Politechnika Wrocławska

Content Providers
Komunikacja przez internet
Web Services

Wiktor Ławski
Marcin Lubimow
Mateusz Dziekan
Marcin Filip
Pod opieką dra Pawła Rogalińskiego

Plan prezentacji - cz. 1 z 3

CONTENT PROVIDERS (Dziekan, Filip)

- Rola dostawców treści w systemie Android
- Modele danych i identyfikator URI
  - klasy Cursor, URI
- Odczytywanie danych udostępnianych przez dostawcę treści
  - klasa ContentValues, ContentResolver
- Tworzenie własnego dostawcy
  - klasa ContentProvider

Plan prezentacji - cz. 2 z 3

KOMUNIKACJA PRZEZ INTERNET (wszyscy)

- informacje ogólne (rodzaje przesyłanej treści)
- problemy i zagrożenia
- klasa HttpClient
- o interfejsach na emulatorach
- przykładowe aplikacje

Plan prezentacji - cz. 3 z 3

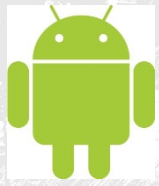
WEB SERVICES (Lubimow, Ławski)

- informacje ogólne
- przykłady Web Services dostępne w internecie
- przykładowa implementacja klienta Web Service (klient do Google Translate) – w tym wykorzystanie klas: URL, HttpUrlConnection

Politechnika Wrocławska

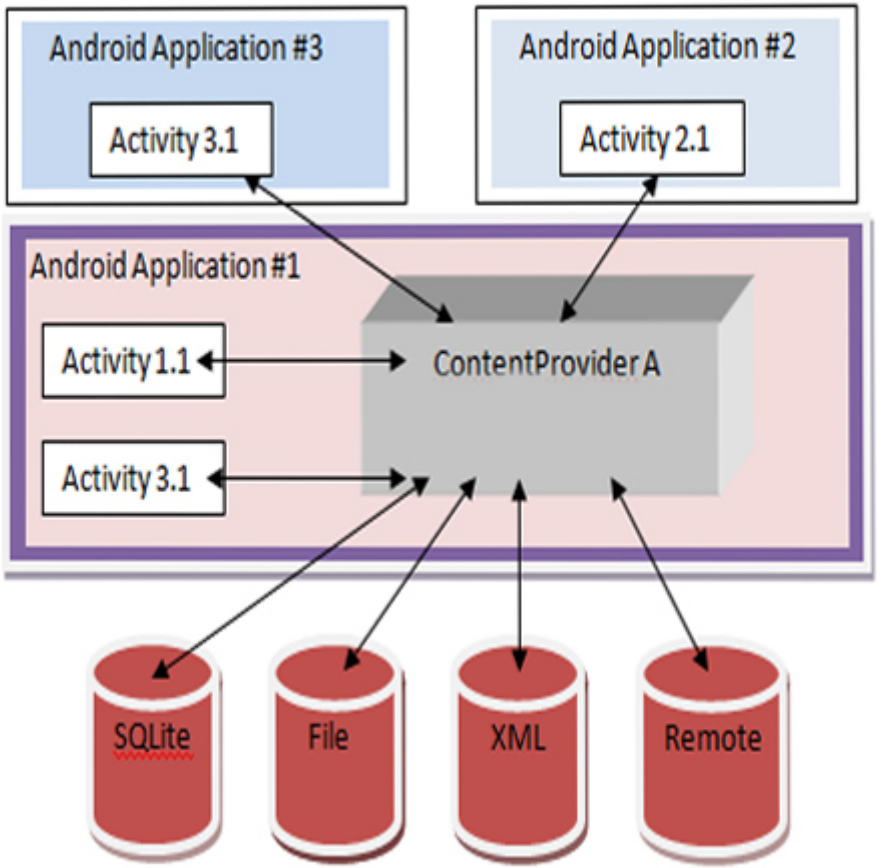
Content Providers
cz. 1
Opis i zastosowanie

Mateusz Dziekan



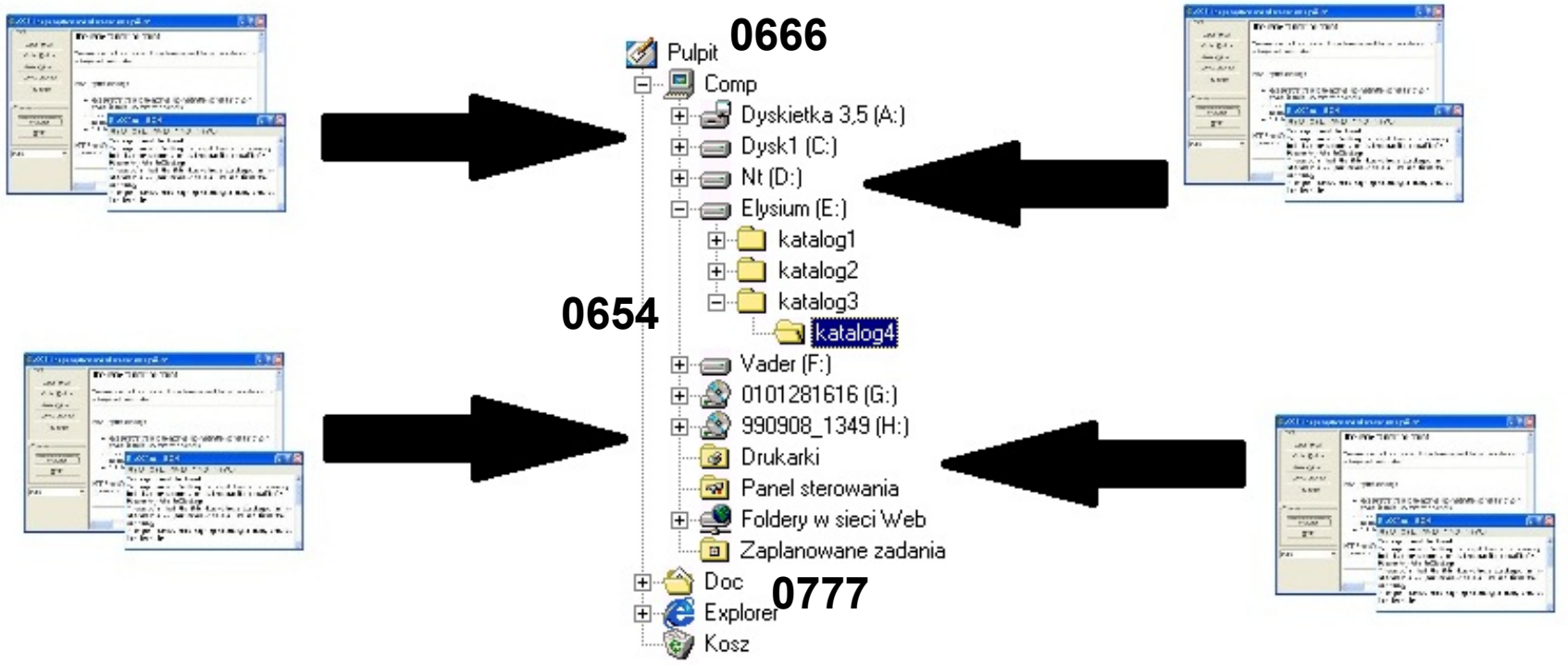
Rola dostawców treści w systemie Android

