły sygnału, przyjęto następujące założenia:

- użytkownik aplikacji korzystającej z *PositionTracking.WiFi* bardzo rzadko znajdował się będzie w bezpośrednim sąsiedztwie punktu dostępowego,
- punkty dostępowe rozlokowane będą na tyle gęsto, że telefon użytkownika zawsze będzie odbierał przynajmniej trzy sygnały o sile powyżej 15%.

Spełnienie tych założeń gwarantuje, że urządzenie w zdecydowanej większości przypadków znajdować się będzie w przedziale odległości, dla których siła sygnału maleje w przybliżeniu liniowo. Pozwala to przyjąć liniową zależność pomiędzy siłą sygnału a odległością od punktu dostępowego. Wzór wyrażający tę zalezność jest następujący:

$$distance = (1 - strength) * range,$$

gdzie:

- *strength* - procentowa siła sygnału pochodzącego od punktu dostępowego - wartość z zakresu $[0, 1]$,
- *range* - zasięg sieci udostępnianej przez punkt dostępowy - wyrażony w metrach[8].

5.3.4 Trilateracja położenia telefonu

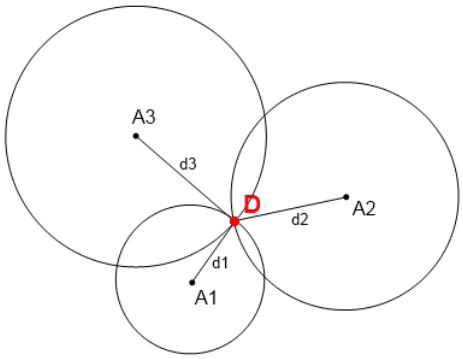
Po obliczeniu odległosci dzielących urządzenie od trzech punktów dostępowych, moduł *PositionTracking.WiFi.Java* dokonuje trilateracji jego pozycji. W tym celu wykorzystywana jest znajomość współrzędnych geograficznych punktów dostępowych i wyznaczonych odległości. Kształt Ziemi jest w tym przypadku przybliżany do płaszczyzny - dla odległości rzędu kilkudziesięciu metrów nie wnosi to znaczącego błędu obliczeniowego. Teoretyczny problem trilateracji obrazuje Rysunek 5.3.

Dla takiego problemu algorytm rozwiązania przebiegałby następująco:

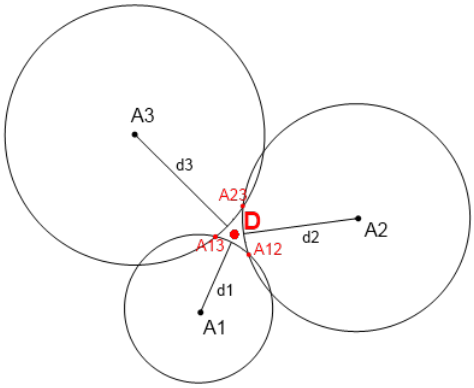
1. Wyznaczana jest para punktów przecięcia okręgów $o(A_1, d_1)$ i $o(A_2, d_2)$ [17].
2. Wyznaczana jest para punktów przecięcia okręgów $o(A_1, d_1)$ i $o(A_3, d_3)$.
3. Znajdowany jest punkt należący do obu wyznaczonych par.

Jednakże w praktyce sytuacja taka nie ma szansy wystąpić - z racji niedokładności odczytów siły sygnału i obliczeń wyznaczających odległość telefonu od punktów dostępowych, rozpatrywane okręgi nie przetną się w jednym punkcie. Przykład rzeczywistego problemu został przedstawiony na Rysunku 5.4.

[8]W przypadku badanego rutera wartość *range* wynosi 45.



Rysunek 5.3: Przykładowy teoretyczny problem trilateracji pozycji urządzenia (D) przy znajomości pozycji punktów dostępowych (A_1, A_2, A_3) i odległości dzielącej urządzenie od tych punktów (d_1, d_2, d_3).



Rysunek 5.4: Przykład rzeczywistego problemu trilateracji z uwzględnieniem niedokładności odczytu odległości d_1, d_2, d_3. A_{12}, A_{13}, A_{23} - punkty przecięć okręgów.

Z racji błędnych wartości odległości d_1 - d_3 nie jest możliwe dokładne wyznaczenie pozycji urządzena. Z tego powodu wyznaczna jest pozycja przybliżona. Kroki algorytmu są następuje:

1. Wyznaczane są wszystkie pary punktów przecięć okręgów $o(A_1, d_1)$, $o(A_2, d_2)$, $o(A_3, d_3)$.
2. Spośród trójkątów utworzonych poprzez wybranie jednego wierzchołka z każdej pary wyznaczonej w poprzednim punkcie, znajdowany jest trójkąt o najmniejszym obwodzie ($\triangle A_{12}A_{13}A_{23}$).
3. Wyznaczane są współrzędne środka okręgu opisanego na znalezionym trójkącie. W tym celu wykorzystywane są wzory [18]:

$$x = \frac{(x_{12}^2+y_{12}^2)(y_{13}-y_{23})+(x_{13}^2+y_{13}^2)(y_{23}-y_{12})+(x_{23}^2+y_{23}^2)(y_{12}-y_{13})}{2(x_{12}(y_{13}-y_{23})+x_{13}(y_{23}-y_{12})+x_{23}(y_{12}-y_{13}))},$$

$$y = \frac{(x_{12}^2+y_{12}^2)(x_{23}-x_{13})+(x_{13}^2+y_{13}^2)(x_{12}-x_{23})+(x_{23}^2+y_{23}^2)(x_{13}-x_{12})}{2(x_{12}(y_{13}-y_{23})+x_{13}(y_{23}-y_{12})+x_{23}(y_{12}-y_{13}))},$$

gdzie:

- x_i - szerokość geograficzna punktu A_i,
- y_i - długość geograficzna punktu A_i,
- (x, y) - współrzędne punktu zwracanego przez algorytm.

5.3.5 Testowanie metody i jej skuteczność

W celu przetestowania metody wyznaczono współrzędne geograficzne trzech punktów, w którch ustawiono punkty dostępowe. Jako punktów dostępowych użyto trzech ruterów D-link Wireless N150 DIR-600 (był to jedyny model dostępny w miejscu przeprowadzania pomiarów). Punkty te tworzyły w przybliżeniu trójkąt równoboczny o długości boku równej około 30 metrów. Wewnątrz tego trójkąta ustalono trzy punkty pomiarowe.

W kolejnych punktach pomiarowych zapisywano odczyty otrzymywane przy użyciu przedstawionej metody oraz metody podstawowej (dostarczanej przez system Android). Porównanie wyników zawiera Tabela 5.2.

	PositionTracking.WiFi	Android
błąd minimalny	4,5 m	16 m
błąd maksymalny	13 m	32 m
błąd średni	7,26 m	23,6 m

Tabela 5.2: Opracowanie wyników pomiarów dokładności metod odczytu pozycji telefonu wewnątrz budynku.

Jak widać, dokładność zaimplementowanej metody nie jest w pełni satysfakcjonująca. Aby móc z powodzeniem używać aplikacji na niej opartych, należy korzystać z pomieszczeń o znacznych rozmiarach[9].

Jednakże głównym celem stworzenia rozszerzenia *PositionTracking.WiFi* nie było osiągnięcie bardzo wysokiej dokładności, a jedynie dokładności lepszej od oferowanej przez system Android. Pod tym względem można mówić o sukcesie - opracowana metoda jest kilkukrotnie dokładniejsza od metody podstawowej.

5.4 Zaimplementowane operacje obliczeniowe

Moduł *PositionTracking.Java* udostępnia podstawowe operacje na współrzędnych geograficznych, przydatne w aplikacjach śledzących pozycje użytkowników:

5.4.1 Wyznaczenie odległości pomiędzy dwoma punktami na kuli ziemskiej

Z racji kształtu kuli ziemskiej, dokładne wyznaczenie odległości pomiędzy dwoma punktami, znając ich współrzędne geograficzne, nie jest zadaniem trywialnym. Przybliżenie

[9]Dotyczy to aplikacji, dla których wymagana dokładność maleje wraz ze wzrostem rozmiarów pola działania.

kształtu Ziemi do kuli nie wnosi jednak znacznego błędu.

PositionTracking wykorzystuje takie przybliżenie i oblicza odległośc na podstawie wzoru [19]:

$$d = S\frac{\pi}{360}\sqrt{(\cos(y_1\frac{\pi}{180})(x_2 - x_1))^2 + (y_2 - y_1)^2},$$

gdzie:

- x_1 - szerokość geograficzna pierwszego punktu,
- y_1 - długość geograficzna pierwszego punktu,
- x_2 - szerokość geograficzna drugiego punktu,
- y_2 - długość geograficzna drugiego punktu,
- S - czynnik skalujący wynik tak, aby był on wyrażony w metrach, wynoszący 12756274 (m).

5.4.2 Wyznaczenie azymutu pomiędzy dwoma punktami

W celu obliczenia azymutu, *PositionTracking* również przybliża kształt Ziemi do kuli. Operacja ta realizowana jest z wykorzystaniem wzoru [20]:

$$a = \text{atan2}(\sin(\Delta y)\cos(x_2), \cos(x_1)\sin(x_2) - \sin(x_1)\cos(x_2)\cos(\Delta y)),$$

gdzie:

- $\Delta y = y_2 - y_1$,
- x_1 - szerokość geograficzna pierwszego punktu,
- y_1 - długość geograficzna pierwszego punktu,
- x_2 - szerokość geograficzna drugiego punktu,
- y_2 - długość geograficzna drugiego punktu,
- atan2 - funkcja zaimplementowana w bibliotekach matematycznych wielu języków programowania, zdefiniowana następująco:

$$\text{atan2}(y, x) = \begin{cases} \arctan(\frac{y}{x}) & : x > 0 \\ \arctan(\frac{y}{x}) + \pi & : y \geqslant 0, x < 0 \\ \arctan(\frac{y}{x}) - \pi & : y < 0, x < 0 \\ +\frac{\pi}{2} & : y > 0, x = 0 \\ -\frac{\pi}{2} & : y < 0, x = 0 \\ 0 & : y = 0, x = 0 \end{cases}$$

5.5 Wykorzystanie PositionTracking i PositionTracking.WiFi

Aby odczytywać pozycję telefonu przy użyciu podstawowych metod udostępnianych przez system Android, należy użyć klasy *PositionProvider* modułu *PositionTracking.Android*. Operacje na współrzędnych geograficznych zostały zaimplementowane w klasach *DistanceCalculator* i *DirectionCalculator* należących do modułu *PositionTracking.Java*.

Aby skorzystać z komponentu *PositionTracking.WiFi* należy dostarczyć modułowi *PositionTracking.WiFi.Java* informacji na temat znanych punktów dostępowych. Jeden punkt dostępowy opisują trzy wartości: adres MAC punktu, jego współrzędne geograficzne i zasięg sieci przez niego udostępnianej. Informacje te są wykorzysywane przez klasę *PositionCalculator* w celu obliczenia pozycji urządzenia. Danych dotyczących siły sygnałów pobliskich sieci bezprzewodowych dostarcza klasa *WiFiDataProvider* umieszczona w module *PositionTracking.WiFi.Android*.

Komponent *PositionTracking* może zostać wykorzystany w wielu typach aplikacji wykorzystujących informacje o lokalizacji ich użytkowników. Mogą to być:

- gry, takie jak wymienione wcześniej podchody, "policjanci i złodzieje", czy paintball,
- wszelkiego rodzaju nawigatory, na przykład zapamiętujące przebytą przez użytkownika ścieżkę, tak aby mógł on łatwo wrócić po swoich śladach,
- aplikacje informujące użytkownika o obiektach znajdujących się w jego otoczeniu - przykładem takiej aplikacji może być przewodnik turystyczny.

