POZNAN UNIVERSITY OF TECHNOLOGY

Patryk Dąbrowski 100584
Aleksander Kędzierski 98875
Paweł Lampe 99277
Mateusz Sikora 99615

Platforma zarządzania zdarzeniami na urządzeniach mobilnych if{y}

Bachelor's Thesis

Supervisor: dr inż. Jerzy Błaszczyński

Poznań, 2014

Spis treści

Rozdział 1

Wstęp

1.1 Opis problemu i koncepcja jego rozwiązania / motywacja

1.2 Cele i zakres pracy

1.3 Podział prac

TODO: przedmowa

- Patryk Dąbrowski
  - text
  - text
- Aleksander Kędzierski
  - text
  - text
- Paweł Lampe
  - Implementacja targowiska
  - Administracja serwerem z systemem Linux
- Mateusz Sikora
  - text
  - text

1.4 Omówienie pracy

Nixx nett hier

Rozdział 2

Wymagania

TODO: przedmowa

2.1 Wymagania funkcjonalne

2.1.1 Przypadki użycia platformy

UC1 Tworzenie Recepty

1 Użytkownik wchodzi na stronę Targowiska.

2 Użytkownik tworzy nową Receptę.

3 Użytkownik pisze kod w edytorze online.

3a Użytkownik pisze kod lokalnie (np. w Eclipse) i przekazuje kod do Targowiska.

4 Serwer kompiluje receptę.

5 Użytkownik pobiera receptę na telefon.

6 Recepta działa na telefonie.

UC2 Ocena Recepty

1 Użytkownik wchodzi na stronę Targowiska.

2 Targowisko

2.1.2 Przypadki użycia Aplikacji - przykładowe Recepty

2.2 Wymagania pozafunkcjonalne

ID:	PF01
Nazwa:	System operacyjny dla aplikacji mobilnej
Kategoria:	Środowisko
Priorytet:	Wysoki
Opis:	Systemem operacyjnym aplikacji mobilnej jest Android w wersji minimum 2.1.

ID:	PF02
Nazwa:	Środowisko uruchomieniowe dla aplikacji serwerowej
Kategoria:	Środowisko
Priorytet:	Wysoki
Opis:	Aplikacja serwerowa powinna działać na maszynie wirtualnej Java.

ID:	PF03
Nazwa:	Używane technologie
Kategoria:	Technologie
Priorytet:	Wysoki
Opis:	Wykorzystane technologie nie mogą być płatne

ID:	PF04
Nazwa:	Zunifikowane środowisko programistyczne
Kategoria:	Narzędzia
Priorytet:	Wysoki
Opis:	Programiści muszą zdecydować się na wspólne narzędzie do redagowania kodu (np. Eclipse)

ID:	PF05
Nazwa:	Ograniczone zużycie energii urządzenia mobilnego
Kategoria:	Wydajność i niezawodność
Priorytet:	Średni
Opis:	Działanie aplikacji nie powinno w znaczącym stopniu skracać czasu pracy urządzenia na baterii

ID:	PF06
Nazwa:	Ograniczone zużycie zasobów urządzenia mobilnego.
Kategoria:	Wydajność i niezawodność
Priorytet:	Średni
Opis:	Aplikacja nie powinna spowalniać działania innych aplikacji.

ID:	PF07
Nazwa:	Czas reakcji aplikacji na zdarzenie
Kategoria:	Wydajność i niezawodność
Priorytet:	Wysoki
Opis:	Aplikacja powinna reagować na zdarzenia lokalne w mniej niż 2 sekundy

ID:	PF08
Nazwa:	Zgodność ze standardami kodowania dla języka Java
Kategoria:	Zgodność ze standardami
Priorytet:	Wysoki
Opis:	Zarówno kod aplikacji mobilnej, jak i serwerowej powinien być redagowany zgodnie ze standardami dla języka Java

ID:	PF09
Nazwa:	Przechowywanie haseł
Kategoria:	Bezpieczeństwo
Priorytet:	Wysoki
Opis:	Szyfrowane zapamiętywanie hasła użytkownika.

ID:	PF10
Nazwa:	Przechowywanie haseł
Kategoria:	Bezpieczeństwo
Priorytet:	Wysoki
Opis:	Przechowywanie skrótu hasła na serwerze.

ID:	PF11
Nazwa:	Długość recepty
Kategoria:	Użyteczność
Priorytet:	Wysoki
Opis:	Kod recepty powinien być możliwie najkrótszy

ID:	PF12
Nazwa:	Tworzenie recept
Kategoria:	Użyteczność
Priorytet:	Wysoki
Opis:	Proces tworzenia recepty powinien być możliwie najprostszy

ID:	PF13
Nazwa:	Dystrybucja recept
Kategoria:	Użyteczność
Priorytet:	Średni
Opis:	Proces dystrybucji recepty powinien być możliwie najprostszy

ID:	PF14
Nazwa:	Bezpieczeństwo recept
Kategoria:	Bezpieczeństwo
Priorytet:	Wysoki
Opis:	Recepta powinna korzystać tylko z biblioteki if{y} oraz pakietów narzędziowych Java

Rozdział 3

Zarządzanie zdarzeniami na urządzeniach mobilnych

TODO: przedmowa

3.1 Definicja pojęć

- Podfunkcjonalność (ang. Feature) – Część biblioteki zapewniająca Receptom dostęp do pozdbioru funkcjonalności Androida.
- Zdarzenie (ang. Event) – Zmiana stanu systemu, która powoduje uruchomienie kodu Recepty.
- Recepta (ang. Recipe) – Napisany przez użytkownika fragment kodu opisujący, co ma się zdarzyć po spełnieniu pewnych warunków.
- Targowisko (ang. Market) – Aplikacja internetowa pozwalająca tworzyć i pobierać Recepty.
- Aplikacja – Aplikacja androidowa wykorzystująca bibliotekę if{Y}.
- Serwer Grup – Komputer z działającą aplikacją, która zarządza grupami użytkowników i Zdarzeniami Grupowymi.
- Zdarzenie Grupowe – Zdarzenie związane z Grupą, wysyłane lub odbierane przez Aplikację z Serwera Grup.
- Grupa – Zbiór użytkowników identyfikowalny przez nazwę zdefiniowany na Serwerze Grup.
- Dziennik (ang. Log) – Moduł systemu odpowiedzialny za zapis zdarzeń.