- Androidowy twór wynikający z przyjętego **modelu bezpieczeństwa**
- **Jedyny sposób** współdzielenia danych między aplikacjami

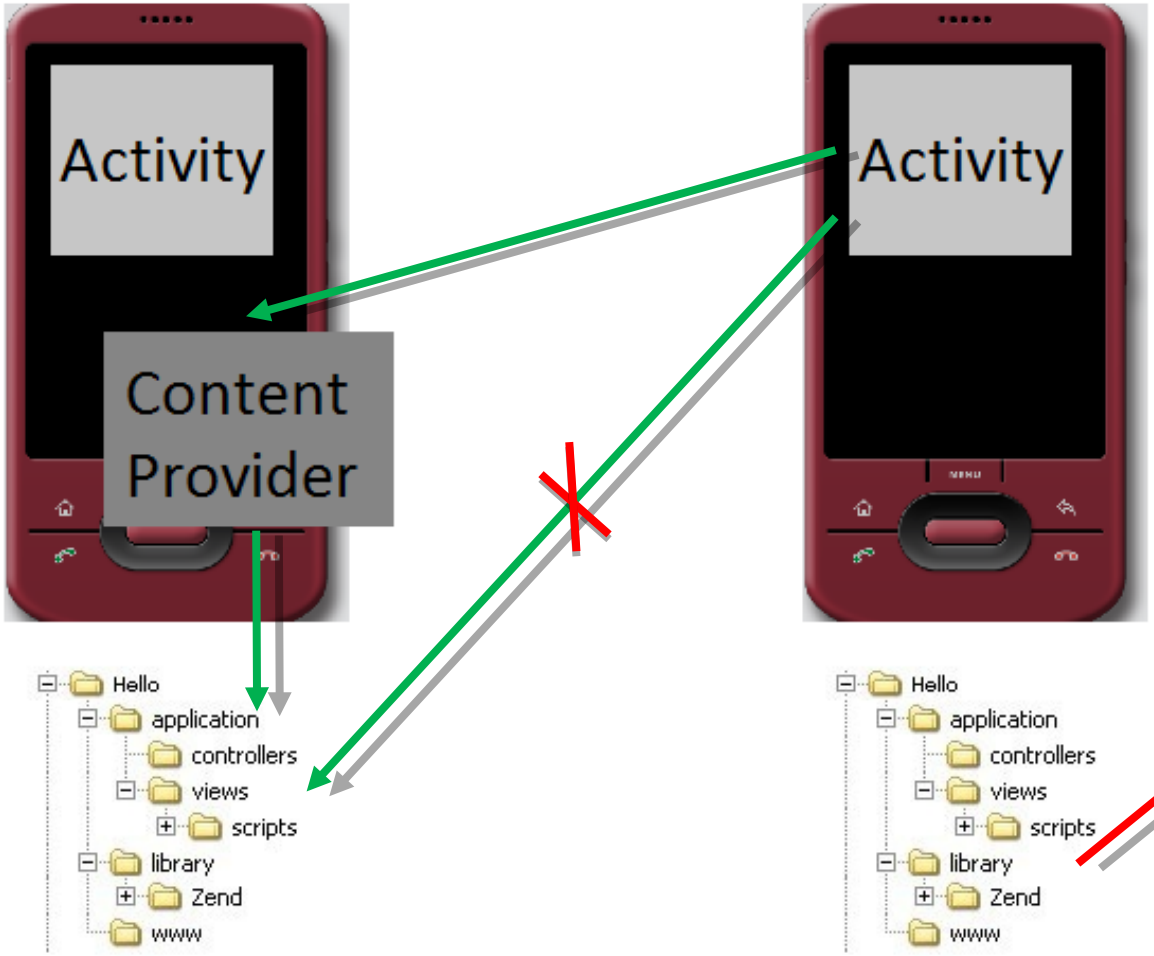


Grafika pochodzi z: http://www.mobisoftinfotech.com

System plików - jak to jest w większości systemów



A jak to rozwiązano w Androidzie?



Dane mogą być przechowywane również w bazie **SQL**, plikach **XML** itd.