Rozdział 6

Wyświetlanie stabilizowanej grafiki trójwymiarowej na ekranie telefonu

Pojęcie rzeczywistości rozszerzonej skupia się głównie na grafice wyświetlanej na obrazie odczytywanym z kamery telefonu. W takim ujęciu, komponentem najpełniej realizjującym ideę AR jest *SpaceGraphics*. Jest to pojedynczy moduł (*SpaceGraphics.Android*) zajmujący się renderowaniem trójwymiarowej grafiki na ekranie telefonu. Wyświetlany model obracany jest tak, aby pozostawał nieruchomy względem otoczenia urządzenia.

6.1 Pierwotne założenie

Pierwontym zamiarem wobec *SpaceGraphics* było stworzenie komponentu, który pozwala umieścić wirtualną grafikę dwuwymiarową w rzeczywistej lokalizacji. Obiekty graficzne wraz z danymi takimi jak współrzędne geograficzne i orientacja urządzenia w momencie umieszczenia grafiki byłyby przechowywane na serwerze, i, po zbliżeniu się użytownika do danej lokalizacji, przesyłane na jego telefon. Następnie, obraz byłby przetwarzany w zależności od kąta, pod jakim użytkownik patrzy (za pośrednictwem kamery telefonu) na obiekt i od dzielącej go od niego odległości. Wyświetlany obraz wyglądałby tak, jak gdyby rzeczywiście znajdował się w danym miejscu - w wyniku przetwarzania byłby odpowiednio obrócony i pomniejszony.

Przykładowo taki komponent mógły służyć do zostawiania i poszukiwania śladów w grze podchody. Uciekający umieszczałby strzałki na ścianach budynków lub na chodniku, a zadaniem goniącego byłoby wypatrywanie tych strzałek przy pomocy telefonu. Innym zastosowaniem mogłaby być wirtualna tablica ogłoszeń zlokalizowana na dowolnej ścianie gmachu Wydziału Elektroniki i Technik Infromacyjnych - ogłoszenia byłyby widoczne tylko na ekranie urządzenia, po wskazaniu ich pozycji obiektywem kamery.

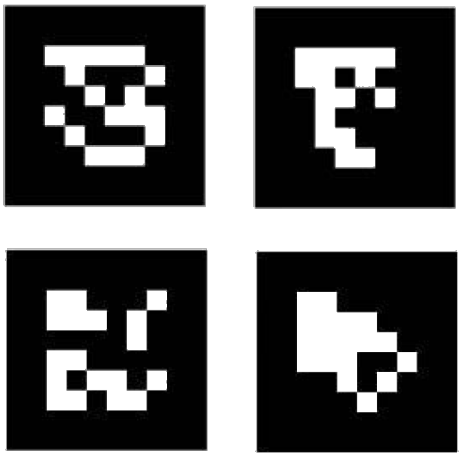
Podczas rozważań teoretycznych na temat możliwości realizacji takiej funkcjonalności okazało się jednak, że dostępne narzędzia nie są wystarczająco dokładne. Główną przeszkodą jest dokładność odbiornika GPS - w przykładzie tablicy ogłoszeń wymagana byłaby dokładność na poziomie jednego metra.

Kolejna przeszkoda to odczytywanie wysokości, na jakiej znajduje się użytkownik aplikacji. Odbiornik GPS dostarcza informacji o wysokości, ale jest ona bardzo niedokładna. Aby uzyskać dokładny odczyt, należy odczyt GPS połączyć z odczytem barometru, który

jest montowany głównie w urządzeniach z najwyższej półki, takich jak Samsung Galaxy SIII, czy Sony Xperia GO[1]. W przypadku budynku, niedokładny odczyt wysokości mógłby powodować wyświetlenie grafiki na niewłaściwym piętrze, co byłoby niedopuszczalne w wielu typach aplikacji (np. wspomniana juz gra podchody).

Sensory służące do lokalizacji telefonu są więc zbyt niedokładne lub zbyt rzadko stosowane, by móc oprzeć na nich algorytm określania położenia wirtualnej grafiki. Potencjalnym usprawnieniem może być posiłkowanie się technikami rozpoznawania obrazu w celu znalezienia tła, na którym umieszczona została grafika. Przybliżona pozycja urządzenia byłaby odczytywana za pomocą GPS, a dokładne miejsce, w którym należy wyświetlić obiekt, byłoby wyznaczane na podstawie analizy obrazu z kamery telefonu. Ta możliwość została jednak odrzucona ze względu na jego potencjalną zawodność - precyzyjne umieszczenie grafiki w budynku o jednolitych ścianach w dalszym ciągu nie byłoby możliwe - przy wysokiej trudności wykonania (należałoby wykorzystać do sześciu sensorów telefonu[2] i zaimplementować uniwersalny algorytm rozpoznawania tła).

W dalszych rozważaniach wzięto pod uwagę metodę, która korzysta z relatywnie prostej techniki rozpoznawania obrazu, a zarazem jest odporna na problem jednolitego tła. Mowa o Wykrywaniu Znaczników (ang. Marker Detection) - jest to technika polegająca na umieszczeniu w rzeczywistym świecie znacznika, który zostanie łatwo rozpoznany przez telefon[21]. Przykłady znaczników używanych przy implementacji Wykrywania Znaczników zawiera Rysunek 6.1



Rysunek 6.1: Przykłady znaczników [21].

Na takim znaczniku telefon mógłby z powodzeniem wyświetlać grafikę. Rozwiązanie to ma jednak kilka wad, które spowodowały jego odrzucenie:

- użytkownik aplikacji, aby umieścić grafikę w danej lokalizacji, musi posiadać znacznik,
- umieszczenie znaczników w niektórych lokalizacjach (na przykład na ścianie bloku mieszkalnego) byłoby nielegalne lub uznane za akt wandalizmu,

[1]Modele te w czasie projektowania *SpaceGraphics* nie były jeszcze dostępne na rynku.
[2]GPS, barometr, akcelerometr, żyroskop, magnetometr, kamera.

- o ile w przypadku tablicy ogłoszeń znacznik może być umieszczony na stałe, o tyle w przypadku gier, po zakończeniu rozgrywki należałoby usunąć umieszczone znaczniki,
- znaczniki są widoczne nie tylko dla urządzenia, ale również dla użytkownika aplikacji - w grze podchody, goniący szukałby nie wirtualnych strzałek, a rzeczywistych znaczników.

Powyższy proces poszukiwania odpowiedniej implementacji założonej funkcjonalności doprowadził do wniosku mówiącego, że na chwilę obecną nie ma możliwości osiągnięcia jej satysfakcjonującej realizacji. Ostatecznie zdecydowano, że komponent wyświetlał będzie grafikę w przestrzeni, a nie na powierzchniach płaskich. Aby maksymalnie zwiększyć obszar zastosowań tego rozwiązania, wyświetlana jest grafika trójwymiarowa. W tym przypadku nie jest wymagane bardzo dokładne określenie pozycji telefonu. Przykładowo, w grze "podchody" uciekający zostawia za sobą trójwymiarowe strzałki, na które goniący może trafić kilka metrów od ich oryginalnego położenia - jako że są one "zawieszone" w powietrzu, a nie "przyklejone" do ściany budynku, gracz nie jest świadomy niedokładności odczytu swojej pozycji (jednakże w tym przypadku mają zastosowanie konsekwencje niedokładności omówione w poprzednim rozdziale).

6.2 Renderowanie grafiki

Do generowania grafiki trójwymiarowej system Android wykorzystuje biblioteki OpenGL ES (OpenGL for Embedded Systems). Jest to odmiana biblioteki OpenGL dedykowana dla systemów wbudowanych używanych między innymi w konsolach i telefonach [22]. Android oferuje wsparcie dla dwóch standardów biblioteki:

- OpenGL ES 1.x - wykorzystuje potok ustalony (ang. fixed pipeline). Oznacza to, że transformacje oraz obliczenia koloru wierzchołków wykonywane w celu imitacji cieniowania i oświetlenia wykonywane są przez kartę graficzną w sposób narzucony z góry. Utrudnia to lub uniemożliwia uzyskanie wielu zaawansowanych efektów graficznych, takich jak mapowanie wypukłości[3], płaszczyzny refleksji[4] [23] czy renderowanie z użyciem szerokiego zakresu dynamicznego[5] [24].
- OpenGL ES 2.x - wykorzystuje potok programowalny (ang. programmable pipeline). Umożliwia pisanie krótkich programów w języku GLSL[6] nazywanych shaderami, które definiują sposób wykonywania obliczeń na wierzchołkach. Umożliwia to uzyskiwanie zaawansowanych efektów graficznych. Jest jednak trudne do zrozumienia dla początkujących programistów, gdyż wykonywanie transformacji lub oświetlenie

[3]Mapowanie wypukłości (ang. bump mapping) - technika umożliwiająca symulowanie obiektów o porowatej powierzchni.

[4]Płaszczyzny refleksji (ang. reflection planes) - technika umożliwiająca uzyskanie efektu lustrzanego odbicia sceny na danej powierzchni.

[5]Renderowanie z użyciem szerokiego zakresu dynamicznego (ang. high dynamic range rendering) - technika umożliwiająca uzyskanie realistycznego oświetlenia przez wierne odwzorowanie bardzo jasnych i ciemnych elementów sceny.

[6]GLSL - OpenGL Shading Language, http://www.opengl.org/documentation/glsl/.

sceny wymaga podania kodu źródłowego tych operacji jako wartości zmiennych napisowych (*String*) - poprawność tego kodu nie jest sprawdzana przez kompilator, co znacznie utrudnia wykrycie ewentualnego błędu.

Ze względu na brak wykorzystania efektów wymagających użycia potoku programowalnego, *fARmework* wykorzystuje bibliotekę OpenGL ES w standardzie 1.x. Klasy jednakże zostały wydzielone w sposób, który umożliwia transparentną wymianę wersji biblioteki bez konieczności redefiniowania modelu.

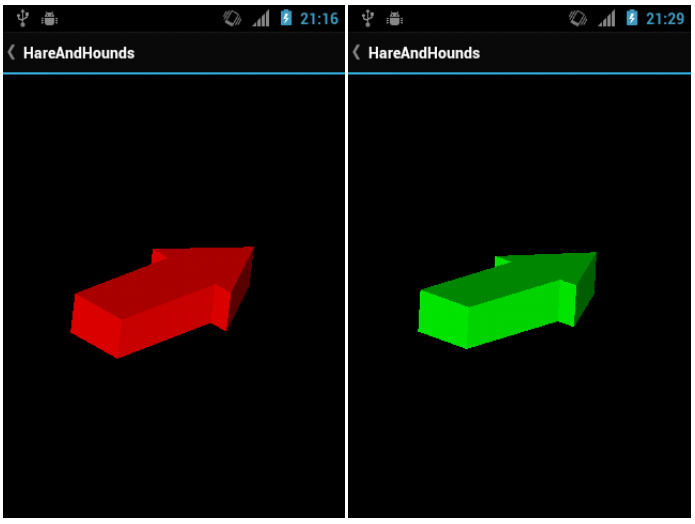
6.2.1 Definicja modelu

Renderowany na ekranie obiekt rozszerza klasę *Model* i jest opisany trójkątnymi ścianami. Do zdefiniowania modelu wymagane są następujące informacje:

- tablica współrzędnych wierzchołków obiektu - każdy wierzchołek opisany jest trzema współrzędnymi (x, y, z),
- tablica kolorów wierzchołków obiektu - każdy wierzchołek opisany jest czterema wartościami (r, g, b, a) definiującymi jasności kanałów: czerwonego, zielonego, niebieskiego i przezroczystości.

W komponencie zdefiniowany jest również specjalny typ modelu - *PhasingModel*. Ma on zdolność automatycznej transformacji koloru wierzchołków na podstawie zadanego współczynnika z zakresu $[0, 1]$. Przed wyświetleniem, wybrane przez programistę składowe barw wierzchołków modelu są mnożone przez ten współczynik. Dzięki temu, wraz ze zmianą wartości współczynnika, model automatycznie zmienia swój kolor.

Komponent zawiera jeden przykładowy model typu *PhasingModel*, przygotowany na potrzeby gry *Hare and Houds*. Jest to strzałka, która wskazuje użytkownikowi kierunek poruszania się. Przykłady kolorowania tego modelu prezentuje Rysunek 6.2.