3.2 Istniejące rozwiązania

3.2.1 On X

Aplikacja firmy Microsoft umożliwiającą kontrolowanie telefonu z systemem Android używając kodu napisanego w JavaScript. Umożliwia wysyłanie Zasad (and. Rules) na telefon

poprzez stronę internetową. Dostęp do funckcjonalości systemu Android jest zapewniony przez API w postaci Wyzwalaczy (ang. Triggers) i Akcji (ang. Actions). Cały system jest [niestety] połączony z Facebookiem i wymaga posiadania tam konta. Na podstawie [1].

3.2.2 Tasker

Rozdział 4

Architektura platformy

System składa się z biblioteki, przykładowej aplikacji appIFY oraz aplikacji działających na serwerze - Serwera Grup oraz Targowiska. Aplikacja korzysta z biblioteki oraz komunikuje się z serwerem. Oprócz tego Serwer Grup oraz Targowisko udostępniają z poziomu przeglądarki takie funkcje jak rejestracja użytkowników czy tworzenie Recept. Kluczowym założeniem było maksymalne uproszczenie kodu recept.

Kod Aplikacji jest podzielony na dwie części:

- bibliotekę IFY
- aplikację appIFY

Celem takiego podziału jest ułatwienie tworzenia innych aplikacji opartych o bibliotekę.

4.1 Recepty

Miejscem, gdzie zdefiniowana jest właściwe działanie Aplikacji są Recepty – są w nich opisane wszystkie zdarzenia, które mają nastąpić po spełnieniu pewnych warunków. Docelowo będą one tworzone przez użytkowników i pobierane z Targowiska, jednak istnieją także przykładowe Recepty wbudowane w Aplikację, mające na celu ułatwienie użytkownikom tworzenia nowych na ich podstawie oraz rozszerzenie początkowej funkcjonalności aplikacji. Na receptę składają się:

- opis używanych podfunkjonalności
- opis wymaganych parametrów
- opis jej właściwego działania

Deklarowanie używanych podfunkcjonalności ma dwa główne cele - po pierwsze, użytkownik widzi, czego używa recepta, co nieco poprawia jego bezpieczeństwo przy używaniu recept innych użytkowników, po drugie pozwala to inicjalizować nasłuchiwanie zdarzeń systemowych tylko wtedy, gdy istnieje aktywna recepta, która na nie reaguje - kod recepty nie musi inicjalizować większości podfunkcjonalności, wystarczy deklaracja ich używania. Wyjątkiem jest podfunkcjonalność grup, gdzie komunikację należy zainicjalizować.

Parametry pozwalają użytkownikowi na dostosowanie recepty do swoich wymagań, bez potrzeby pisania nowej. W naszych przykładowych receptach były to np. numer telefonu do wysłania SMS lub jego tekst czy też zasięg znajdowania znajomych na podstawie GPS.

TODO: Lanie wody poniżej? Właściwa logika recepty jest zawarta w funkcji reakcji na zdarzenie. Jest to rozwiązanie podobne do wzorca obserwatora, Recepta staje się jednak obserwatorem automatycznie na podstawie zadeklarowanych podfunkcjonalności, a wszytkie zdarzenia wywołują tą samą metodę w Recepcie. Takie rozwiązanie pozwala zmniejszyć ilość kodu w receptach.

4.1.1 Cykl życia

W konstekście platformy if{y} zdefiniować można następujący cykl życia recepty:

1. Pisanie kodu
2. Kompilacja i budowa
3. Dystrybucja
4. Pobranie na urządzenie mobilne
5. Uruchomienie
6. Działanie

4.2 Biblioteka

Biblioteka zawiera głównie API dostępne z poziomu recept, czyli między innymi Podfunkcjonalności, które agregują i upraszczają dostęp do metod z API systemu Android. Oprócz tego znajduje się tam moduł odpowiedzialny za zarządzanie cyklem życia Recept i Podfunkcjonalności, który działa cały czas w tle. Podfunkcjonalności są inicjalizowane przez serwis przy uruchamianiu recepty, która deklaruje ich użycie. Zapewniają one dostęp do określonych funkcji, takich jak odczyt danych z sensorów, odbieranie i wysyłanie SMS'ów i wiele innych.

4.3 Aplikacja kliencka

Najważniejszym elementem Aplikacji jest interfejs użytkownika – ekrany takie jak wyświetlanie listy dostępnych lub aktywnych recept, ustalanie ich parametrów i ich włączanie i wyłączanie. Dodatkowo aplikacja jest zintegrowana z Targowiskiem umożliwiając pobieranie z niego recept. Umożliwia też logowanie się do Serwera Grup. TODO: Dop