Najważniejsi, standardowo dostępni dostawcy treści

- **ContactsContract**
- **MediaStore** (pliki audio, video, obrazki)
- **Browser** (zakładki, historia)
- **Settings** (preferencje urządzenia, ustawienia systemu)
- **CallLog** (ostatnie połączenia)
- **AlarmClock**

wcześniej **Contacts**

większość **tylko do odczytu**

Sposoby przechowywania prywatnych danych aplikacji

- **Preferencje** - pary klucz-wartość
- **SQLite** - bazy danych
- **Pliki** – poddrzewa katalogów przypisane danej aplikacji

Istnieje również możliwość przechowywania **publicznych danych** np. na **kartach SD**

Model danych Content Providera

- Prosta, relacyjna bazy danych

NIE SQLite

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

Tabela pochodzi z: http://developer.android.com

Klasa Cursor (1)

- *boolean* **moveToFirst**(),
  boolean **moveToLast**(),
  boolean **moveToNext**(),
  boolean **moveToPrevious**(),
  boolean **moveToPosition** (*int* position)

false – **Cursor** jest **pusty**

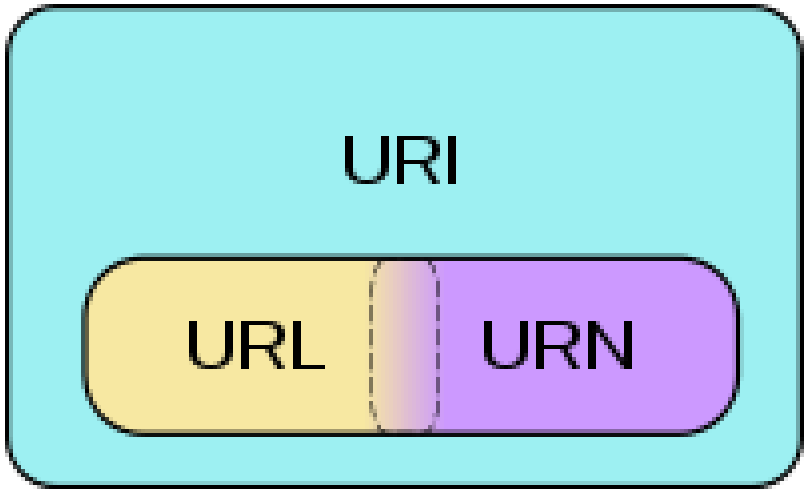
- *int* **getColumnIndex** (*String* columnName)
- *int* **getInt** (*int* columnIndex),
  String **getString** (*int* columnIndex),
  byte[] **getBlob** (*int* columnIndex),
  itd...

Klasa Cursor (2)

- *int* **getType** (*int* columnIndex):
  - FIELD_TYPE_NULL
  - FIELD_TYPE_INTEGER
  - FIELD_TYPE_FLOAT
  - FIELD_TYPE_STRING
  - FIELD_TYPE_BLOB

URI *(Uniform Resource Identifier)*

Jednolity Identyfikator Zasobów



➢**RFC 2396:**

„niewielki ciąg znaków identyfikujący abstrakcyjny bądź fizyczny zasób”

Każdy **zasób** ma własne **niepowtarzalne URI**

Przykłady URI

- Typy URI:
  - **Hierarchiczny** – podlega parsowaniu:

http://www.example.pl:8080/katalog/plik?parametr=wartosc#fragment

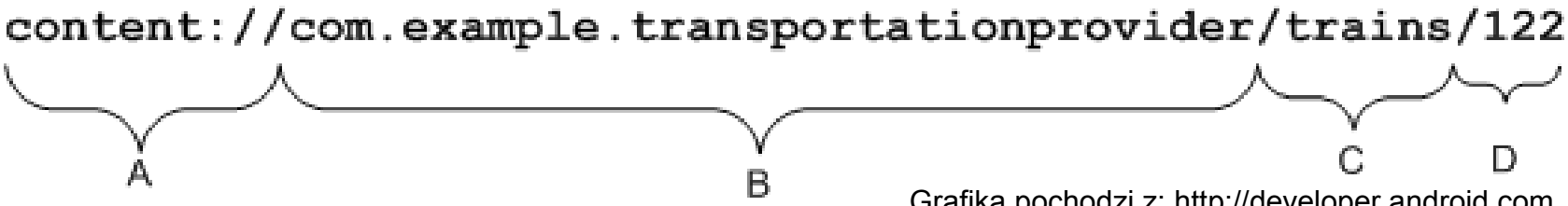
schemat (protokół)	autoryzacja	port	ścieżka do pliku	zapytanie	fragment

  - **Niehierarchiczny** (**nieprzezroczysty - *opaque***) – nie podlega parsowaniu:

schemat:część_zależna_od_schematu#fragment

mailto:jankowalski@exmpl.pl

Identyfikator URI Content providera



Grafika pochodzi z: http://developer.android.com

- **A** – standardowy, niezmienny prefiks
- **B** - <**provider** android:**authorities**=„**com.example...**”> (*AndroidManifest.xml*)
- **C** – rodzaj zwracanych danych, nazwa tabeli
- **D** – wartość **_ID rekordu**

android.provider.Contacts.Phones.CONTENT_URI

Klasa URI

- Uri
  Uri.withAppendedPath
  (
  Uri contentUri, -> **URI** dostawcy
  String id, -> **ID** rekordu
  );

Można także użyć funkcji
ContentUri.withAppendedId

long

- Uri **Uri.parse** (String uriStr);
- String **getScheme**(), **getAuthority**(), **getFragment**()
  itp...
- Boolean **isHierarchical**(), **isOpaque**()

Tworzenie zapytań

- Cursor **ContentResolver**.**query**
(
    Uri uri, -> **URI** dostawcy
    String[] projection, -> **zwracane kolumny** (null-wszystkie)
    String selection, -> **SQL**'owe **WHERE**
    String[] selectionArgs, -> argumenty „?" w zapytaniu **WHERE**
    String sortOrder -> **SQL**'owe **ORDER BY**
);

Można także użyć funkcji Activity.managedQuery (pod kontrolą aktywności)

Tworzenie zapytań - przykład plik manifestu

```
<manifest ...>
    ...
    <uses-permission
    android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission
    android:name="android.permission.READ_CONTACTS" />
    <uses-permission
    android:name="android.permission.WRITE_CONTACTS" />
    <application ... >
        ...
    </application>
</manifest>
```