Rysunek 6.2: Przykład kolorowania modelu strzałki.

6.2.2 Proces renderowania

Dla każdego zdefiniowanego modelu moduł renderujący grafikę automatycznie:

1. Oblicza i normalizuje wartości wektorów normalnych dla każdej ściany, co umożliwia poprawne oświetlenie obiektu.
2. Ustawia położenie kamery oraz definiuje sposób wyświetlania sceny:
   - kamera jest automatycznie pozycjonowana ponad zdefiniowanym obiektem,
   - scena wyświetlana jest domyślnie z użyciem projekcji ortogonalnej, co zwiększa czytelność modelu.
3. Dokonuje obrotu modelu zgodnie z przekazaną do modułu macierzą obrotów. Pozwala to wypozycjonować go tak, aby niezależnie od orientacji telefonu pozostawał nieruchomy względem otoczenia.
4. Dokonuje obrotu modelu zgodnie z przekazanym do modułu azymutem. Pozwala to na orientację obiektu tak, aby wskazywał zadany kąt względem magnetycznej północy.
5. Definiuje kolor wierzchołków (w przypadku modelu typu *PhasingModel*).
6. Rysuje zdefiniowany obiekt.
7. Oświetla scenę.

Dzięki temu programista może swobodnie używać komponentu bez znajomości biblioteki OpenGL. Wystarczy, że wygeneruje współrzędne oraz kolory wierzchołków trójwymiarowego obiektu (co można wykonać automatycznie narzędziami takimi jak 3ds Max[7]) i jako model przekaże je do komponentu (patrz sekcja 6.4), a reszta pracy zostanie wykonana samoczynnie.

Automatyczne obliczanie normalnych zostało wykonane z użyciem metody Newella dla ścian zdefiniowanych trójkątami [25]. Mając trójkąt o wierzchołkach p_1, p_2, p_3, dla wektorów $U = \overrightarrow{p_1p_2}$ oraz $V = \overrightarrow{p_1p_3}$, wektor normalny $N = U \times V$ można wyznaczyć korzystając z wzorów (6.1), (6.2), (6.3).

$$N_x = U_yV_z - U_zV_y \tag{6.1}$$

$$N_y = U_yV_z - U_zV_y \tag{6.2}$$

$$N_z = U_yV_z - U_zV_y \tag{6.3}$$

6.3 Obracanie modelu

Aby utrzymać model w pozycji pozornie nieruchomej względem otoczenia telefonu, niezbędna jest wiedza o aktualnej orientacji urządzenia i obracanie modelu przeciwnie do tej orientacji.

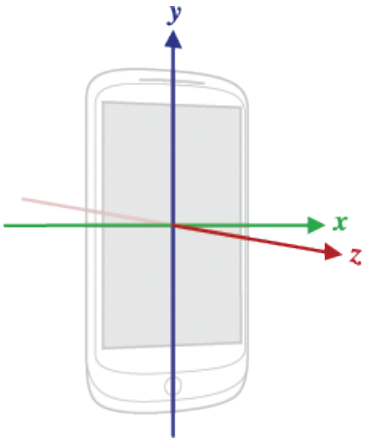
[7] 3ds Max - oprogramowanie służące do modelowania, renderowania i animowania grafiki 3D. Strona produktu dostępna jest pod adresem: http://usa.autodesk.com/3ds-max/.

6.3.1 Wyznaczenie orientacji telefonu

Źródłem wiedzy o orientacji urządzenia są odczyty z wbudowanych w telefon sensorów: akcelerometru i magnetometru.

Akcelerometr

Dostarcza informacji o przyśpieszeniach działających na telefon. Przykładowo jeśli telefon trzymany jest pionowo w miejscu, to działa na niego jedynie siła grawitacji, dając odczyt wynoszący około $9,81\frac{m}{s^2}$ na osi y. Zorientowanie osi obrazuje Rysunek 6.3.



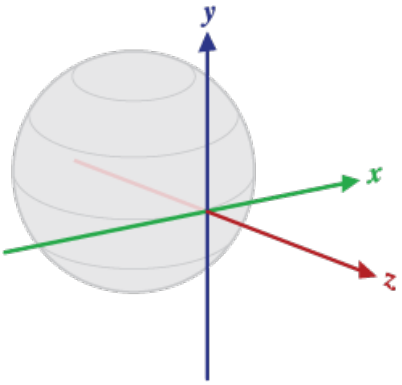
Rysunek 6.3: Zorientowanie osi telefonu dla odczytów akcelerometru [26].

Na podstawie informacji o tym, na jakie osie i w jakim stopniu działa siła grawitacji, można wnioskować o nachyleniu telefonu. Jako że akcelerometr mierzy przyśpieszenie wypadkowe, podczas ruchu urządzenia wnioskowanie to jest obarczone dużym błędem. Aby zminimalizować błąd, w systemie Android zastosowano filtr dolnoprzepustowy, który częściowo odcina wpływ poruszania na dokładność odczytów.

Magnetometr

Odczytuje wartości pola magnetycznego otaczającego urządzenie. Na podstawie tych wartości można określić, w którą stronę świata zwrócony jest telefon. Sensor podaje odczyty dla wszystkich trzech osi - orientacja urządzenia nie jest więc przeszkodą. Należy jednak zwrócić uwagę na to, że odczyty nie mają szansy być bardzo dokładne - magnetometr mierzy nie tylko pole magnetyczne Ziemi, ale także pola generowane przez inne podzespoły telefonu, na przykład głośnika.

Klasa *SensorManager* należąca do API systemu Android udostępnia operację (*getRotationMatrix*) pozwalają wyznaczyć macierz obrotów telefonu. Macierz tę można zastosować do konwersji współrzędnych punktu z układu współrzędnych telefonu do układu współrzędnych jego otoczenia (zorientowanie osi otoczenia przedstawia Rysunek 6.4). Aby ją uzyskać, należy pobrać odczyty akcelerometru i magnetometru i przekazać je metodzie *getRotationMatrix*.



Rysunek 6.4: Zorientowanie osi świata dla orientacji telefonu [27].

6.3.2 Orientacja urządzenia a tryb wyświetlania

Macierz obrotu zwracana przez metodę *getRotationMatrix* przechowuje wartości określające fizyczną orientację telefonu. Obrót modelu według tych wartości będzie niepoprawny, jeśli ekran urządzenia będzie wyświetlał treści w trybie innym niż domyślny. Dwa podstawowe tryby wyświetlania, to:

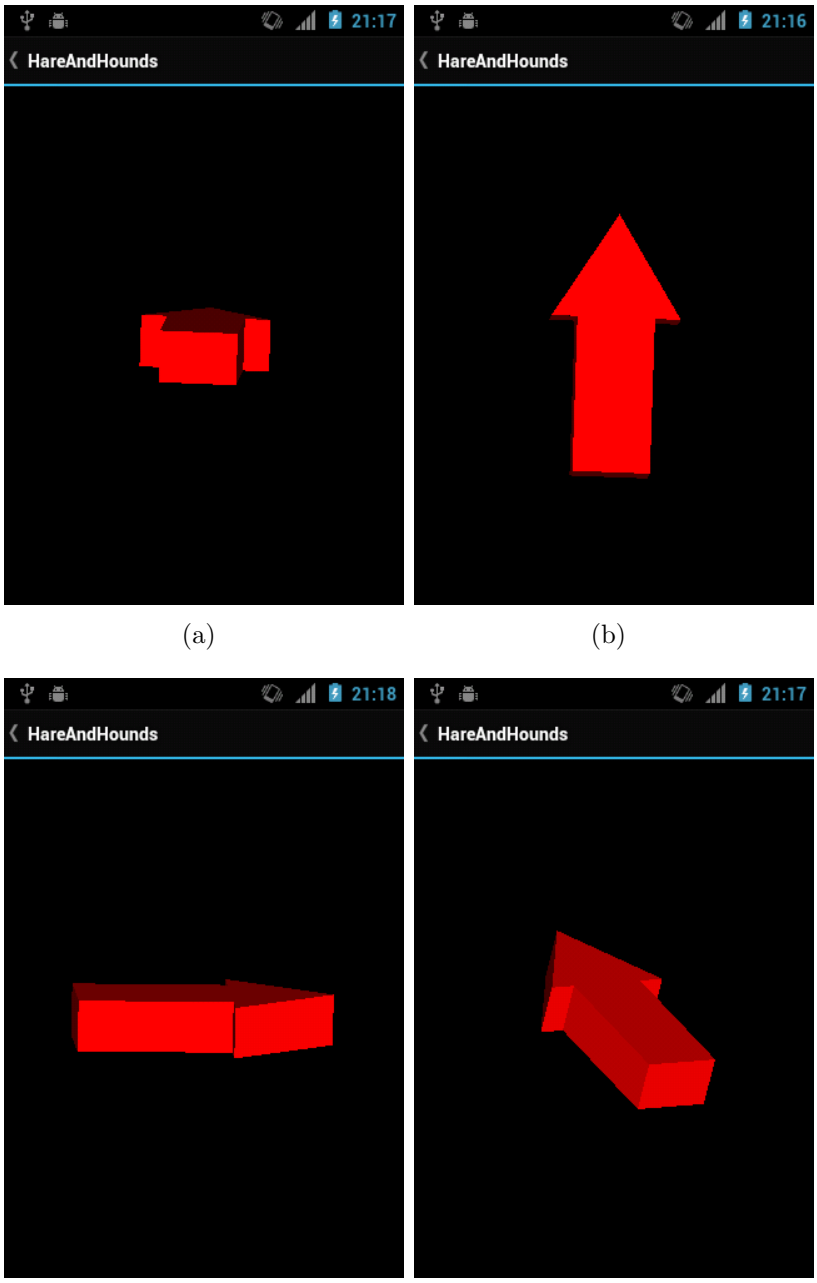
- portrait mode (tryb portretu, domyślny) - tryb używany, gdy telefon znajduje się pozycji pionowej,
- landscape mode (tryb krajobrazu) - tryb używany w pozycji poziomej; interfejs graficzny aplikacji jest odpowiednio obrócony.

Przełączanie pomiędzy trybami odbywa się automatycznie w zależności od pozycji, w jakiej znajduje się urządzenie. Zachowanie to można wyłączyć korzystając z ustawień systemu, a także kontrolować z poziomu aplikacji - dlatego wyznaczenie poprawnej macierzy obrotów jedynie na podstawie odczytów sensorów nie zawsze jest możliwe.

Android udostępnia informacje o aktualnie stosowanym trybie wyświetlania (dostarcza ich metoda *getRotation* klasy *Display*). W przypadku, gdy jest on inny od domyślnego, macierz należy skorygować, tzn. zamienić miejscami dwie z trzech osi. Na przykład przy obrocie ekranu o 90° w prawo, oś y wskazuje tam, gdzie poprzednio wskazywała oś x, a oś x - w kierunku przeciwnym do pierwotnie wskazywanego przez oś y. Podając macierz obrotu i sposób zamiany osi jako argumenty operacji *remapCoordinateSystem* klasy *SensorManager*, komponent uzyskuje skorygowaną macierz.

6.3.3 Obrót modelu

Skorygowana macierz obrotu bez dalszych modyfikacji może zostać użyta do obrócenia wyświetlanego modelu tak, aby zniwelować obrót urządzenia. Przed każdorazowym wyrenderowaniem modelu na ekranie telefonu, pobierana jest ostatnio odczytana macierz. Przy pomocy funkcji OpenGL model jest obracany zgodnie z wartościami macierzy, a następnie rysowany. Rysunek 6.5 prezentuje kilka przykładów wyświetlania obróconego modelu strzałki.



(a) (b)

(c) (d)

Rysunek 6.5: Przykłady obróconego modelu strzałki: (a) telefon trzymany pionowo, skierowany na północ; (b) telefon leżący poziomo, skierowany na północ; (c) telefon trzymany pionowo, skierowany na zachód; (d) telefon skierowany lekko w dół i na prawo od północy.

Model strzałki domyślnie wskazuje północ. Wskazywany kierunek można jednak dowolnie zmieniać - obrotem modelu może sterować komponent zewnętrzny.