4.4 Targowisko

4.4.1 Analiza problemu

4.4.2 Rozwiązanie

4.4.3 Zintegrowane środowisko programistyczne

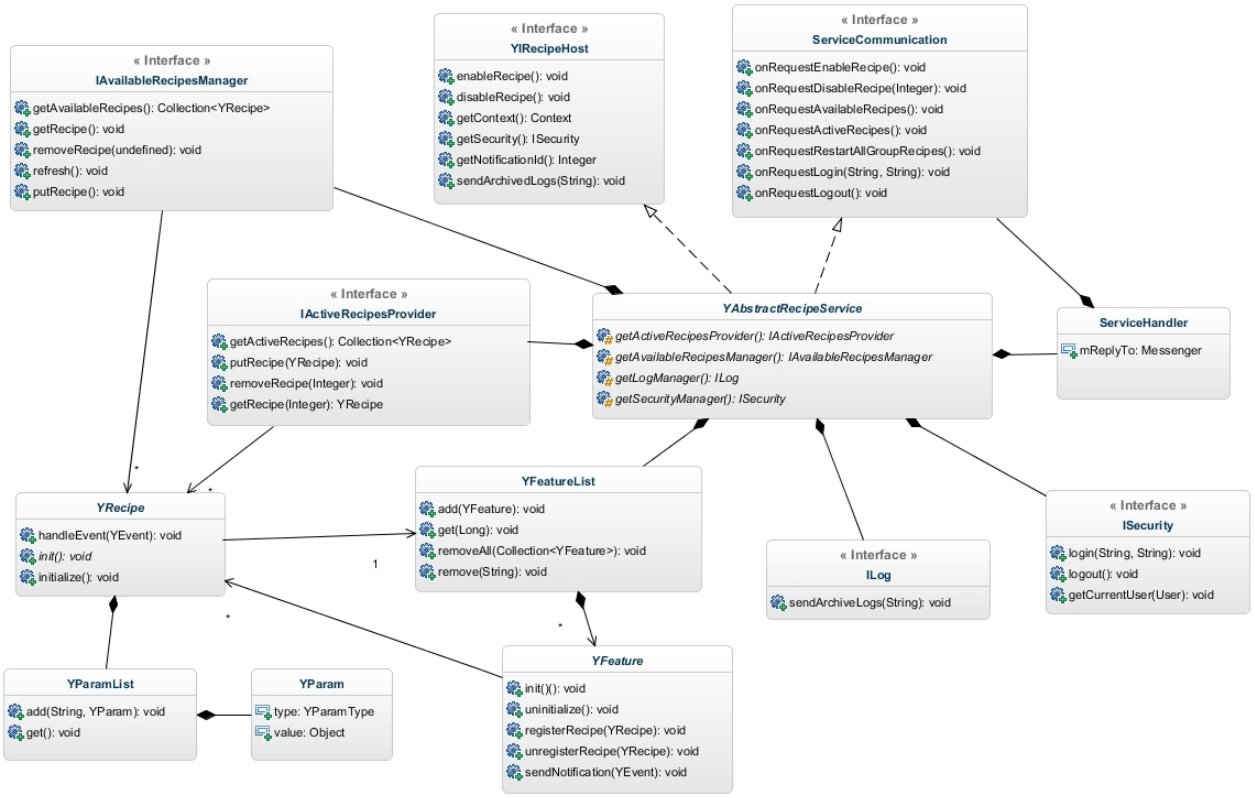
4.4.4 Repozytorium recept

4.4.5 Rozwidlanie

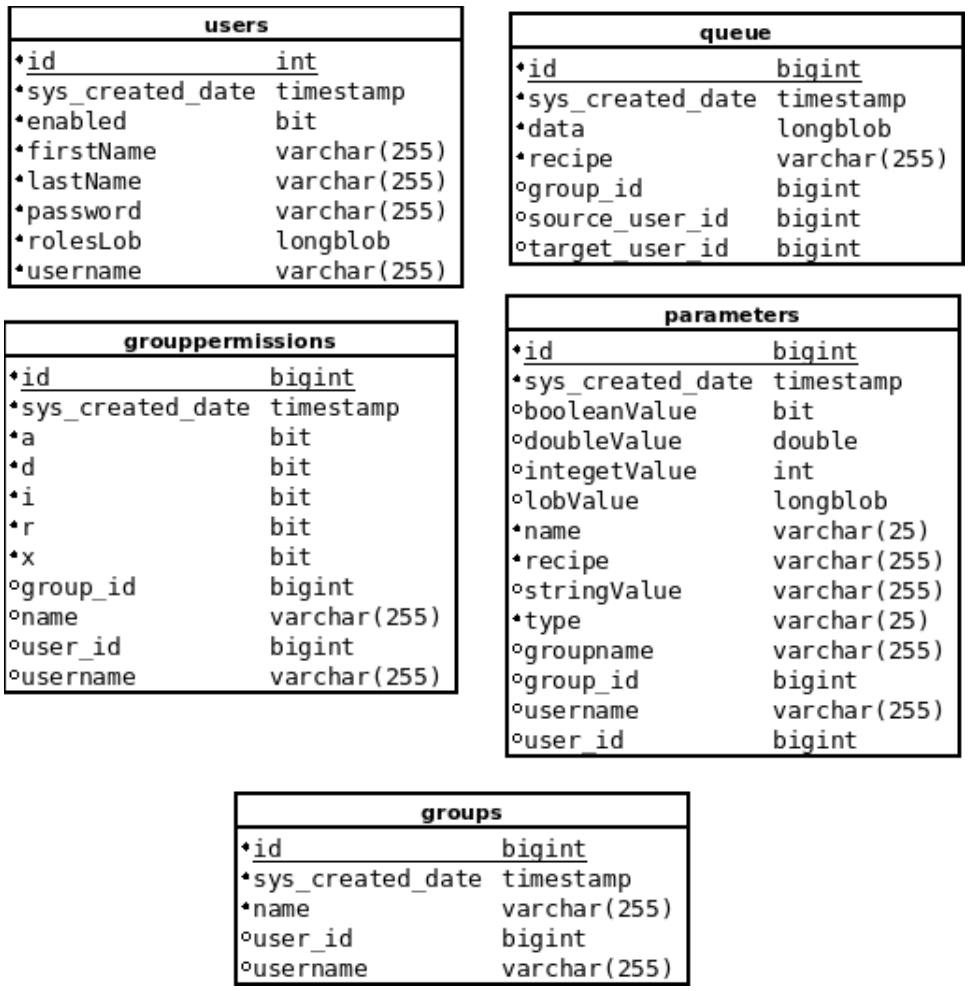
4.5 Serwer

Głównym zadaniem przy tworzeniu recept grupowych jest przekazywanie wiadomości umożliwiające komunikację miedzy klientami i wymianę danych. Informacje niezbędne do działania tak ważnej funkcjionalności powinny być jednocześnie rozsyłane w prosty i łatwy do odczytania sposób przez każdą ze stron. Jednym z podejść było wykorzystanie protokołu MQTT (MQ Telemetry Transport), który nie spełniał naszych oczekiwań ponieważ nie posiadał gotowych rozwiązań na systemy mobilne oraz okazał się zbyt trudny w implementacji przy założeniu projektu o jak najprostrzym rozwiązaniu komunikacji. Kolejnym sposobem rozwiązania problemu komunikacji mogła być usługa Google Cloud Messaging(GCM), lecz nie spełniała jednych z założeń projektu o odseparowaniu aplikacji od sieci społecznościowych i isniejących serwisów. Bezpośrednia komunikacja z użyciem połączenia internetowego między urządzeniami mobilnymi nie jest możliwa, dlatego niezbędnym było wprowadzenie urządzenia pośredniczącego w przesyłaniu danych. Serwer pełni w takim wypadku funkcję łacznika, dzieki czemu znany jest adres na jaki nalezy wysłac komunikat który ostatecznie ma dotrzeć do odbiorcy. Koncepcja ta opiera sie o tak zwany mechanizm odpytywania (ang. polling), który jest prosty do zaimplementowania ale jednocześnie spełnia wszytski wymagania stawiane w projekcie. Odbieranie danych z serwera wykonywane poprzez cykliczne zapytania eliminuje problem łączności między aplikacja a pośredniczącym serwerem.