Tworzenie zapytań - przykład

```
import android.provider.ContactsContract.Contacts;

Uri contactsUri = Phone.CONTENT_URI;
String[] projection = new String[] { Phone.CONTACT_ID,
                                     Phone.DISPLAY_NAME,
                                     Phone.NUMBER };
String selection = Phone.NUMBER + „ LIKE ?”;
String selectionArgs[] = new String[] { "%444%" };
String sortOrder = Phone.DISPLAY_NAME + „ ASC”;
Cursor managedCursor = managedQuery (
        contactsUri,
        projection,
        selection,
        selectionArgs,
        sortOrder
);
```

? -> selectionArgs

Odczytywanie danych z Cursora - przykład

```
if ( managedCursor.moveToFirst() ) {

    String name;
    String phoneNumber;
    int nameColumn =
    managedCursor.getColumnIndex ( Phone.DISPLAY_NAME );
    int phoneColumn =
    managedCursor.getColumnIndex ( Phone.NUMBER );

    do {
        name = managedCursor.getString ( nameColumn );
            phoneNumber = managedCursor.getString ( phoneColumn );

    } while ( managedCursor.moveToNext() );
}
```

Modyfikowanie danych

- Dodawanie nowych rekordów
- Dodawanie nowych wartości do istniejących rekordów → Również **modyfikowanie**
- Aktualizowanie istniejących rekordów
- Usuwanie rekordów

Nowy mechanizm -> ContentProviderOperation

Klasa ContentValues

- ***ContentValues*** (-> konstruktor
  int size -> początkowy rozmiar
  );
- *void* **put** (
  String key, -> **nazwa kolumny**
  Byte/Integer/itd. value -> **wartość**
  );

Klasa ContentResolver

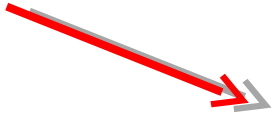
- *Uri* **insert** (*Uri* uri,
  ContentValues values);
- *int* **update** (*Uri* uri,
  ContentValues values,
  String selection,
  String[] selectionArgs);
- *int* **delete** (*Uri* uri,
  String selection,
  String[] selectionArgs);

Przykład (nowy kontakt)

```
ContentValues values = new ContentValues();

values.put(People.DISPLAY_NAME, „Abraham Lincoln”);
values.put(People.STARRED, 1);
Uri uri = getContentResolver().insert(People.CONTENT_URI, values);

Uri phoneUri =
    Uri.withAppendedPath(uri, People.Phones.CONTENT_DIRECTORY);
values.put(People.Phones.TYPE, People.Phones.TYPE_MOBILE);
values.put(People.Phones.NUMBER, „123321456”);
getContentResolver().insert(phoneUri, values);
```

Przykład (update, delete)

```
ContentValues values = new ContentValues();

String selection = People.Phones.NUMBER + „ LIKE ‚%444%’”;
Uri phoneUri = People.CONTENT_URI;
values.put(People.Phones.TYPE, People.Phones.TYPE_MOBILE);
values.put(People.Phones.NUMBER, „123321456”);
getContentResolver().update(phoneUri, values, selection, null);

getContentResolver().delete(phoneUri, null, null);
```

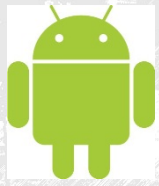


Usuwa **wszystkie** kontakty

Politechnika Wrocławska

Content Providers
cz. 2
Tworzenie własnego dostawcy

Marcin Filip



Tworzenie własnego dostawcy treści

- Stworzenie systemu do przechowywania danych (system plików lub *SQLite*)
- Rozszerzenie klasy ContentProvider
- Zadeklarowanie content providera w pliku manifestu (*AndroidManifest.xml*)

Klasa ContentProvider cz.1

Rozszerzenie klasy ContentProvider oznacza głównie zaimplementowanie 6 metod zadeklarowanych w klasie ContentProvider:

- boolean **onCreate** ()
- Cursor **query** (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
- String **getType** (Uri uri)

Klasa ContentProvider cz.2

- Uri **insert** (Uri uri, ContentValues values)
- int **update** (Uri uri, ContentValues values, String selection, String[] selectionArgs)
- int **delete** (Uri uri, String selection, String[] selectionArgs)

Trzeba pamiętać o tym, iż metody operujące na danych mogą być wywołane z rożnych obiektów klasy ContentResolver w rozmaitych procesach i wątkach.

Więcej na: http://developer.android.com/reference/android/content/ContentProvider.html

Tworzenie własnego Content Providera

Poza zdefiniowaniem samej podklasy są jeszcze inne kroki, które należy wykonać, aby uprościć prace z dostawcą treści i uczynić go lepiej dostępnym:

- Zdefiniowanie obiektu public static final Uri nazwanego CONTENT_URI

Przykład:

```
public static final Uri CONTENT_URI =
Uri.parse("content://pl.pkp.providerstacji");
```

Tworzenie własnego Content Providera

Jeśli nasz content provider zawiera podtablice należy zdefiniować też obiekty public static final Uri dla każdej z nich.

Przykład:

```
public static final Uri WROCGLOWNY_URI =
Uri.parse("content://pl.pkp.providerstacji/wrocglowny");
```

Tworzenie własnego Content Providera

- Zdefiniowanie nazw kolumn, które nasz dostawca treści będzie zwracać klientom. Dodatkowo zdefiniowanie obiektów public static final String, które klient będzie mógł użyć do określenia kolumn w zapytaniach bądź innych instrukcjach.

- Udokumentowanie typów danych przechowywanych w konkretnych kolumnach. Klient potrzebuje tych informacji do odczytu danych.

Tworzenie własnego Content Providera

- Jeśli dostarczasz dane, które są zbyt duże aby je umieścić w tabeli np. duży plik muzyczny, wtedy pola dostarczające te dane klientom powinny zawierać „content://URI String” w obiekcie Uri. W tym wypadku rekord powinien zawierać dodatkowe pole, nazwane „_data” , zawierające dokładną ścieżkę do tego pliku.

- Jeśli obsługujesz nowy typ danych to musisz zdefiniować nowy typ MIME do zwrotu w twojej implementacji **ContentProvider.getType()**. Typ zależy częściowo od tego czy Uri przekazane metodzie **getType()** ogranicza się do jednego rekordu.