6.4 Wykorzystanie komponentu

Aby użyć *SpaceGraphics* w aplikacji, należy na jednym (bądź więcej) z jej ekranów wykorzystać dostarczany przez ten komponent element interfejsu użytkownika - *SpaceGraphicsView*. Element ten zajmuje się wyświetlaniem grafiki przetwarzanej przez bibliotekę OpenGL. Organizacja klas przez niego wykorzystywanych zostanie omówiona w rozdziale 8.

Aby wyświetlić własny model, należy użyć metody *setModel* obiektu *SpaceGraphicsView*, podając własną konkretną implementację abstrakcyjnej klasy *Model*. Definicja przykładowego modelu wygląda następująco:

```
public class MyModel extends Model {
 public MyModel(float width, float length, float height,
                float[] backgroundColor) {
  super(width, length, height, backgroundColor);
 }

 @Override
 protected float[] generateVertices() {
  reutrn new float[] {
    0.0f * _width,   0.5f * _length,   0.5f * _height,
    0.5f * _width,   0.1f * _length,   0.5f * _height,
    // other vertices...
  };
 }

 @Override
 protected float[] generateColors() {
  return new float[] {
   1.0f, 0.0f, 0.0f, 1.0f,
   1.0f, 0.0f, 0.0f, 1.0f,
   // other vertices...
  };
 }
}
```

SpaceGraphics może znaleźć zastosowanie w wielu aplikacjach wykorzystujących rzeczywistość rozszerzoną lub rzeczywistość wirtualną. Przykłady:

- Wspomniana wcześniej strzałka w grze "podchody".
- Trójwymiarowy model kuli ziemskiej, który można oglądać z różnych stron, odpowiednio obracając telefon.
- Gra polegająca na eksploracji wirtualnych pomieszczeń. Wykorzystanie *SpaceGraphics* polegałoby na stworzeniu dużego modelu pomieszczenia, w środku którego "znajdowałby się" gracz. Obracanie telefonu ukazywałoby kolejne ściany pomieszczenia.

Rozdział 7

Hare and Hounds - przykładowa gra oparta na platformie

Gra *Hare and Hounds* jest przeniesieniem tradycyjnej gry podchody w świat rzeczywistości rozszerzonej. Gracze używają swoich telefonów w celu połączenia się, a także zostawiania i poszukiwania śladów.

7.1 Zasady gry

Zasady gry są następujące:

- W grze bierze udział dwóch graczy: uciekający i goniący.
- Uciekający co 20 sekund zostawia ślad w postaci swojej pozycji:
  - ślad zostawiony w odległości mniejszej niż 20 metrów od poprzedniego śladu jest ignorowany,
  - uciekający wygrywa, gdy zostawi 30 śladów.
- Gdy goniący zbliży się na 10 metrów do śladu pozostawionego przez uciekającego, na ekranie jego telefonu pojawia się strzałka wskazująca kierunek, w jakim powinien się poruszać, aby dotrzeć do kolejnego śladu:
  - strzałka początkowo ma kolor czerwony - w miarę zbliżania się do śladu, płynnie zmienia kolor na zielony,
  - wyświetlaniu strzałki towarzyszy krótki dźwięk - jest on odtwarzany tym częściej, im bliżej śladu znajduje się goniący,
  - po oddaleniu się od śladu na odległość 10 metrów, strzałka znika.
- Goniący wygrywa, gdy dotrze do śladu, który nie prowadzi do kolejnego śladu, tzn. gdy złapie uciekającego zanim ten pozostawi 30 śladów.

Wszystkie wyżej wymienione parametry (częstotliwość zostawiania śladów przez uciekającego, maksymalna odległość od śladu, w jakiej może znajdować się uciekający aby

widzieć strzałkę, itd.) są przechowywane na serwerze w pliku XML[1] i mogą zostać zmienione w dowolnym momencie jego działania.

7.2 Wykorzystanie komponentów fARmework

Hare and Hounds korzysta z podstawowego komponentu *fARmework* (*Core*) w celu realizacji komunikacji pomiędzy klientem a serwerem. Wykorzystuje także komponenty rzeczywistości rozszerzonej: *PositionTracking* do śledzenia pozycji graczy i *SpaceGraphics* do wyświetlania strzałki na ekranie telefonu osoby goniącej. Komponent *PositionTracking* jest również używany podczas przesyłania klientowi listy gier, do których może dołączyć - klient widzi tylko te gry, których gospodarze znajdują się nie więcej niż 200 metrów od niego.

7.3 Współpraca klienta z serwerem

Klient wymienia z serwerem wiadomości takie jak prośba o utworzenie nowej gry lub informacja o wygranej jednego z graczy. Co 20 sekund uciekający informuje serwer o swojej pozycji (zostawia ślad), a goniący - co 10 sekund (w celu sprawdzenia, czy znajduje się w pobliżu śladu).

Serwer przechowuje informacje o utworzonych grach, których gospodarze czekają na współgracza, a także o rozpoczętych rozgrywkach. Podczas rozgrywki ślady pozostawiane przez uciekającego są zapamiętywane na serwerze. Osoba goniąca, oprócz informacji o tym, czy znajduje się w pobliżu śladu i o kierunku, w którym powinna się poruszać, nie otrzymuje żadnych informacji. W szczególności nie otrzymuje informacji o dokładnym położeniu kolejnego śladu - ma to uniemożliwić stworzenie alternatywnej wersji aplikacji klienckiej, która zamiast jedynie naprowadzać goniącego, jawnie zaznaczałaby kolejny ślad na mapie.

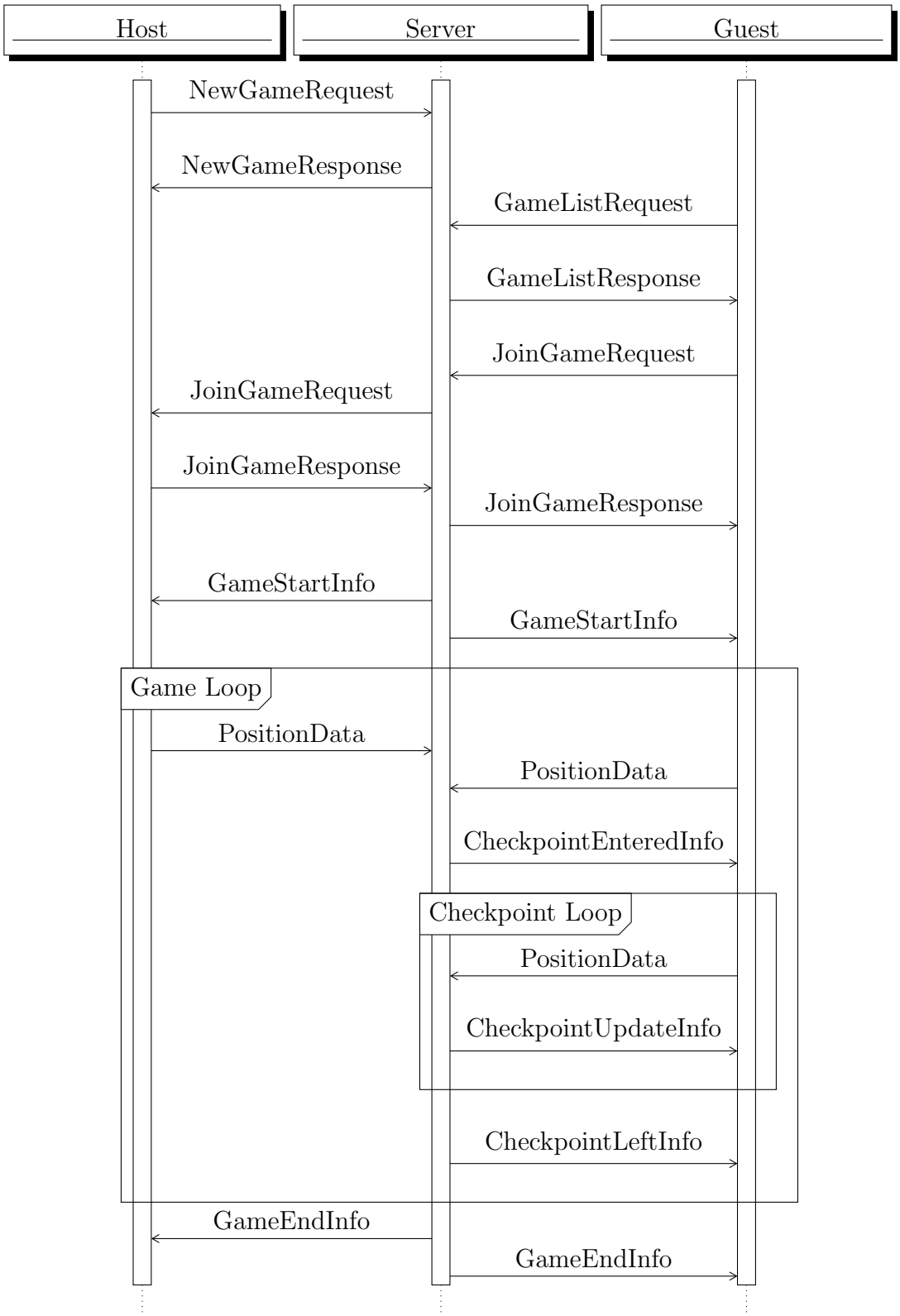
Przebieg komunikacji odbywającej się pomiędzy klientami a serwerem pokazuje Rysunek 7.1.

7.4 Ekrany aplikacji

Rozgrywka po stronie klienckiej *Hare and Hounds* podzielona została na kilka ekranów aplikacji. Są to:

- Ekran startowy - pozwala na połączenie z serwerem jako gospodarz gry lub jej gość; umożliwia dostęp do ekranu opcji.
- Ekran opcji - zawiera ustawienia aplikacji takie jak adres serwera czy nazwa gracza.
- Ekran tworzenia gry - jest wyświetlany podczas oczekiwania na gościa po wyborze połączenia jako gospodarz.

[1] Odczytywaniem pliku XML zajmuje się biblioteka Apache Commons [28].



Rysunek 7.1: Wymiana wiadomości w grze *Hare and Hounds*. Host - osoba uciekająca, Guest - osoba goniąca.

- Ekran listy gier - wyświetla listę dostępnych gier, umożliwia dołączenie do jednej z nich.
- Ekran rozgrywki - jest wyświetlany po rozpoczęciu gry.
- Ekran śladu - pojawia się na urządzeniu osoby goniącej, gdy ta zbliży się do śladu pozostawionego przez osobę uciekającą. Wyświetlna strzałkę wskazującą kierunek ruchu.

Implementacja widoków zostanie szczegółowo omówiona w Rozdziale 8.

7.5 Testowanie aplikacji

Na potrzeby testów *Hare and Hounds* powstał *HareBot* - program udający gracza uciekającego. Automatycznie łączy się on z serwerem i tworzy grę. Po dołączeniu się osoby goniącej i rozpoczęciu gry, *HareBot* symuluje ucieczkę po zadanej trasie. Pozwoliło to na dokładne przetestowanie *Hare and Hounds* w kontrolowanych warunkach. Wiedząc z góry, w których miejscach należy szukać kolejnych śladów uciekającego, możliwe było sprawdzenie:

- dokładności odbiornika GPS,
- poprawności działania operacji udostępnianych przez komponent *PositionTracking*,
- poprawności kierunku wskazywanego przez strzałkę.

Oprócz testów z wykorzystaniem *HareBot*, przeprowadzono również liczne testy, w których udział brały dwa rzeczywiste telefony obsługiwane przez dwie osoby. Miały one na celu sprawdzenie użyteczności wskazówek dostarczanych przez strzałkę, a także ogólnego wrażenia z gry. Testy te wypadły pomyślnie, a na ich podstawie ustalono domyślne parametry wymienione w zasadach gry.