Istotnym aspektem w wymianie danych między użytkownikami są ograniczenia do komunikacji aby niemozliwe było otrzymanie wiadomosci od niezidentyfikowanego użytkownika. Intuicyjnym rozwiązanie jest połaczenie użytwkoników w grupy, w obrębie których będa mogli rozsyłać wiadomości. Grupy pozwalają ograniczyć wymianę danych do skończonej liczby użytkowników.



Rysunek 4.1: TODO:



Rysunek 4.2: Schemat bazy danych serwera

Rozdział 5

Opis implementacji

5.1 Recepty

Recepty dziedziczą po klasie abstrakcyjnej YRecipe i implementują jej abstrakcyjne metody. Obrazuje to poniższy przykład recepty, która odrzuca wszystkie nadchodzące połączenia i wysyła SMS o zdefiniowanej przez użytkownika treści do dzwoniącej osoby.

```
public class YSampleCallsSMS extends YRecipe {
   @Override
   public void requestParams(YParamList params) {
      //Message to send in SMS
      params.add("MSG",YParamType.String, "Sorry, I'm busy.");
   }

   @Override
   public long requestFeatures() {
      return Y.Calls | Y.SMS;
   }

   @Override
   public void handleEvent(YEvent event) {
      //event is incoming call
      if(event.getId() == Y.Calls){
         YCallsEvent e = (YCallsEvent) event;
         //extract phone number
         String phone = e.getIncomingNumber();
         //discard call
         mFeatures.getCalls().discardCurrentCall();
         //send sms
         mFeatures.getSMS().sendSMS(phone, mParams.getString("MSG"));
      }
   }
   @Override
   public String getName() {
```

```
      return "YSampleCallsSMS";
   }
   @Override
   public YRecipe newInstance() {
      return new YSampleCallsSMS();
   }
}
```

5.1.1 Parametry – requestParams

Metoda requestParams ma za zadanie poinformować, jakich parametrów recepta wymaga do działania. Początkowo miała ona po prostu zwrócić listę i wyglądałaby tak:

```
public void requestParams() {
   YParamList params = new YParamList();
   params.add("MSG",YParamType.String, "Sorry, I'm busy.");
   return params;
}
```

jednak tworzenie listy i zwracanie jej to dwie linie, które byłyby identyczne w każdej recepcie - ich wpisywanie może nieco irytować. Wobec tego obecnie metoda ta przyjmuje jako argument pustą listę parametrów, którą ma za zadanie wypełnić, zgodnie z założeniem maksymalnego uproszczenia kodu recepty.

5.1.2 Używane Podfunkcjonalności – requestFeatures

Metoda requestFeatures ma za zadanie poinformować system, jakich Podfunkcjonalności używa Recepta. Początkowo była ona podobna do requestParams i wypełniała listę nowymi obiektami odpowiedniej klasy, co wyglądałoby tak:

```
public void requestFeatures(YFeatureList features) {
   features.add(new YCallsFeature());
   features.add(new YSMSFeature());
   return params;
}
```

Przy takim rozwiązaniu jednak tworzyło się wiele niepotrzebnych obiektów - poprawnie zainicjalizowane Podfunkcjonalności powinny być tworzone w systemie tylko raz. Wystarczyłaby zatem lista identyfikatorów, pozwalająca zainicjalizować odpowiednie Podfunkcjonalności. Identyfikatorów jest jednak na tyle mało, że tak naprawdę nie potrzeba prawdziwej listy, wystarczy maska bitowa. Ułatwia to przesyłanie takiej listy między modułami systemu, działającymi w różnych procesach - nie trzeba się martwić o implementację w liście interfejsu Parcelable, potrzebnego do przesyłania obiektów między procesami w Androidzie.

Ostatecznie zatem metoda ta zwraca liczbę typu long, będącą sumą bitów reprezentujących poszczególne Podfunkcjonalności. Mapowanie tych bitów jest zawarte w klasie Y.

```
[...]
public static final long Wifi = 0x0008;
public static final long GPS = 0x00010;
[...]
```

Dodatkowo warto zauważyć, że nazwy stałych w tej klasie odpowiadają nazwom Podfunkcjonalności oraz Zdarzeń - dla stałej **ABC** klasa z Podfunkcjonalnością nazywa się Y**ABC**Feature, a zdarzenie - Y**ABC**Event. Powinno to ułatwić automatyczne generowanie kodu recept.

5.1.3 Logika recept – handleEvent

Metoda jest wywoływana, gdy w systemie nastąpi zdarzenie związane z Podfunkcjonalnością używaną przez receptę. W argumencie podawane jest zdarzenie – obiekt typu YEvent. Aby poznać szczegóły zdarzenia recepta musi sprawdzić jego typ porównując wartość zwracaną przez getId() ze stałymi z klasy Y. Następnie można zrzutować zdarzenie na odpowiedni typ i poznać jego szczegóły.

Recepty mogą też zażądać pewnych danych od systemu, które są dostarczane asynchronicznie - na przykład przetłumaczenie danych z GPS na adres (Geocoder). Wyniki tego typu operacji również są przekazywane do recepty jako typ YEvent.