Przykłady typów MIME

- Dla jednego rekordu:
  vnd.android.cursor.**item**/vnd.*nazwaprzedsiebiorstwa.typzawartosci*

  Np. dla pociągu, zatrzymującym się na dworcu Wrocławskim, pod rekordem 14 o Uri - content://com.pkp.providerstacji/wrocglowny/14

  vnd.android.cursor.**item**/vnd.*pkp.pociag*

- *Dla wielu rekordów:*
  vnd.android.cursor.**dir**/vnd.*nazwaprzedsiebiorstwa.typzawartosci*

  Np. dla wszystkich pociągów, zatrzymujących się na dworcu Wrocławskim, o Uri - content://com.pkp.providerstacji/wrocglowny/

  vnd.android.cursor.**dir**/vnd.*pkp.pociag*

Deklarcja własnego content providera

```
<provider android:name="com.pwr.MojContentProvider„
          android:authorities="com.pwr.mojcontentprovider">
</provider>
```

atrybuty opcjonalne:

- enabled=["true" | "false"]
- exported=["true" | "false"]
- grantUriPermissions=["true" | "false"]
- icon="drawable resource"
- initOrder="integer"
- label="string resource"
- multiprocess=["true" | "false"]
- permission="string"
- process="string"
- readPermission="string"
- syncable=["true" | "false"]
- writePermission="string"

Więcej na: http://developer.android.com/guide/topics/manifest/provider-element.html

Politechnika Wrocławska

Komunikacja przez internet

Wiktor Ławski
Marcin Lubimow
Mateusz Dziekan
Marcin Filip

Informacje ogólne - rodzaje przesyłanej treści

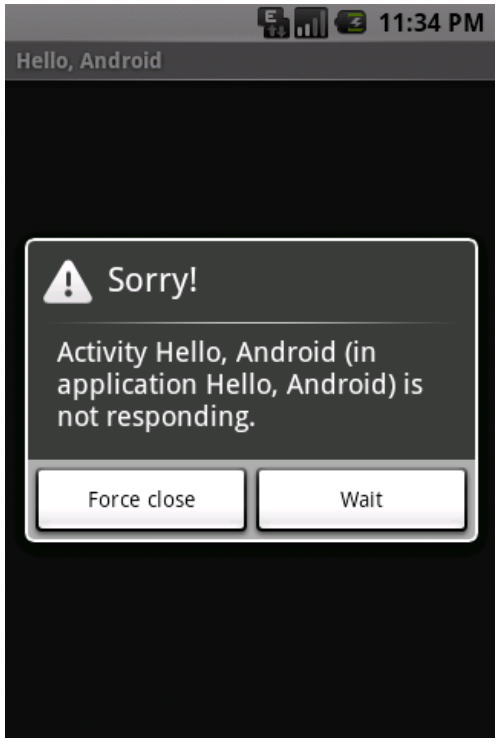
- tekst
- pliki (tekst, obraz, dźwięk, wideo)
- przesyłanie strumieniowe

Problemy i zagrożenia

- poufność danych
- sytuacje wyjątkowe – szczególne dla sieci komputerowych (serwer niedostępny, brak dostępu do sieci, brak odpowiedzi)

Konieczność stosowania wielowątkowości

- zniecierpliwienie użytkownika (2 s)
- „application not responding” ANR (5 s)



Grafika pochodzi z Android Developers – Designing for responsiveness (http://developer.android.com/images/anr.png)

Złe zaimplementowanie wielowątkowości

```
public void onClick(View v) {
        new Thread(new Runnable() {
            public void run() {
                Bitmap b = loadImageFromNetwork();
                mImageView.setImageBitmap(b); !!!
        }
        }).start();
}
```

klasa ImageView

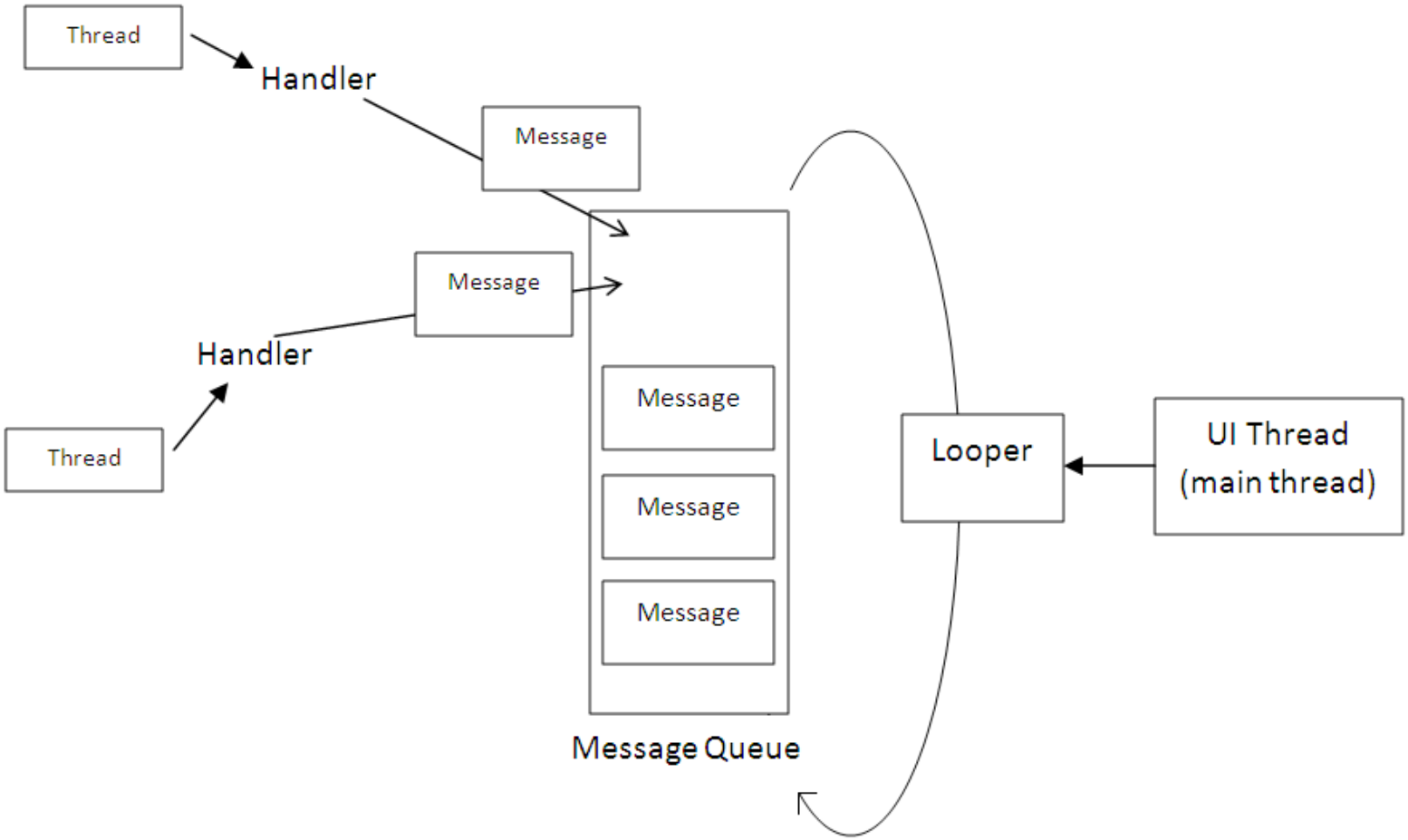
Przykład pochodzi z: http://developer.android.com/resources/articles/painless-threading.html