Rozdział 8

Praktyki programistyczne zastosowane w projekcie

Ważnym celem powstania *fARmework* było poszerzenie wiedzy programistycznej jego autorów. O ile informacje na temat specyficznych funkcji systemu Android i działaniu sensorów są przydatne tylko podczas implementacji aplikacji należących do relatywnie wąskiego spektrum, o tyle znajomość uniwersalnych wzorców i technik programowania znajduje zastosowanie w każdym projekcie. Z tego powodu plaforma tworzona była z dbałością o poprawne stosowanie dobrych praktyk programistycznych.

Dodatkową motywacją do takiego podejścia był fakt, że autorzy przed rozpoczęciem prac nad plaformą mieli jedynie powierzchowny kontakt z systemem Android i specyfiką programowania aplikacji mobilnych - nie posiadali więc wiedzy na temat możliwości wykorzystania w tym środowisku praktyk właściwych dużym projektom informatycznym. Projektowanie fARmework zyskało dzięki temu charakter badawczy.

Spośród trendów popularnych obecnie wśród programistów wybrane zostały zarówno praktyki o globalnym zasięgu - dotyczące architektury projektu i sposobu zarządzania nim - jak i pomniejsze wzorce pomocne w realizacji pojedynczych funkcjonalności.

8.1 Zapewnienie elastycznej implementacji

W przypadku dużych systemów o długim czasie realizacji i utrzymania bardzo ważny jest jak najniższy koszt wprowadzania modyfikacji i dostarczania alternatywnych implementacji modułów. Ten sam wymóg dotyczy narzędzi takich jak *fARmework* - programista z nich korzystający powinien móc w łatwy sposób ingerować w ich działanie. Oprogramowanie posiadające tę cechę nazywa się oprogramowaniem elastycznym.

8.1.1 Programowanie pod kątem interfejsu

Podstawową praktyką umożliwiającą wymienność modułów aplikacji jest programowanie pod kątem interfejsu (ang. interface-based programming) [29]. Polega ono na unikaniu używania w procedurach programu konkretnych klas wykonujących dane czynności, a skupianiu się na samych czynnościach i sposobie ich wykorzystania. Zamiast klas, należy korzystać z interfejsów.

Przykładowo komponent *SpaceGraphics* pozwala programiście wyznaczyć kierunek, w którym zwrócony będzie wyświetlany model. Wyświetlaniem zajmuje się obiekt *GraphicsRenderer*, a decydowaniem o kierunku - dostarczany z zewnątrz obiekt implementujący interfejs *IDirectionProvider* (patrz Rysunek 8.1). Rozwiązanie to ma tę zaletę, że *GraphicsRenderer* nie zna typu obiektu wyznaczającego kierunek - wymiana klasy implementującej *IDirectionProvider* jest z punktu widzenia *GraphicsRenderer* transparentna.

8.1.2 Inversion of Control i Dependency Injection

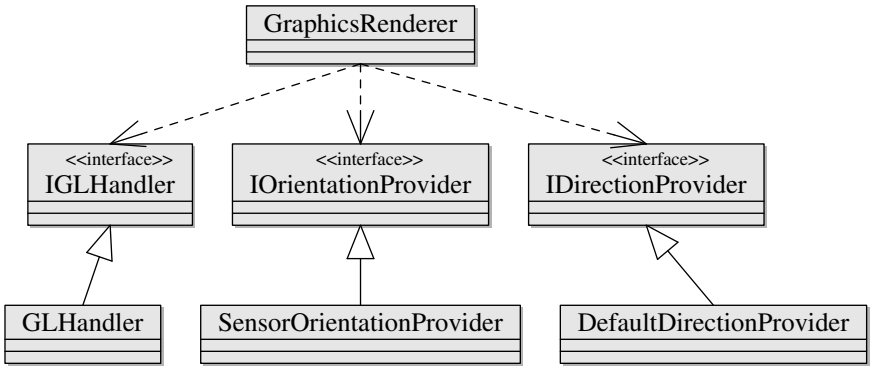
Inversion of Control (IoC, odwrócenie sterowania) jest techniką polegającą na pozbawieniu obiektów możliwości kontrolowania wybranych operacji - kontrola powinna zostać przeniesiona poza te obiekty. Według definicji [30] oznacza to występowanie w projekcie rozwiązań, w których kontrola przekazywana jest w kierunku odwrotnym do stosowanego w programowaniu proceduralnym.

Dependency Injection (DI, wstrzykiwanie zależności) jest implementacją Inversion of Control nakazującą zorganizowanie współpracy pomiędzy obiektami w taki sposób, aby obiekty zależące od instancji obiektów innych klas nie tworzyły tych instancji samodzielnie, a otrzymywały je z zewnątrz. W tym przypadku odwrócone zostaje sterowanie procesesem tworzenia zależności między obiektami - decyzja o tym, która klasa konkretna zostanie użyta jako zależność danego obiektu jest wyniesiona poza ten obiekt.

Najczęściej stosowanym wariantem Dependency Injection jest podawanie wstrzykiwanych obiektów jako parametrów konstruktora klasy zależnej. Na przykład konstruktor wspomnianej wcześniej klasy *GraphicsRenderer* przyjmuje trzy parametry:

- obiekt zajmujący się rysowaniem obrazu przy pomocy funkcji OpenGL - *GLHandler*,
- obiekt dostarczający informacji o orientacji telefonu - *OrientationProvider*,
- wspomniany wcześniej obiekt *DirectionProvider*.

Rezultat zastosowania wstrzykiwania zależności i programowania pod kątem interfejsu przedstawia Rysunek 8.1.



Rysunek 8.1: Zależność obiektu *GraphicsRenderer* od obiektów *GLHandler*, *OrientationProvider* i *DirectionProvider*. Składowe klas i interfejsów zostały pominięte.

8.1.3 Wykorzystane biblioteki

O ile Inversion of Control jest abstrakcyjną zasadą, a interface-based programming niewymagającą stosowania narzędzi praktyką projektową, o tyle wstrzykiwanie zależności jest wspomagane przez biblioteki zewnętrzne. Biblioteki te implementują tzw. DI Container (kontener wstrzykiwania zależności). Pozwala on rejestrować klasy i obiekty, które zostaną użyte w miejscu wystąpienia interfejsów, które są przez nie implementowane.

Istnieje wiele takich bibliotek przeznaczonych dla popularnych środowisk takich jak Java (biblioteki Spring i google-guice) czy .NET (Unity, Castle Windsor, Ninject). Wybór publicznie dostępnych narzędzi wspierających wstrzykiwanie zależności dla systemu Android jest jednak bardzo ograniczony. Podczas projektowania architektury fARmework (marzec 2012 roku) istniała tylko jedna[1] taka biblioteka - roboguice [31]. Posiada ona swój odpowiednik dedykowany "bazowej" platformie Java - google-guice [32]. Ta właśnie para została wybrana jako implementacja DI Container zarówno w warstwie klienckiej (roboguice) jak i serwerowej (google-guice) fARmework.

Obie biblioteki mają wspólną wadę - nie są przezroczyste dla systemu, który z nich korzysta. Wymagają stosowania specyficznych adnotacji w klasach, których zależności powinny zostać wstrzyknięte. Taki wymóg skutkuje występowaniem odniesień do danej biblioteki w wielu rozproszonych fragmentach kodu źródłowego, znacznie utrudniając wymianę tej biblioteki na inną.

8.1.4 Przykład zastosowania roboguice

W celu skonfigurowania biblioteki roboguice tak, aby konstruktor klasy *GraphicsRenderer* otrzymywał argumenty odpowiednich typów, należy:

- Oznaczyć ten konstruktor odpowiednią adnotacją (wspomniana wada biblioteki):

```
@Inject
public GraphicsRenderer(IGLHandler glHandler,
           IOrientationProvider orientationProvider,
           IDirectionProvider directionProvider) {
   // inicjalizacja pól prywatnych
}
```

- Stworzyć tzw. moduł Dependency Injection, tzn. klasę przypisującą interfejsom odpowiednie implementacje:

```
public class SpaceGraphicsModule extends AbstractModule
{
 @Override
 protected void configure() {
  bind(IGLHandler.class).to(GLHandler.class);
  bind(IOrientationProvider.class)
   .to(SensorOrientationProvider.class);
```

[1]Obecnie (styczeń 2013) dostępna jest także biblioteka Dagger: http://square.github.com/dagger/.

```
  bind(IDirectionProvider.class)
   .to(DefaultDirectionProvider.class);
 }
}
```

- Zarejestrować moduł w zasobach aplikacji (w systemie Android zasoby przechowywane są w postaci plików XML). W tym celu należy utworzyć listę o nazwie "roboguice_modules" i dodać do niej pełną nazwę klasy modułu.

```
<resources>
 <string-array name="roboguice_modules">
     <item>[nazwa pakietu].SpaceGraphicsModule</item>
     <!-- inne moduły -->
 </string-array>
</resources>
```

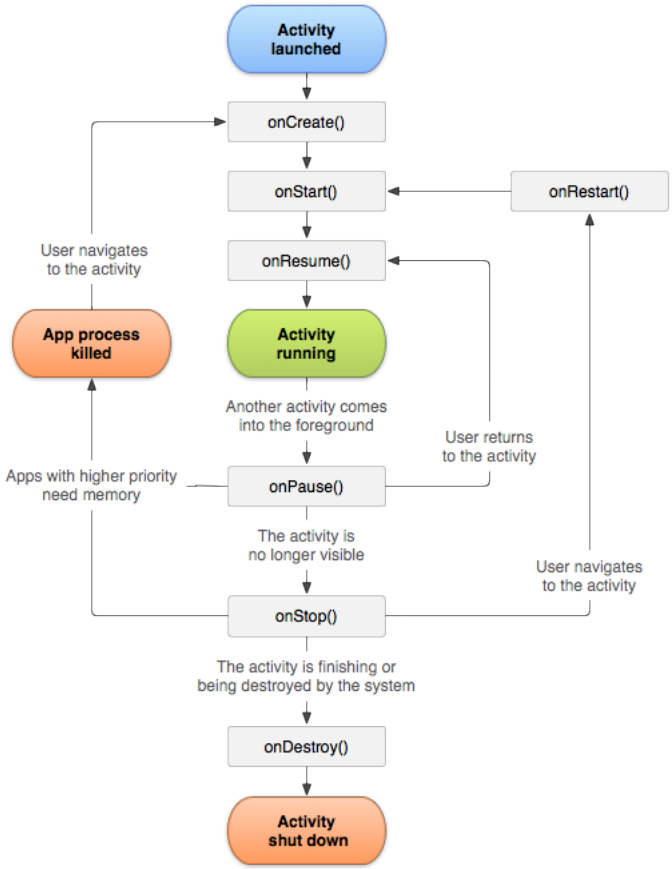
Po wykonaniu tych czynności za każdym razem, gdy biblioteka roboguice konstruować będzie obiekt klasy *GraphicsRenderer* (jako zależność obiektu innej klasy), jako parametry jego konstruktora użyte zostaną obiekty klas przypisanych do kolejnych interfejsów. Jak widać, przypisanie klasy do interfejsu odbywa się tylko w jednym miejscu - odseparowanym od reszty kodu module Dependency Injection. Taka organizacja umożliwia zmianę przypisania bez ingerencji w kod źródłowy klas realizujących funkcjonalności aplikacji, co świadczy o odpowiedniej elastyczności implementacji.

8.2 Zasada jednej odpowiedzialności a system Android

System Android udostępnia podstawową klasę umożliwiającą kontrolowanie treści wyświetlanych na ekranie telefonu - *Activity* (aktywność). Reprezentuje ona pojedynczy ekran aplikacji lub pojedynczą czynność, którą może wykonać użytkownik. Rysunek 8.2 przedstawia schemat cyklu życia aktywności.

Aktywność realizuje szereg zadań [33]:

- tworzy na ekranie okno (wypełniające cały dostepny obszar lub tylko jego część), w którym wyświetla elementy interfejsu użytkownika (metoda *onCreate*) i wypełnia je danymi,
- zarządza używanymi przez siebie zasobami, takimi jak kamera telefonu czy odbiornik GPS, jak również udostępnia je innym obiektom,
- zanim zostanie przesłonięta przez inną aktywność, zapisuje swój stan (metoda *onPause*) i przywraca go, zanim powróci na ekran urządzenia (metoda *onResume*).
- przechowuje kontekst aplikacji, tj. informacje o stanie aplikacji i środowisku, w którym jest wykonywana,



Rysunek 8.2: Cykl życia obiektu *Activity* [33].