Z poziomu obsługi zdarzenia można także dostać się do listy Podfunkcjonalności oraz listy Parametrów poprzez metody getFeatures() i getParams(). Początkowo dostęp do Podfunkcjonalności odbywał się następująco:

```
YCallsFeature cf = (YCallsFeature) mFeatures.get(Y.Calls);
```

Jednak wymuszało to rzutowanie i niepotrzebnie wydłużało kod, zatem obecnie klasa YFeatureList zawiara metody pobierające konkretne podfuncjonalności.

```
public YCallsFeature getCalls() {
    return (YCallsFeature) get(Y.Calls);
}
```

Ich utrzymanie może być później nieco kłopotliwe - każde dodanie Podfunkcjonalności będzie wymagało dodania odpowiedniej metody, jednak uproszczenie kodu recepty jest tego warte.

Warto również wspomnieć, że metoda handleEvent może rzucić dowolny wyjątek - recepta zostanie wówczas wyłączona. Ułatwia to pisanie recept zapewniając jednocześnie stabilność aplikacji.

5.1.4 Aktywacja

Fragmenty kodu przedstawione poniżej różnią się od oryginalnych – dla poprawy czytelności nie ma w nich tworzenia logów. Recepta jest aktywowana przez serwis, na podstawie nazwy i listy parametrów.

```
   public int enableRecipe(String name, YParamList params) {
      int id = ++mRecipeID;
```

```
    int timestamp = (int) (System.currentTimeMillis() / 1000);
    YRecipe recipe = mAvailableRecipesManager.getRecipe(name).newInstance();
    long feats = recipe.requestFeatures();
    YFeatureList features = new YFeatureList(feats);
    initFeatures(features);
    params.setFeatures(feats);
    if(!recipe.initialize(this, params, features, id, timestamp)){
       return 0;
    }
    for (Entry<Long, YFeature> entry : features) {
       entry.getValue().registerRecipe(recipe);
    }
    mActiveRecipesManager.put(id, recipe);
    return id;
 }
```

Generowany jest ID konretnej instancji recepty oraz zapisywany jest czas jej uruchomienia. Następnie tworzony jest nowy obiekt typu właściwego do konkretnej recepty. W tym celu znajdujemy niezainicjaliwaną receptę w bazie i posługujemy się metodą newInstance - nie w tym miejscu kodu nie jest znana nazwa klasy recepty, aby móc wprost wywołać konstruktor. Innym możliwym rozwiązaniem byłby mechanizm refleksji, jednak to rozwiązanie jest szybsze, gdyż nie mogą być optymalizowane przez maszynę wirtualną [5] Dalej na podstawie zwróconej przez receptę maski bitowej tworzona jest lista podfunkcjonalności wymaganych przez receptę do działania. Następnie podfunkcjolności które już są aktywne są wpisywane do listy w miejsce niezainicjalizowanych, a pozostałę są aktywowane i dodawane do listy aktywnych.

```
 protected void initFeatures(YFeatureList features) {
    for (Entry<Long, YFeature> entry : features) {
       Long featId = entry.getKey();
       YFeature feat = mActiveFeatures.get(featId);
       if (feat != null) {
          entry.setValue(feat);
       } else {
          feat = entry.getValue();
          feat.initialize(this);
          mActiveFeatures.add(feat);
       }
    }
 }
```

Po zainicjalizowaniu Podfunkcjonalności Recepta jest w nich rejestrowana. Umożliwia to wywoływanie metody handleEvent w odpowiedzi na zdarzenia systemowe. Warto zauważyć, że zarówno Recepty jak i Podfunkcjonalności są leniwie inicjalizowane, co pozwala tymczasowo używać niezainicjalizowanych obiektów, a potem zastępować je innymi bez wykonywania zbędnych operacji.

```
public final boolean initialize(IYRecipeHost host, YParamList params,
      YFeatureList features, int id, int timestamp) {
   mHost = host;
   mParams = params;
   mFeatures = features;
   mId = id;
   mTimestamp = timestamp;
   Log = new YLogger(createTag(mId, getName()), host);
   try {
      init();
   } catch (Exception e) {
      e.printStackTrace();
      return false;
   }
   return true;
}
```

Sama inicjalizacja recepty to głównie wstrzyknięcie jej parametrów, Podfunkcjonalności, ID oraz czasu aktywacji. Oprócz tego jest tworzony Dziennik Recepty oraz jest wywoływana funkcja init() zawierająca kod inicjalizacyjny specyficzny dla danej recepty (na przykład otwarcie kanału komunikacji z Serwerem Grup). Takie rozwiązanie w połaczeniu w modyfikatorem final w metodzie zapewnia jej wywołanie, a kod recepty nie ma dostępu do danych, które nie są mu potrzebne. Dodatkowo funkcja init() może się nie powieść - wyjątki są wówczas łapane, metoda initialize() zwraca wówczas wartość false, a recepta nie jej dodawana do listy aktywnych.

5.1.5 Deaktywacja

Deaktywacją recepty również zajmuje się serwis. Polega ona na usunięciu wyrejestrowaniu jej z Podfunkcjonalności, co powoduje, że nie dostanie ona powiadomienia o zdarzeniach, a następnie usunięciu jej z listy dostępnych recept. Dodatkowo są odinicjalizowane podfunkcjonalności, z których nie korzysta żadna inna recepta. Ich usuwanie z listy odbywa się w drugim przebiegu pętli, aby zabezpieczyć się przed wyjątkiem ConcurrentModificationException.

```
public void disableRecipe(Integer id) {
   YRecipe recipe = mActiveRecipesManager.get(id);

   List<Long> toDelete = new ArrayList<Long>();
   for (Entry<Long, YFeature> entry : recipe.getFeatures()) {
      YFeature feat = entry.getValue();
      YLog.d("SERVICE", "UnregisterRecipe: " + recipe.getName()
            + " from " + entry.getKey());
      feat.unregisterRecipe(recipe);
      if (!feat.isUsed()) {
         toDelete.add(entry.getKey());
```

```
            YLog.d("SERVICE", "UninitializeFeature: " + feat.getId());
            feat.uninitialize();
        }
    }
    mActiveFeatures.removeAll(toDelete);
    mActiveRecipesManager.remove(id);
}
```

5.2 Biblioteka

5.2.1 Serwis

Wszystkie operacje odbywające się w bibliotece działają w kontekście serwisu, zaimplementowane w klasie YAbstractRecipeService. Serwis w Androidzie to komponent aplikacji przeznaczony do długotrwałego wykonywania operacji w tle, nieposiadający interfejsu użytkownika. [6] Wszelka komunikacja z użytkownikiem przebiega poprzez aplikację, która komunikuje się z serwisem.