Poprawne implementowanie wielowątkowości

Wymagania

- interfejs ciągle dostępny dla użytkownika
- jednowątkowy model interfejsu użytkownika

Kolejka komunikatów i Looper



Grafika pochodzi ze strony: http://www.aviyehuda.com/2010/12/android-multithreading-in-a-ui-environment/

Poprawne implementowanie wielowątkowości

Dostęp do wątku interfejsu użytkownika z poziomu innych wątków

- Activity.**runOnUiThread**(Runnable r)
- boolean View.**post**(Runnable r)
- boolean View.**postDelayed**(Runnable r, long ms)
- **Handler**
- **AsyncTask**

Poprawne implementowanie wielowątkowości - przykład

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap b = loadImageFromNetwork();
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(b);
                }
            });
        }
    }).start();
}
```


Przykład pochodzi z: http://developer.android.com/resources/articles/painless-threading.html

O interfejsach na emulatorach

Network Address	Description
10.0.2.1	Router/gateway address
10.0.2.2	**Specjalny alias do pętli zwrotnej interfejsu hosta (127.0.0.1)**
10.0.2.3	First DNS server
10.0.2.4 / 10.0.2.5 / 10.0.2.6	Optional second, third and fourth DNS server (if any)
10.0.2.15	**The emulated device's own network/ethernet interface**
127.0.0.1	The emulated device's own loopback interface

Tabela pochodzi z: http://developer.android.com/guide/developing/devices/emulator.html

O interfejsach na emulatorach

- Konsola windows:

C:/> *telnet localhost* <***port-serwera***>

redir add <*protokół*>:<*port-hosta*>:<*port-nasłuchu*>

TCP/UDP/...

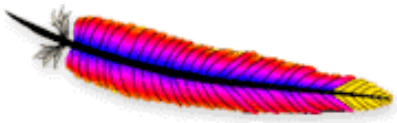
Z którym ma łączyć się klient

Na którym nasłuchuje serwer

Serwer: **10.0.2.15**:<**port-nasłuchu**>
Klient: **10.0.2.2**:<**port-hosta**>

Klasa HttpClient
Wprowadzenie

- Klasa służąca do obsługi protokołu HTTP.
- Alternatywa dla HttpURLConnection.
- Podstawowe funkcje:
  Metody POST, GET, PUT, DELETE...
  Obsługa plików cookies
  Uwierzytelnianie



Apache Commons
http://commons.apache.org/

Klasa HttpClient
Najprostszy przykład użycia

```
HttpClient httpclient = new DefaultHttpClient();
try {     // metoda GET
    HttpGet httpget = new HttpGet("http://www.google.com/");

    ResponseHandler<String> responseHandler;
    responseHandler = new BasicResponseHandler();
    // wysłanie zapytania
    String responseBody = httpclient.execute(httpget, responseHandler);

    System.out.println(responseBody);
} finally {
    httpclient.getConnectionManager().shutdown();  //zwolnienie
zasobów
}
```

Klasa HttpClient
Podsumowanie

1. Klasa umożliwia wysyłanie zapytań HTML, odbieranie zawartości wynikowej; obsługę plików cookies; uwierzytelnianie.

2. Dostępny kod źródłowy.

Przesyłanie pliku z użyciem gniazda Wprowadzenie

Założenia:

- obsługa gniazda wykonywana w osobnym wątku.
- obsługa UI wykonywana przy pomocy klasy Handler (kolejka).
- przesłanie pliku do klienta.

Przesyłanie pliku z użyciem gniazda
Klasa gniazda

```
public class ServerThread implements Runnable {
    public void run() {
        try {
            // otworzenie gniazda
            serverSocket = new ServerSocket(serverPort);
            while(true)  {
                // oczekiwanie na klienta
                Socket client = serverSocket.accept();

                // … obsługa strumieni …
            }
          catch (Exception e)  { }
    }
}
```

Przesyłanie pliku z użyciem gniazda Obsługa UI

```
Handler handler = new Handler();
serverStatus = (TextView)findViewById(R.id.statustv);

handler.post(new Runnable() {
    public void run() {
        serverStatus.setText("Listening: " + serverIp);
    }
});
```

Obiekt typu Handler wysyła instrukcje do kolejki, przesyłającej je dalej do wątku GUI.

Przesyłanie pliku z użyciem gniazda Wysłanie pliku