Takie rozwiązanie jest poniekąd wygodne (cała implementacja prostej aplikacji może znaleźć się w pojedynczej klasie *Activity*), jednak stanowi naruszenie najważniejszej zasady programowania obiektowego: zasady jednej odpowiedzialności (ang. Single Responsibility Principle, SRP [34]). Z punktu widzenia architektury Model-View-Controller (MVC), aktywność pełni rolę kontrolera, jak również częściowo realizuje rolę widoku. Aby obiekty Activity nie stały się "przeładowane" funkcjonalnością, należy tę odpowiedzialność rozdysponować pomiędzy inne obiekty. W tym celu część kliencka komponentu *Utils* zawiera klasy wspomagające implementację wzorca architektonicznego Model-View-ViewModel.

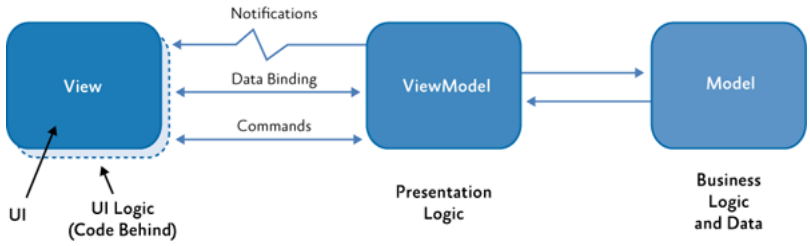
8.2.1 Model-View-ViewModel

Wzorzec Model-View-ViewModel (MVVM) jest oparty na wzorcu MVC, wprowadzającym podział aplikacji na trzy warstwy:

- *Model* (model) - przechowuje dane należące do dziedziny (ang. domain) aplikacji, a także realizuje operacje na tych danych (logika biznesowa, ang. business logic).
- *View* (widok) - służy do prezentacji danych pobranych z modelu - reprezentuje interfejs graficzny aplikacji.

- *Controller* (kontroler) - kontroluje przepływ danych z modelu do widoku. Na podstawie danych pobranych z widoku zleca modelowi wykonanie odpowiedniej operacji i wybiera widok, który zostanie wyświetlony po wykonaniu tej operacji.

Według wzorca MVVM rola modelu i widoku pozostaje niezmieniona, natomiast kontroler zostaje zastąpiony przez *ViewModel*. *ViewModel* jest całkowicie oseparowany od widoku - to odróżnia go od klasy *Activity*. Jego rolą jest udostępnienie danych i operacji, za pomocą których widok może prezentować model i manipulować nim. Schemat interakcji pomiędzy warstwami wzorca MVVM obrazuje Rysunek 8.3.



Rysunek 8.3: Organizacja wzorca Model-View-ViewModel [35].

Ważnym elementem wzorca, ułatwiającym widokowi pobieranie treści do wyświetlenia, jest wiązanie danych (ang. data binding). Pozwala ono na automatyczne związanie danych pomiędzy obiektem *ViewModel* a danym widokiem - jeśli dana operacja zmieni wartość danych udostępnianych przez *ViewModel*, zmiana ta zostanie samoczynnie odzwierciedlona w widoku.

Realizacją wiązania danych w *Hare and Hounds* zajmuje się zewnętrzna biblioteka android-binding [36]. Wymaga ona stosowania w obiektach *ViewModel* dostarczanych przez siebie typów danych (np. *StringObservable* zamiast *String*), jak również dostosowania kodu widoków. W przypadku bibliotek tego rodzaju jest to jednak zrozumiałe.

Biblioteka ta została wybrana z powodu braku alternatywy. W chwili obecnej android-binding w dalszym ciągu wydaje się być jedynym narzędziem realizującym wiązanie danych dla systemu Android. Prawdopodobnie wynika to stąd, że wzorzec MVVM powstał w firmie Microsoft - na potrzeby technologii Silverlight i WPF - i nie jest szeroko znany i aprobowany wśród programistów innych platform.

8.2.2 Przykład wykorzystania android-binding

Przykładem zastosowania biblioteki android-binding jest widok opcji aplikacji klienckiej *Hare and Hounds*. Zawiera on pola tekstowe umożliwiające wprowadzenie takich wartości jak nazwa użytkownika (*UserName*), a także przycisk zapisujący ustawienia (*Save*).

Modelem tego widoku jest obiekt *SettingsProvider* realizujący dostęp do zasobów przechowujących ustawienia aplikacji. Klasa *ViewModel* przeznaczona dla tego widoku jest więc zależna od interfejsu *ISettingsProvider*, a także zawiera pole *UserName* i komendę *Save*:

```
public class OptionsViewModel extends ViewModel
{
```

```
 private final ISettingsProvider _settingsProvider;

 public StringObservable UserName = new StringObservable();
 // inne opcje

 public Command Save = new Command() {
  @Override
  public void Invoke(View arg0, Object... arg1) {
   _settingsProvider.setUserName(UserName.get());
  }
 }
}
```

Elementy interfejsu użytkownika (ang. layout) w systemie Android definiuje się w pliku XML. Definicja ekranu opcji, zawierającego pola tekstowe i przycisk zapisu, wygląda następująco:

```
<LinearLayout ...>
    <EditText binding:text="UserName" ... />
    <!-- inne opcje -->

    <Button binding:onClick="Save" ... />
</LinearLayout>
```

Atrybuty "binding" wskazują pola obiektu *ViewModel*, z którymi powiązane są dane elementy. Po związaniu powyższego ekranu z obiektem klasy *OptionsViewModel* i wpisaniu przez użytkownika wartości w pole tekstowe *UserName*, wartość ta zostanie automatycznie nadana polu *UserName* tego obiektu, a po naciśnięciu przycisku *Save* wywołana zostanie jego komenda *Save*.

Odpowiedzialność związania interfejsu danego ekranu z odpowiednim obiektem *ViewModel* spoczywa na aktywności reprezentującej ten ekran. Pozwala to całkowicie odseparować *ViewModel* od widoku. Gdyby wystąpiła potrzeba stworzenia alternatywnej wersji aplikacji przeznaczonej na komputery osobiste i korzystającej z klasycznych bibliotek graficznych Java, takich jak AWT czy Swing, wystarczyłoby jedynie wymienić implementację interfejsu graficznego - warstwy *ViewModel* i *Model* mogłyby pozostać niezmienione.

8.2.3 Zarządzanie kontekstem aplikacji

Wśród wymienionych wcześniej zadań klasy *Activity* znajduje się zarządzanie kontekstem aplikacji. Aby uwolnić ją od tej odpowiedzialności, komponent *Utils* zawiera klasę *ContextManager*, która udostępnia operacje związane z kontekstem, takie jak aktywowanie *Activity* lub wyświetlenie okna dialogowego. Co więcej, *ContextManager* przechowuje stos tworzonych aktywności, co umożliwia zakończenie aplikacji w dowolnym momencie (na przykład w reakcji na błąd krytyczny aplikacji) poprzez zakończenie wszystkich aktywności ze stosu[2].

[2]System Android domyślnie nie udostępnia takiej operacji.

Po zastosowaniu wzorca MVVM i stworzeniu klasy *ContextManager*, jedyną odpowiedzialnością jaka pozostała obiektom *Activity* w aplikacjach klienckich opartych na *fARmework* jest pośredniczenie pomiędzy warstwą *View* a *ViewModel* (związanie interfejsu graficznego z obiektem *ViewModel*).

8.3 Wzorce projektowe

Oprócz wzorców architektonicznych, wpływających na kształt całego projektu, w *fARmework* znalazły zastosowanie także wzorce projektowe, rozwiązujące pomniejsze problemy współpracy obiektów. Podczas projektowania struktury niektórych fragmentów platformy wykorzystano następujące wzorce:

- Most (ang. Bridge) - wzorzec oddzielający abstrakcję klasy od jej implementacji [37],
- Fabryka (ang. Factory) - wzorzec pozwalający odseparować miejsce utworzenia obiektu od miejsca jego wykorzystania [38],
- Strategia (ang. Strategy) - wzorzec umożliwiający dynamiczną zmianę zachowania programu w zależności od potrzeb [12],
- Metoda szablonowa (ang. Template method) - wzorzec pozwalający wyodrębnić część wspólną danej operacji, przy zachowaniu możliwości dostarczenia różnych implementacji specyficznych kroków tej operacji [39].

Poniżej wyjaśniona zostanie implementacja wzorca Strategii w warstwie klienckiej komponentu *Core*:

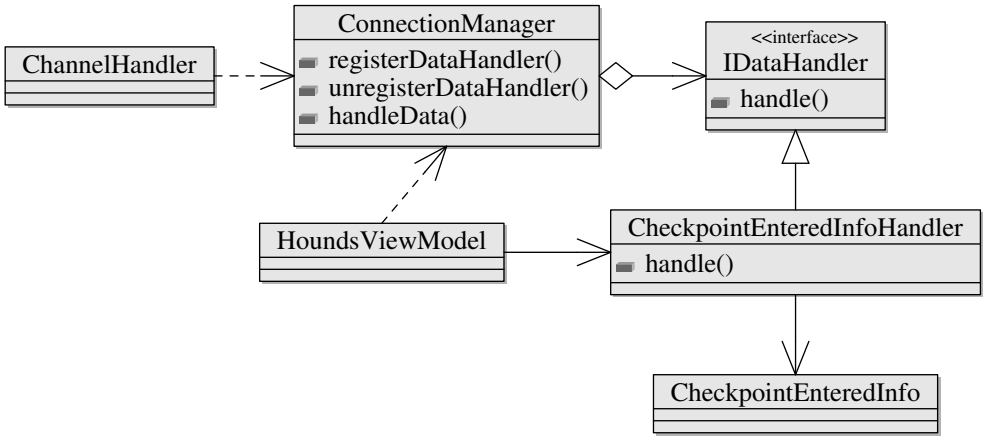
8.3.1 Wykorzystanie wzorca Strategii

Komunikacja pomiędzy klientem a serwerem opiera się na wiadomościach wysyłanych przez nadawcę, na które reaguje odbiorca. Jak zostało wcześniej opisane, sposób reakcji określają procedury obsługi wiadomości (obiekty *DataHandler*). Jako że sposób reagowania jest zazwyczaj zależny od stanu, w jakim znajduje się odbiorca, zarówno strona kliencka jak i serwerowa powinny móc dynamicznie wymieniać używane procedury.

W tym celu zastosowano wzorzec Strategii. Polega on na określeniu abstrakcyjnego interfejsu danej operacji (w tym przypadku metody *handle*) i dynamicznym podejmowaniu decycji o tym, która implementacja tego interfejsu zostanie użyta w danym momencie. Dynamika oznacza tutaj fakt, że decyzja podejmowana jest w trakcie wykonania programu, a nie w trakcie jego kompilacji.

Przykładem obrazującym działanie wzorca w grze *Hare and Hounds* jest przebieg rozgrywki gracza goniącego. Rozgrywkę tę reprezentuje klasa *HoundsViewModel*. Między innymi określa ona sposób reakcji na wiadomość *CheckpointEnteredInfo*, mówiącą, że gracz znalazł się w pobliżu śladu pozostawionego przez osobę uciekającą. Używając metody *registerDataHandler* klasy *ConnectionManager*, *HoundsViewModel* rejestruje obiekt *CheckpointEnteredInfoHandler* implementujący procedurę wyświetlającą strzałkę wskazującą goniącemu kierunek ruchu. Kiedy *ChannelHandler* przekaże metodzie *handleData* obiektu *ConnectionManager* wiadomość *CheckpointEnteredInfo*, uruchomiona zostanie właśnie ta procedura.

Zależności pomiędzy wymienionymi klasami przestawia Rysunek 8.4.



Rysunek 8.4: Uproszczony schemat implementacji wzorca strategii po stronie klienckiej *Hare and Hounds*.