5.3 Podfunkcjonalności

Podfunkcjonalności to klasy agregujące pewne funkcje związane z systemem. Muszą być inicjalizowane, gdy zajdzie taka potrzeba i przechwytywać zdarzenia systemowe, przekazując je odpowiednim receptom. Klasą bazową jest dla nich YFeature. Są w niej zaimplementowane metody związane z czasem życia Podfunkcjonalności i Recepty, odpowiedzialne za rejestrowanie i odrejestrowywanie Recept, inicjalizację i deinicjalizację Podfunkcjonalności oraz sprawdzanie, czy Podfunkcjonalność jest używana przez recepty. Poza tym znajduje się w niej metoda odpowiedzialna za wysyłanie zdarzenia do Recept - sendNotification, wykorzystywana w poszczególnych Podfunkcjonalnościach. Zaimplementowano następujące podfunkcjonalności:

- Akcelerometr (YAccelerometerFeature.java)
  Umożliwia reagowanie na odczyty akcelerometru wbudowanego w urządzenie.
- AudioManager (YAudioManager.java)
  Umożliwia zarządzanie poziomem głośności dzwonka.
- Battery (YBatteryFeature.java)
  Umożliwia reagowanie na zmiany poziomu baterii urządzenia.
- Calls (YCallsFeature.java)
  Umożliwia reagowanie na połączenia przychodzące i inicjowanie połączeń wychodzących.
- Files (YFilesFeature.java)

- Geocoder (YGeocoderFeature.java)
  Umożliwia pobranie adresu związanego z podaną długościa i szerokością geograficzną.
- GPS (YGPSFeature.java)
  Umożliwa śledzenie pozycji urządzenia za pomocą modułu GPS.
- Group (YGroupFeature.java)
  Niezbędny do obsługi zdarzeń grupowych.
- Intent (YIntentFeature.java)
  Pozwala wysyłać intencje[8] umożliwiające m. in. uruchamianie innych aplikacji.
- Internet (YInternetFeature.java)
  Umożliwia wysyłanie i pobieranie danych z podanego adresu.
- Notification (YNotificationFeature.java)
  Umożliwia wyświetlanie powiadomień w interfejsie graficznym urządzenia.
- RawPlayer (YRawPlayerFeature.java)
  Umożliwia odtwarzanie dźwięków na podstawie tablicy częstotliwości.
- Shortcut (YShortcutFeature.java.java)
  Pozwala na tworzenie skrótów do Recepty na głównym ekranie.
- SMS (YSMSFeature.java)
  Umożliwia wysyłanie wiadomości SMS oraz reagowanie na wiadomości przychodzące.
- Sound (YSoundFeature.java)
  Pozwala odtrzarzać pliki dźwiękowe.
- Text (YTextFeature.java)
  Umożliwia wprowadzanie tekstu do recepty z poziomu aplikacji.
- Time (YTimeFeature.java)

- Wifi (YWifiFeature.java)
  Umożliwia włączanie i wyłączanie modułu WiFi urządzenia.

5.4 Aplikacja kliencka

5.4.1 Obsługa Targowiska

Moduł obsługi Targowiska jest odpowiedzialny za wyświetlanie danych dotyczących recept dodanych w aplikacji internetowej oraz pobieranie plików .jar ze skompilowanymi receptami, które następnie są zapisywane na pamięci wewnętrznej urządzenia mobilnego (w celu zachowania tej samej bazy recept w przypadku w którym użytkownik usunie zewnętrzny nośnik pamięci z urządzenia). Informacje o plikach z receptami (ich nazwy oraz ścieżki) przechowywanymi na telefonie zapisywane są po pomyślnym pobraniu w bazie danych.

5.4.2 Obsługa pobranych Recept

W aplikacji klienckiej zrealizowanej w ramach pracy inżynierskiej rozróżniamy dwa typy recept - wbudowane i pobrane z Targowiska. Kod źródłowy recept pierwszego typu jest zawarty w kodzie źródłowym Aplikacji. W przypadku recept pobranych z Targowiska, w celu umożliwienia Aplikacji korzystania z takiej recepty wykorzystywane jest archiwum .jar, zawierające plik .dex (Dalvik Executable) z kodem wykonywalnym zrozumiałym dla maszyny wirtualnej Dalvik. Informacje potrzebne to załadowania kodu recepty (nazwa klasy oraz ścieżka dostępu do pliku .jar) przechowywane są w bazie danych recept pobranych na urządzenie.

5.4.3 Komunikacja serwisu z aplikacją kliencką

W opisie komunikacji między aplikacją kliencką a serwisem recept wykorzystane będą klasy z Android SDK - Messenger, Bundle i interfejs Parcelable. Klasa Messsenger umożliwia przesyłanie danych między procesami. [7] Do opakowania danych wykorzystywana jest klasa Bundle, która przechowuje obiekty i typy prymitywne w postaci mapy. Warto wspomnieć, że aby uzyskać możliwość przechowania obiektu w tej klasie, musi on implementować interfejs Serializable lub Parcelable. Pierwszy z nich umożliwia serializacje obiektów znaną z Javy, natomiast drugi został zaimplementowany w Android SDK w celu zwiększenia wydajności serializacji. W pracy inżynierskiej wykorzystujemy drugi z mechanizmów. Po uruchomieniu serwis recept wystawia obiekt implementujący interfejs IBinder służący do wiązania obiektów klasy Activity z obiektami klasy Service, z którym z kolei jest związany obiekt klasy Messenger zaimplementowany w serwisie recept. Aby ustanowić połączenie, Aktywność musi stworzyć obiekt klasy Intent, sparametryzować go klasą Service z którą nawiązywane jest połączenie i zapewnić obiekt implementujący interfejs ServiceConnection, który reaguje na uzyskanie i zerwanie połączenia, a następnie wywołać metodę bindService jako parametr podając wspomniany wyżej obiekt klasy Intent. Po nawiązaniu połączenia następuje wymiana obiektów klasy Messenger, dzięki czemu możliwa jest komunikacja w obie strony. Warto dodać, że ten mechanizm komunikacji jest asynchroniczny. Wiadomości wysyłane przez klasę Messenger odbierane są przez klasę Handler, ich zawartość jest interpretowana dzięki wysyłanemu kluczowi, a następnie dane są przekazywane serwisowi recept lub aktywności aplikacji klienckiej w celu dalszego przetwarzania. W pracy inżynierskiej wykorzystano dwie klasy dziedziczące po klasie Handler - ServiceHandler dla obsługi wiadomości przychodzących do serwisu recept i ActivityHandler dla obsługi wiadomości przychodzących do aplikacji klienckiej. W celu rozszerzenia komunikacji o wiadomości których obecna implemetnacja nie przewiduje, należy stworzyć własną klasę dziedziczącą po klasie ServiceHandler i we własnej implementacji klasy YAbstractService nadpisać metodę getServiceHandler. Podobnie, aby rozszerzyć komunikację w drugą stronę należy stworzyć własną klasę dziedziczącą po ActivityHandler i użyć go do odbierania wiadomości od serwisu recept.