```
try {
    File myFile = new File ("/system/media/audio/ui/VideoRecord.ogg");
    byte [] mybytearray  = new byte [(int)myFile.length()];
    FileInputStream fis = new FileInputStream(myFile);
    BufferedInputStream bis = new BufferedInputStream(fis);
    // wczytanie pliku do tablicy
    bis.read(mybytearray,0,mybytearray.length);
    // pobranie strumienia wyjściowego z gniazda
    OutputStream os = client.getOutputStream();
    // wysłanie pliku do strumienia wyjściowego
    os.write(mybytearray,0,mybytearray.length);

} catch (IOException e) {}
```

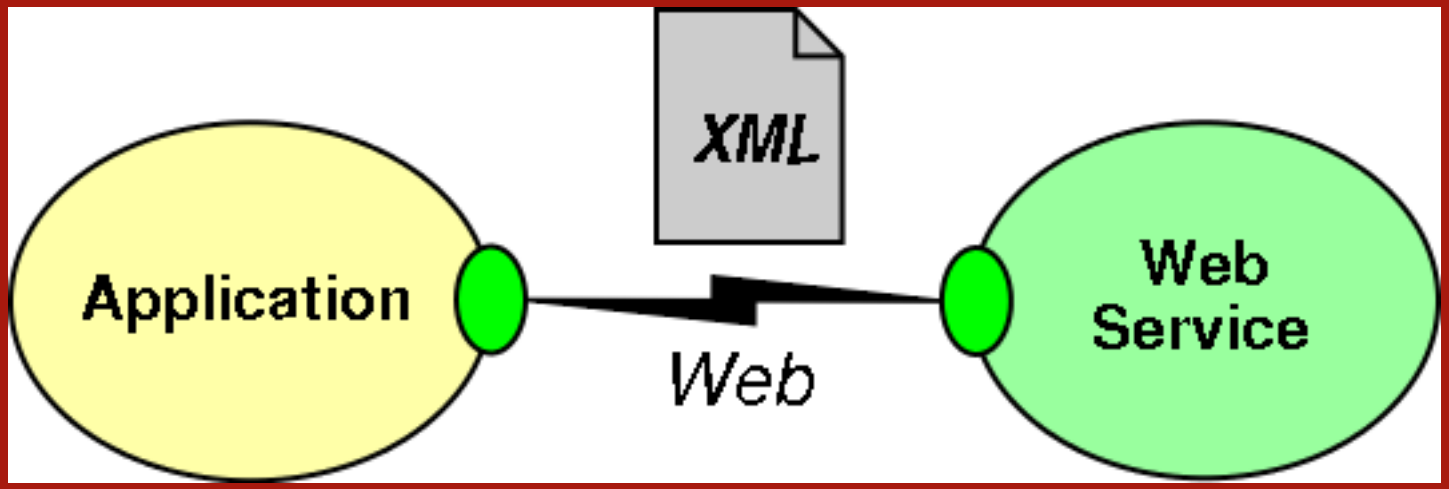
Przesyłanie pliku z użyciem gniazda Podsumowanie

Do zapamiętania:

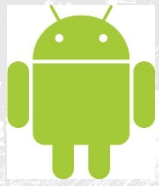
- aby ingerować w obiekty UI z obcych wątków, należy używać obiektu klasy Handler lub AsyncTask.
- cała reszta podobnie jak w standardowej Javie SE.

Politechnika Wrocławska

Web Services



Wiktor Ławski
Marcin Lubimow



Web Services – informacje ogólne

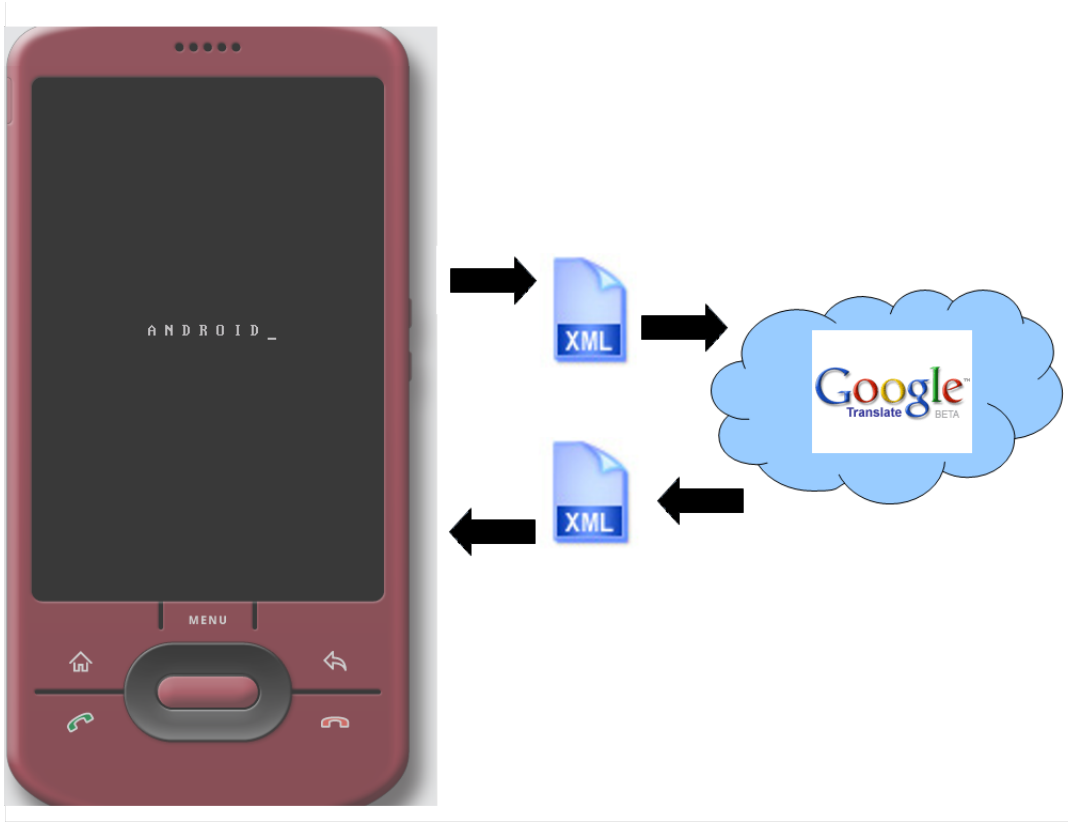
- usługa sieciowa jako aplikacja przeznaczona do współpracy różnych maszyn w sieci
- WSDL – język opisujący protokoły używane przez usługi sieciowe
- SOAP – lekki protokół przeznaczony do wymiany informacji
- XML – rozszerzalny język znaczników

Web Services - przykłady zastosowań

- Google: Google SOAP Search API (już niedostępne), Google Translate API, Google Maps API Web Services
- Amazon: Amazon Fulfillment Web Service (Amazon FWS), Amazon Simple Queue Service (Amazon SQS)

Sposób wykorzystania Web Services

- Google Translate API



googleTranslateExample - działanie

- wczytanie frazy i uruchomienie nowego wątku AsyncTask
- wygenerowanie zapytania URL
- otwarcie połączenia i wysłanie zapytania
- odczytanie otrzymanej odpowiedzi (łańcuch w formacie XML)
- parsowanie odpowiedzi
- wyświetlenie frazy wynikowej

googleTranslateExample - działanie

- wczytanie frazy i uruchomienie nowego wątku AsyncTask
- wygenerowanie zapytania URL
- otwarcie połączenia i wysłanie zapytania
- odczytanie otrzymanej odpowiedzi (łańcuch w formacie XML)
- parsowanie odpowiedzi
- wyświetlenie frazy wynikowej

Wczytanie frazy i uruchomienie nowego wątku AsyncTask