8.4 Test-Driven Development (TDD)

Test-Driven Development (TDD) to jedna z metodyk stosowanych podczas tworzenia oprogramowania zgodnie z zasadami programowania zwinnego (ang. Agile Software Development [40]). Polega ona na pisaniu testów jednostkowych testujących daną funkcjonalność zanim ta funkcjonalność zostanie zaimplementowana. Programowanie z wykorzystaniem tej techniki składa się z trzech etapów:

1. Red (czerwony) - stworzenie szkieletu funkcjonalności i napisanie testującego ją testu jednostkowego (lub zestawu testów) - który na tym etapie kończy się niepowodzeniem.

2. Green (zielony) - implementacja funkcjonalności tak, aby napisany test (lub zestaw testów) kończył się powodzeniem.

3. Refactor (refaktoryzacja) - ewentualna poprawa jakości kodu powstałego w punkcie 2. Testy jednostkowe nadal powinny kończyć się powodzeniem.

Stosowanie metodyki TDD nie przynosi znacznych korzyści podczas tworzenia trywialnych funkcjonalności, z którymi wiąże się bardzo niskie ryzyko popełnienia błędu. Z drugiej strony, jest bardzo czasochłonne podczas implementacji skomplikowanych rozwiązań. Z tego względu, zgodnie z zasadami Test-Driven Development powstała tylko ta część *fARmework*, na której implementację składa się co najwyżej kilka klas. Są to głównie moduły wykonujące obliczenia lub implementujące algorytmy.

Testy jednostkowe w *fARmework* powstały przy użyciu platformy JUnit [41] - jest to zdecydowanie najbardziej popularne narzędzie tego typu dedykowane środowisku Java. Do tworzenia zaślepek[3] obiektów testowych wykorzystano bibliotekę mockito [42].

[3]Zaślepka (ang. mock) - kontrolowany obiekt danej klasy, przekazywany testowanej metodzie w celu weryfikacji jej współpracy z obiektami tej klasy.

8.4.1 Przykład zastosowania

Przykładem zastosowania TDD w *fARmework* może być proces implementacji klasy obliczającej azymut pomiędzy dwoma punktami na kuli ziemskiej (klasa *DirectionCalculator* modułu *PositionTracking.Java*). W tym przypadku kolejne etapy przebiegały następująco:

1. Red:
   (a) stworzenie szkieletu klasy *DirectionCalculator* i klasy testującej,
   (b) wybranie zestawu par punktów na kuli ziemskiej (na przykład miast) i znalezienie rzeczywistych azymutów pomiędzy każdą z tych par - do tego celu posłużył serwis Time and Date [43],
   (c) napisanie testów sprawdzających poprawność działania klasy *DirectionCalculator* dla każdej z tych par.
2. Green:
   (a) znalezienie wzoru na obliczenie azymutu [20],
   (b) implementacja wzoru w klasie *DirectionCalculator*,
   (c) uruchomienie testów jednostkowych i ewentualna poprawa implementacji.
3. Refactor - poprawa czytelności kodu realizującego wzór.

Taka organizacja pracy daje gwarancję poprawnej implementacji klasy *DirectionCalculator*. Jej dodatkową zaletą jest ułatwienie testowania regresyjnego - jeśli w przyszłości podjęta zostanie próba zwiększenia dokładności przeprowadzanych obliczeń, to ewentualne niepowodzenie istniejących testów jednostkowych zasygnalizuje błąd w zmienionej implementacji.

8.5 Zarządzanie projektem

Ze względu na to, że nad *fARmework* pracowały dwie osoby, możliwe było praktyczne wykorzystanie podstawowych narzędzi służących do zarządzania projektem informatycznym. Narzędzia takie umożliwiają monitorowanie stanu projektu i zmian w nim zachodzących. Podczas prac nad plaformą używane były:

- aplikacja internetowa służąca do monitorowania postępów prac nad projektem,
- system kontroli wersji.

8.5.1 Monitorowanie postępów prac nad projektem

Jako narzędzie służące do śledzenia postępów prac na projektem wybrany został serwis TargetProcess [44]. Pozwala on na definiowanie zadań, przypisywanie im priorytetów i wyznaczanie osób, które je wykonają. Jego największą zaletą jest możliwość prezentacji wszystkich najważniejszych informacji o stanie projektu na jednym ekranie.

8.5.2 Kontrola wersji

Podczas pracy nad projektem trwającym więcej niż kilka tygodni zalecane jest używanie systemu kontroli wersji (ang. version control system, VCS). Taki system przechowuje wszystkie pliki projektu wraz z historią ich zmian - oznacza to, że w dowolnym momencie możliwe jest przywrócenie nadpisanej wersji dowolnego pliku. Co więcej, możliwe jest prześledzenie wszystkich zmian danego pliku i uzyskanie informacji o autorach tych zmian. Dostępnych jest wiele popularnych systemów kontroli wersji, na przykład Subversion, Git czy Mercurial. Na potrzeby *fARmework* wybrany został ostatni z wymienionych [45]. Jako serwis wspierający użycie tego systemu wykorzystano Google Code [46]. Jako klienta Mercurial użyto programu TortoiseHG[4] [47].

Mercurial jest systemem zdecentralizowanym (ang. decentralized version control system, DVCS), co oznacza, że osoby biorące udział w projekcie nie muszą być stale połączone z siecią w celu zachowania synchronizacji. Umożliwia to jednoczesną pracę wielu osób nad różnymi wersjami tych samych fragmentów programu. Ponadto system ten implementuje mechanizm gałęzi. Ścieżkę pracy nad projektem można rozgałęziać i scalać, dzięki czemu możliwe jest tworzenie tymczasowych odgałęzień przeznaczonych na pojedyncze zadania, przy zachowaniu w pełni działającej wersji projektu w gałęzi głównej. Po zaimplementowaniu i przetestowaniu zadanej funkcjonalności wystarczy scalić gałąź zadania z gałęzią główną.

8.5.3 Przykład wykorzystania narzędzi

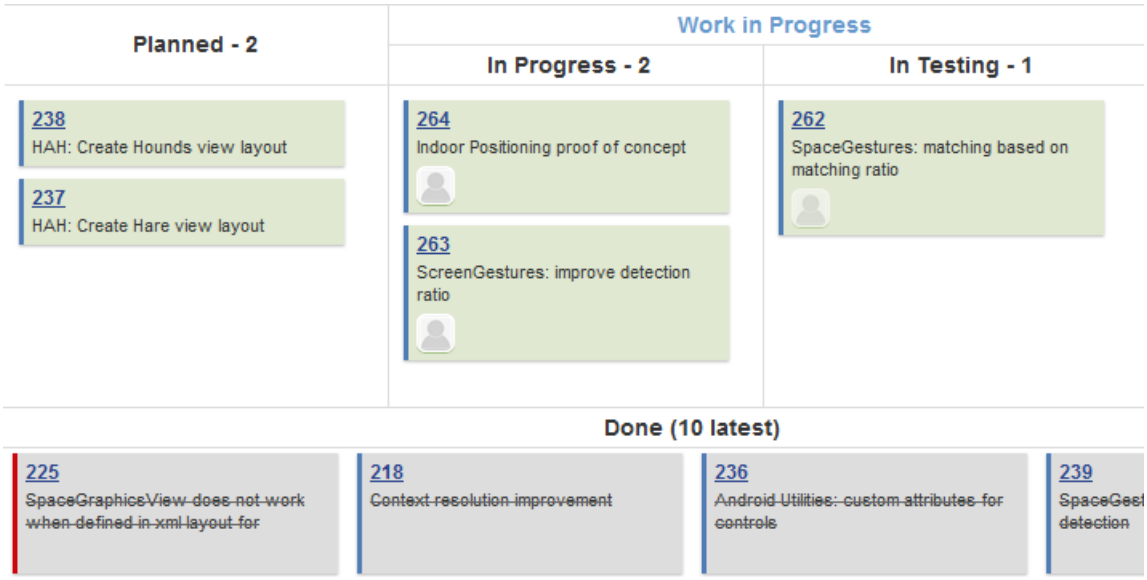
Jako przykład zastosowania opisanych narzędzi przedstawiona zostanie organizacja pracy podczas równoległego wykonywania dwóch zadań:

- poprawa skuteczności mechanizmu rozpoznawania getsów w module ScreenGestures (zadanie nr 263),
- stworzenie szkieletu modułu PositionTracking.WiFi (zadanie nr 264).

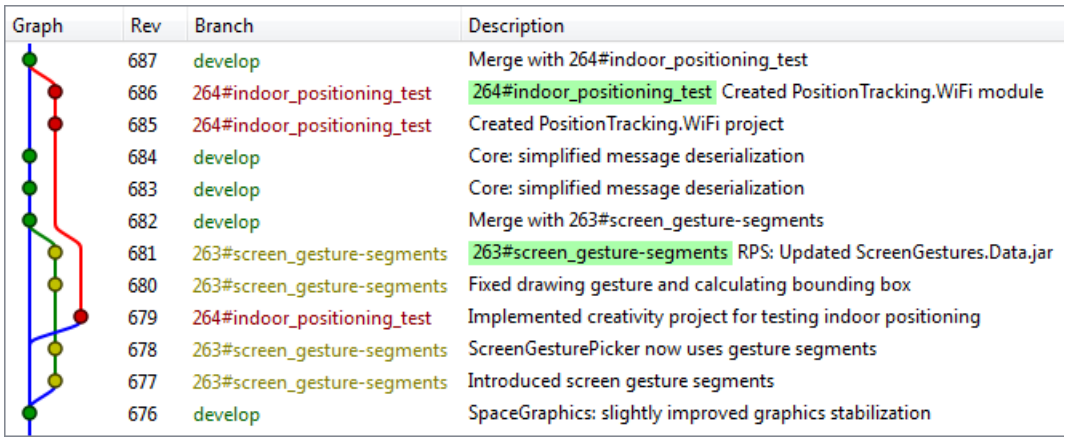
Proces definiowania i implementacji powyższych funkcjonalności oraz ich integracji z projektem przebiegał następująco:

1. Wspólne określenie wymagań co do zadań.
2. Utworzenie zadań w programie TargetProcess (Rysunek 8.5) i nadanie im statusu “In Progress” (“w toku”).
3. Niezależne utworzenie odgałęzień projektu przeznaczonych na zadania (Rysunek 8.6).
4. Niezależna stopniowa implementacja zadań.
5. Niezależne scalenie odgałęzień z główną gałęzią projektu (Rysunek 8.6).

[4]Tortoise* - popularna rodzina klientów systemów kontroli wersji, do której należą m.in. TortoiseHG, TortoiseSVN (system Subversion), TortoiseGit (system Git).



Rysunek 8.5: Zrzut ekranu aplikacji TargetProcess. *Planned* - zadania zaplanowane; *In Progress* - zadania rozpoczęte; *In Testing* - zadania ukończone wymagające weryfikacji; *Done* - zadania przetestowane z wynikiem pozytywnym.



Rysunek 8.6: Zrzut ekranu aplikacji TortoiseHG pokazujący odgałęzienia projektu: *Develop* - gałąź główna; *263#screen_gesture-segments* - gałąź zadania nr 263; *264#indoor_positioning_test* - gałąź zadania nr 264. Kolejne wiersze oznaczają kolejne porcje kodu dodawane do projektu.

Rozdział 9

Podsumowanie

Celem niniejszej pracy inżynierskiej było zaprojektowanie i zaimplementowanie platformy wspomagającej tworzenie wieloosobowych aplikacji wykorzystujących koncepcję rzeczywistości rozszerzonej. Wszystkie przestawione we Wstępie cele (patrz rodział 1) zostały zrealizowane:

1. Wszystkie komponenty platformy zostały przetestowane na smartfonie Sony Ericsson Xperia X8. Do niektórych testów użyto także modeli Samsung Galaxy Gio i Samsung Galaxy S III.