5.5 Targowisko

5.5.1 Wzorzec MVC

5.5.2 Edytor Ace

5.5.3 Skrypt shell

5.5.4 API

5.6 Serwer

5.6.1 Repozytorium recept

5.6.2 Serwer recept grupowych

5.7 Protokół komunikacji

Komunikacja aplikacji klienckich oparta jest o ciagłe odpytywanie (ang. polling). Wymiana danych odbywa się przy użyciu tekstowego formatu danych JSON.

5.8 Użyte technologie

W tej części zaprezentowano opis technologii użytych bezpośrednio w implementacji składowych platformy.

- Android
  System operacyjny z rodziny Linux przeznaczony dla urządzeń mobilnych. Aktualnie rozwijane przez sojusz biznesowy Open Handset Alliance.
- Android SDK
  Platforma programistyczna umożliwiająca tworzenie aplikacji dla systemu Android. Zawiera wtyczkę do środowiska Eclipse, narzędzia wspierające prace programisty, emulator i biblioteki potrzebne do zbudowania aplikacji. Programy dedykowne platformie pisane są w języku Java i uruchamiane na maszynie wirtualnej Dalvik.
- Apache Commons
- Apache HTTP Server
- Git
  Rozproszony oraz wieloplatformowy system kontroli wersji będący wolnym oprogramowaniem.
- HTML 5

- Hibernate
  Narzędzie odwzorowań obiektowo-relacyjnych (ang. object-relation mapping, ORM) rozwijany na zasadzie wolnego oprogramowania. Umożliwia odworowania obiektowo-relacyjne, pamięć podręczną, leniwe (ang. Lazy loading), chciwe pobieranie oraz rozproszoną pamięć podręczną.
- JSON
  Skrót od JavaScript Object Notation. Jest to lekki, tekstowy format wymiany danych niezależny od języka programowania. Został wybrany ze względu na swoją czytelność i wsparcie ze strony bibliotek programistyzcnych.
- Java

- JavaScript
  Skryptowy język oprogramowania stosowany na stronach internetowych.
- Apache Maven
  Narzędzie automatycznego budowania oprogramowania dla języka JAVA. Głównymi problemami jakie rozwiązuje Maven przy budowaniu aplikacji są: zarządzanie zależnościami, mozliwość wieloma modułami, wsparcie dla testów.
- MySQL
  System zarządzania relacyjnymi bazami danych. Jest to wolne oprogramowanie szczególnie upodobane przez twórców aplikacji internetowych. Bardzo dobrze współpracuje z językami takimi jak PHP czy Java
- PHP
  Obiektowy język programowania dedykowany generowaniu stron internetowych w czasie rzeczywistym. Szczególnie użyteczny w przypadku tworzenia prototypów tudzież niewielkich projektów wymagających stosunkowo niskiego poziomu abstrakcji.
- RESTeasy
  Framework oprogramowania służacy do tworzenia aplikacji rozproszonych, oparty na wzorcu architektury oprogramowania Representational State Transfer(REST).
- SpringFramework
  Framework(Szkielet) tworzenia aplikacji w języku Java a w szczególności JavaEE. Do najważniejszych fukcji Springa zalicza się wstrzykiwanie zależności (ang. dependency injection, DI) oraz programowanie aspektowe (ang. aspect-oriented programming, AOP).
- Vaadin
  Framework sieciowy służący do tworzenia aplikacji sieciowych w szczególnosci interfejsu użytkownika w oparciu o Google Web Toolkit (GWT) w języku JAVA.
- JUnit
  Biblioteka służaca do tworzenia testów jednostkowych w jezyku Java.

5.9 Użyte narzędzia

- Apache Tomcat
  Kontener aplikacji sieciowych.
- Eclipse
  Popularne zintegrowane środowisko programistyczne (IDE) wspierające głównie język Java (wtyczki pozwalają obsługiwać inne języki).
- Android developer tools
  Wtyczka do Eclipse pozwalająca tworzyć aplikacje androidowe. Dodaje takie funkcjonalności jak edycja plików XML odpowiadających za wygląd aplikacji (również w trybie graficznym) czy debugowanie na telefonach oraz emulatorze.
- String Tool Suite
  Zintegrowane środowisko programistyczne oparte o Eclipsa dostosowany do Spring-Framework.
- Emacs
  Popularny, w pełni rozszerzalny edytor tekstowy spotykany głównie w systemach operacyjnych z rodziny Unix.
- Git bash for windows
  Narzędzie umożliwiające używanie Gita z linii poleceń w systemie Windows poprzez wbudowane środowisko MinGW.
- Github
  Serwis internetowy gromadzący społeczność programistów z całego świata. Służy jako hosting dla otwartoźródłowych projektów zarządzanych za pomocą systemu Git. Udostępnia szereg narzędzi wspierających - system śledzenia zadań, budowa statystyk.
- Latex
- Linux
  Rodzina systemów operacyjnych będących wolnym oprogramowaniem oraz używających jądra Linux.
- Notepad++
  Prosty edytor tekstowy umożliwiający kolorowanie składni w wielu językach.
- Przeglądarki internetowe
  Google Chrome, Mozilla Firefox, Opera
- Windows

5.10 Urządzenia mobilne

Aplikacja była testowana na następujących urządzeniach mobilnych:

- LG Swift GT540
  Procesor: Qualcomm MSM7227 600 MHz Pamięć RAM: 256 MB System operacyjny: Android 4.0.1 (Cyanogen mod)
- Media-Droid IMPERIUS EN3RGY MT7013 Procesor: dwurdzeniowy, 1GHz ARM7 MTK6577 Pamięć RAM: 256 MB System operacyjny: Android 4.1.2
- Motorola Defy MB525
  Procesor: TI OMAP3610 800 MHz Pamięć RAM: 512 MB System operacyjny: Android 4.3.1 (Cyanogen mod)
- Sony LT18 Xperia Arc S
  Procesor: Qualcomm MSM8255T 1,40 GHz Pamięć RAM: 512 MB System operacyjny: Android 4.0.4
- Samsung Galaxy Mini GT-S5570
  Procesor: Qualcomm MSM7227 600 MHz Pamięć RAM: 384 MB System operacyjny: Android 2.2

5.11 Opis pakietów

5.11.1 Pakiety Aplikacji

pl.poznan.put.cs.ify.app - główny pakiet Aplikacji. pl.poznan.put.cs.ify.jars - pakiet odpowiedzialny za zarządzanie plikami .jar zawierającymii recepty pobrane z Targowiska. pl.poznan.put.cs.ify.core - pakiet odpowiedzialny za zarządzanie dostępnymi i aktywowanymi Receptami. pl.poznan.put.cs.ify.appify.receipts - pakiet zawierający Recepty wbudowane w Aplikację. pl.poznan.put.cs.ify.app.ui - pakiet zawierający kontrolki interfejsu użytkownik. pl.poznan.put.cs.ify.app.ui.params - pakiet zawierający kontrolki interfejsu użytkownika wykorzystywane do wprowadzania parametrów przy inicjalizacji Recepty. pl.poznan.put.cs.ify.app.market - pakiet odpowiedzialny za pobieranie danych z Targowiska i wyświetlanie ich. pl.poznan.put.cs.ify.app.fragments - pakiet zawierający widoki ekranów aplikacji.

5.11.2 Pakiety Biblioteki

pl.poznan.put.cs.ify.api - pakiet główny Biblioteki. pl.poznan.put.cs.ify.api.exceptions - pakiet zawierający wyjątki, które mogą być rzucane przez metody z Biblioteki. pl.poznan.put.cs.ify.api.features - pakiet zawietający Podfunkcjonalności i Zdarzenia. pl.poznan.put.cs.ify.api.group - pakiet odpowiedzialny za obsługę Recept Grupowych. pl.poznan.put.cs.ify.api.log - pakiet odpowiedzialny za obsługę logowania i domyślny widok logów. pl.poznan.put.cs.ify.api.params - pakiet zawierający typy parametrów wykorzystywanych przez Recepty. pl.poznan.put.cs.ify.api.security - pakiet odpowiedzialny za moduł uprawnień Biblioteki. pl.poznan.put.cs.ify.api.types - pakiet zawierający typy danych wykorzystywanych przez Biblioteke.

5.11.3 Pakiety Serwera

pl.poznan.put.cs.ify.webify - pakiet główny serwera. pl.poznan.put.cs.ify.webify.data.dao - pakiet zawierający warstwe dostępu do danych. pl.poznan.put.cs.ify.webify.data.entity - pakiet zawierający klasy odwzorowywane na bazę danych. pl.poznan.put.cs.ify.webify.data.enums - pakiet zawierajacy potrzebne w bazie danych typy wyliczeniowe(np. lista ról). pl.poznan.put.cs.ify.webify - pakiet główny graficznego interfejsu użytkownika. pl.poznan.put.cs.ify.webify.gui.windows - paiet zawierający wszytskie okna aplikacji sieciowej. pl.poznan.put.cs.ify.webify.gui.components - pakiet zawierający komponenty użyte w aplikacji. pl.poznan.put.cs.ify.webify.gui.session - pl.poznan.put.cs.ify.webify.service - pakiet zawierający logikę. pl.poznan.put.cs.ify.webify.rest - pakiet zawerajacy obsługę zapytań typu REST. pl.poznan.put.cs.ify.webify.utils - pakiet, w którym przechowywane są funkcje pomocnicze używane w całym projkcie.

Rozdział 6

Testy oraz wyniki?

Rozdział 7

Zakończenie

.1 TODO

Cykl życia recepty i feature'a (ogólnie, rysunki, w architekturze) /sikor Dokładny opis deaktywacji recepty (implementacja) /alx Podfunkcjonalności do sekcji o blibliotece - ogólny opis + te z przewodnika usera. Przypadki użycia /alx Wymagania pozafunkcjonalne /alx UML Serwisu i okolic /sikor Schemat komunikacji - aplikacja <-> serwis /sikor - z serwerem Tworzenie jarów - rozszerzyć /sikor Apache commons, latex - dopisać lub wyjebać narzędzia - android support v4

Bibliografia

[1] Projekt on{X} http://www.onx.ms/#!findOutMorePage. Ostatnio odwiedzone 6/02/13.

[2] C. Walls. *Spring in action, 3rd edition*. Manning Publication Co, 2011.

[3] Vaadin https://vaadin.com/book/vaadin6/-/page/preface.html

[4] E. Gamma. *Design Patterns, First edition*. Person Education, Inc, 1995.

[5] The Reflection API http://docs.oracle.com/javase/tutorial/reflect/index.html Ostatnio odwiedzone 31.01.2014

[6] Android API Guide - Service http://developer.android.com/guide/components/services.html Ostatnio odwiedzone 31.01.2014

[7] Android API Guide - Messenger http://developer.android.com/guide/components/bound-services.htmlMessenger Ostatnio odwiedzone 31.01.2014

[8] Android API Guide - Intents and Intent Filters http://developer.android.com/guide/components/intents-filters.html Ostatnio odwiedzone 31.01.2014

[9] Introducing Google Play: All your entertainment, anywhere you go - http://googleblog.blogspot.com/2012/03/introducing-google-play-all-your.html Ostatnio odwiedzone 31.01.2014