2. Aplikacja serwerowa z powodzeniem uruchamiana była w kilku środowiskach, w tym na systemach Windows i Ubuntu, zainstalowanych na komputerach przenośnch, oraz systemie Debian zainstalowanym na zdalnej maszynie wirtualnej HyperVM[1].

3. Komponenty rzeczywistości rozszerzonej nie narzucają typu aplikacji, w której zostaną wykorzystane - komponent *PositionTracking* dostarcza uniwersalnych metod odczytu pozycji telefonu, a komponent *SpaceGraphics* pozwala wyświetlić dowolny model.

4. Wszelkie obliczenia i przechowywanie danych odbywa się po stronie serwera. Aplikacje klienckie wykonują jedynie operacje specyficzne dla systemu Android.

5. Komponenty rzeczywistości rozszerzonej nie narzucają wykorzystania architektury klient-serwer. Ich struktura została jednak zorganizowana w taki sposób, aby ich wykorzystanie w tej architekturze było możliwe i bezproblemowe.

6. Próba wykorzystania standardowych technik programistycznych powiodła się - struktura platformy Android, a także dostępne narzędzia (mimo bardzo wąskiego ich wyboru) umożliwiają zorganizowanie projektu w sposób właściwy dużym projektom informatycznym. Podczas projektowania i implementowania platformy stosowano wzorce architektoniczne i projektowe, a także inne dobre praktyki programistyczne.

[1] http://lxcenter.org/software/hypervm

9.1 Ocena jakości platformy i perspektywy rozwoju

Platforma *fARmework* wydaje się być wykonana prawidłowo, zarówno z punktu widzenia korzystającego z niej programisty jak i końcowego użytkownika aplikacji wykonanych z jej użyciem. Dostarczone komponenty współpracują poprawnie - podczas testów szyny komunikacyjnej nie swierdzono ani jednego przypadku błędnego nadania lub odbioru wiadomości, a podczas testowania przykładowej aplikacji bardzo rzadko zdarzały się niespodziewane błędy (które niezwłocznie eliminowano). Co więcej, dzięki zastosowaniu standardowych wzorców zwiększających elastyczność oprogramowania, programista ma możliwość samodzielnego rozszerzania *fARmework* i modyfikowania jego działania.

Doświadczenia użytkownika końcowego również powinny być pozytywne. Dokładność metod odczytu lokalizacji i orientacji telefonu jest na tyle duża, że możliwe jest oparcie na nich funkcjonalnych aplikacji wykorzystujących rzeczywistośc rozszerzoną. Miłym dowodem powyższego stwierdzenia jest entuzjastyczne przyjęcie gry *Hare and Hounds* wśród osób, które miały z nią styczność.

Platforma prawdopodobnie będzie rozwijana - stworzone zostaną nowe komponenty rzeczywistości rozszerzonej, a działanie istniejących zostanie usprawnione. Niewykluczone też, że pojawią się wersje *fARmework* dedykowane systemom innym niż Android, np. Windows Phone lub iOS. Postępy prac nad platformą można śledzić pod adresem: http://code.google.com/p/hare-and-hounds/ - kod źródłowy został opublikowany jako oprogramowanie open-source na lincencji GNU GPL 3.

9.2 Wnioski

Przebieg i ekekt pracy nad *fARmework* prowadzi do wniosku mówiącego, że dokładność sensorów - szczególnie odbiornika GPS - nie jest jeszcze na tyle duża, aby na podstawie ich odczytów możliwe było tworzenie aplikacji działających w konkretnych miejscach, w bardzo małym promieniu wokół użytkownika. Dopiero połączone działanie sensorów takich jak żyroskop, akcelerometr, magnetometr, obiornik GPS i barometr może dawać możliwość dokładnego zlokalizowania telefonu w przestrzeni. Niestety, na chwilę obecną, w pełen zestaw sensorów wyposażone są jedynie nieliczne urządzenia z najwyższej półki.

Z drugiej strony, procesowi projektowania i powstawania fARmework nieustannie towarzyszyło poczucie ogromnych możliwości, jakie niesie ze sobą wykorzystanie pełnego potencjału smartfonów. Wykorzystanie zaawansowanych technologicznie sensorów jest w zasięgu możliwości każdego programisty. Śledzenie pozycji użytkownika, wyświetlanie informacji w zależności od jego otoczenia, odczytywanie jego gestów lub obrazu z kamery telefonu nie jest karkołomnym zadaniem wymagającym głębokiej wiedzy i znajomości zaawansownych algorytmów. Aby podjąć się stworzenia aplikacji realizującej którąś z tych funkcjonalności, wystarczy poświęcić kilka godzin na przestudiowanie artykułów na jej temat i zapoznać się z gotowymi przykładami zamieszonymi w dokumentacji systemu Android.

Tworzenie kolejnych kompomentów *fARmework* przebiegało w takim właśnie trybie. Ich opracowanie było źródłem cennej wiedzy zarówno programistycznej, jak i dotyczącej innych dziedzin, takich jak działanie poszczególnych sensorów czy podstawy geolokalizacji.

Bibliografia

[1] *7 things you should know about Augmented Reality.* EduCause. dostęp: styczeń 2013. w: http://net.educause.edu/ir/library/pdf/ELI7007.pdf

[2] *What is Augmented Reality (AR): Augmented Reality Defined, iPhone Augmented Reality Apps and Games and More.* Digital Trends. dostęp: styczeń 2013. w: http://www.digitaltrends.com/mobile/what-is-augmented-reality-iphone-apps-games-flash-yelp-android-ar-software-and-more/

[3] Márquez, J. *An Introduction to Virtual Reality.* dostęp: styczeń 2013. w: http://web.mit.edu/16.459/www/VR1.pdf

[4] *40 Best Augmented Reality iPhone Apps.* iPhoneNess. dostęp: styczeń 2013. w: http://www.iphoneness.com/iphone-apps/best-augmented-reality-iphone-applications/

[5] *Gartner Says Worldwide Sales of Mobile Phones Declined 3 Percent in Third Quarter of 2012; Smartphone Sales Increased 47 Percent.* Gartner. dostęp: styczeń 2013. w: http://www.gartner.com/it/page.jsp?id=2237315

[6] *Number of available Android applications.* AppBrain. dostęp: styczeń 2013. w: http://www.appbrain.com/stats/number-of-android-apps

[7] Netty project. dostęp: grudzień 2012. w: https://netty.io/

[8] google-gson. dostęp: grudzień 2012. w: http://code.google.com/p/google-gson/

[9] *Java Object Serialization Specification.* Oracle. dostęp: styczeń 2013. w: http://docs.oracle.com/javase/7/docs/platform/serialization/spec/serialTOC.html

[10] *Extensible Markup Language (XML).* dostęp: styczeń 2013. w: http://www.w3.org/XML/

[11] JSON. dostęp: grudzień 2012. w: http://www.json.org/

[12] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (2010). *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku.* Wydawnictwo HELION. ISBN 978-83-246-2662-5. **321-333**

[13] Cejas, Fernando. (28.10.2010). *Android Location Providers - gps, network, passive.* dostęp: styczeń 2013. w: http://www.android10.org/index.php/articleslocationmaps/226-android-location-providers-gps-network-passive/

[14] Vaughan-Nichol, Steven J. *How Google–and everyone else–gets Wi-Fi location data.* dostęp: styczeń 2013. w: http://www.zdnet.com/blog/networking/how-google-and-everyone-else-gets-wi-fi-location-data/1664/

[15] *Wi-Fi Location-Based Services—Design and Deployment Considerations.* Cisco Systems. dostęp: styczeń 2013. w: https://learningnetwork.cisco.com/servlet/JiveServlet/previewBody/3418-102-1-9429/ccmigration_09186a008073ce3a.pdf

[16] *WifiManager.* Android Developers. dostęp: styczeń 2013. w: http://developer.android.com/reference/android/net/wifi/WifiManager.html

[17] Circle Intersection Solver. dostęp: styczeń 2013. w: http://mysite.verizon.net/res148h4j/javascript/script_circle_intersections.html#the source code

[18] *Circumscribed circle.* dostęp: styczeń 2013. w: http://en.wikipedia.org/wiki/Circumcenter#Cartesian_coordinates

[19] *Astronomiczne podstawy geografii / Odległości.* Wikibooks. dostęp: styczeń 2013. w: http://pl.wikibooks.org/wiki/Astronomiczne_podstawy_geografii/Odległości

[20] *Calculate distance, bearing and more between Latitude/Longitude points.* Movable Type Scripts. dostęp: styczeń 2013. w: http://www.movable-type.co.uk/scripts/latlong.html

[21] Hirzer, Martin. (2008). *Marker Detection for Augmented Reality Applications.* **1-3**

[22] *OpenGL ES - The Standard for Embedded Accelerated 3D Graphics.* dostęp: styczeń 2013. w: http://www.khronos.org/opengles/

[23] *Advanced Shaders.* dostęp: styczeń 2013. w: http://www.techsoft3d.com/getting-started/hoops-visualize/advanced-shaders

[24] Green S., Cebenoyan C. *High Dynamic Range Rendering on the GeForce 6800.* dostęp: styczeń 2013. w: http://download.nvidia.com/developer/presentations/2004/6800_Leagues/6800_Leagues_HDR.pdf

[25] *Calculating a Surface Normal.* dostęp: styczeń 2013. w: http://www.opengl.org/wiki/Calculating_a_Surface_Normal

[26] *SensorEvent.* Android Developers. dostęp: styczeń 2013. w: http://developer.android.com/reference/android/hardware/SensorEvent.html

[27] *SensorManager.* Android Developers. dostęp: styczeń 2013. w: http://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrix(float[], float[], float[], float[])

[28] *Apache Commons. Configuration.* Apache Commons. dostęp: grudzień 2013. w: http://commons.apache.org/configuration/

[29] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (2010). *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku.* Wydawnictwo HELION. ISBN 978-83-246-2662-5. **31**

[30] *Inversion of Control: Overview with Examples.* CodeProject. dostęp: styczeń 2013. w: http://www.codeproject.com/Articles/380748/Inversion-of-Control-Overview-with-Examples

[31] roboguice. dostęp: grudzień 2012. w: http://code.google.com/p/roboguice/

[32] *google-guice.* dostęp: grudzień 2012. w: http://code.google.com/p/google-guice/

[33] *Activity.* Android Developers. dostęp: grudzień 2012. w: http://developer.android.com/reference/android/app/Activity.html

[34] Martin, Robert C. (2002). *Agile Software Development, Principles, Patterns, and Practices.* Prentice Hall. ISBN 0-13-597444-5.

[35] *Implementing the MVVM Pattern.* MSDN. dostęp: grudzień 2012. w: http://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx

[36] android-binding. dostęp: grudzień 2012. w: http://code.google.com/p/android-binding/

[37] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (2010). *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku.* Wydawnictwo HELION. ISBN 978-83-246-2662-5. **181-190**

[38] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (2010). *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku.* Wydawnictwo HELION. ISBN 978-83-246-2662-5. **101-109**

[39] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (2010). *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku.* Wydawnictwo HELION. ISBN 978-83-246-2662-5. **264-268**

[40] *Agile Software Development: A gentle introduction.* dostęp: styczeń 2013. w: http://www.agile-process.org/

[41] JUnit. dostęp: grudzień 2012. w: http://www.junit.org/

[42] mockito. dostęp: grudzień 2012. w: http://code.google.com/p/mockito/

[43] Time and Date. dostęp: styczeń 2013. w: http://www.timeanddate.com/worldclock/distance.html

[44] TargetProcess. dostęp: grudzień 2012. w: http://www.targetprocess.com/

[45] mercurial. dostęp: grudzień 2012. w: http://mercurial.selenic.com/

[46] Google Code. dostęp: grudzień 2012. w: http://code.google.com/intl/pl/

[47] TortoiseHG. dostęp: styczeń 2013. w: http://tortoisehg.bitbucket.org/pl/

Oświadczenie

Oświadczam, że Pracę Dyplomową pod tytułem *"Platforma aplikacji klient-serwer wykorzystujących rzeczywistość rozszerzoną dla systemu Android"*, którą kierował dr inż. Jakub Koperwas, wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Michał Aniserowicz