```
public void translate(View v) {

 EditText etext =
(EditText)findViewById(R.id.editText1);

TextView tview =
(TextView)findViewById(R.id.textView1);

new TranslateTask(tview).execute(
    etext.getText()
        .toString());
   }
```

googleTranslateExample - działanie

- wczytanie frazy i uruchomienie nowego wątku AsyncTask
- wygenerowanie zapytania URL
- otwarcie połączenia i wysłanie zapytania
- odczytanie otrzymanej odpowiedzi (łańcuch w formacie XML)
- parsowanie odpowiedzi
- wyświetlenie frazy wynikowej

Wygenerowanie zapytania URL

```
String q = URLEncoder.encode(arg0[0], "UTF-8");

URL url = new URL(
"https://www.googleapis.com/language/translate/v2?
key=YOUR-KEY&q="
+ q +
"&source=en&target=pl"
                        );
```

googleTranslateExample - działanie

- wczytanie frazy i uruchomienie nowego wątku AsyncTask
- wygenerowanie zapytania URL
- otwarcie połączenia i wysłanie zapytania
- odczytanie otrzymanej odpowiedzi (łańcuch w formacie XML)
- parsowanie odpowiedzi
- wyświetlenie frazy wynikowej

Otwarcie połączenia i wysłanie zapytania

```
HttpURLConnection con =
    (HttpURLConnection) url.openConnection();

con.setReadTimeout(10000 /* milliseconds */);
con.setConnectTimeout(15000 /* milliseconds */);
con.setRequestMethod("GET");
con.setDoInput(true);
con.connect();
```

googleTranslateExample - działanie

- wczytanie frazy i uruchomienie nowego wątku AsyncTask
- wygenerowanie zapytania URL
- otwarcie połączenia i wysłanie zapytania
- odczytanie otrzymanej odpowiedzi (łańcuch w formacie XML)
- parsowanie odpowiedzi
- wyświetlenie frazy wynikowej

Odczytanie otrzymanej odpowiedzi (łańcuch w formacie XML)

```
InputStreamReader reader =
    new InputStreamReader(
      con.getInputStream(), "UTF-8");

while((temp = reader.read()) != -1) {
    c = (char)temp;
    result += c;
                }

reader.close();
```

googleTranslateExample - działanie

- wczytanie frazy i uruchomienie nowego wątku AsyncTask
- wygenerowanie zapytania URL
- otwarcie połączenia i wysłanie zapytania
- odczytanie otrzymanej odpowiedzi (łańcuch w formacie XML)
- parsowanie odpowiedzi
- wyświetlenie frazy wynikowej

Przykładowa odpowiedź

```
{
    "data": {
        "translations": [
        {
            "translatedText": "bruk"
        }
        ]
    }
}
```

Parsowanie odpowiedzi

```
JSONObject jsonObject = new JSONObject(result);

int trans = jsonObject.getJSONObject("data")
.getJSONArray("translations").length();

for(int i = 0; i < trans; i++) {
    result =
        jsonObject.getJSONObject("data").
        getJSONArray("translations").getJSONObject(i)
        .getString("translatedText");
    if(i != trans - 1)
        result += "\n";
                }
```

googleTranslateExample - działanie

- wczytanie frazy i uruchomienie nowego wątku AsyncTask
- wygenerowanie zapytania URL
- otwarcie połączenia i wysłanie zapytania
- odczytanie otrzymanej odpowiedzi (łańcuch w formacie XML)
- parsowanie odpowiedzi
- wyświetlenie frazy wynikowej

Wyświetlenie frazy wynikowej

```
@Override
protected void onPostExecute(String result) {
    super.onPostExecute(result);
    resulttv.setText(result);
}
```

Podsumowanie

- model bezpieczeństwa wymuszający użycie Content Providerów
- unikalny URI identyfikujący każdy Content Provider
- Content Provider zwraca dane w postaci tabel
- wymagane wpisy (permission) w manifeście do komunikacji z Content Providerem

Podsumowanie

- konieczna wielowątkowość
- jednowątkowy model obsługi UI
- Web Services jako rozwiązanie wieloplatformowe

Literatura - Content Providers

- Developer's Guide – Content Providers:
  http://developer.android.com/guide/topics/providers/content-providers.html
- About URI:
  https://secure.wikimedia.org/wikipedia/en/wiki/Uniform_Resource_Identifier
- Package - android.providers:
  http://developer.android.com/reference/android/provider/package-summary.html

Literatura - Communicating via the Internet

- Painless Threading | Android Developers:
  http://developer.android.com/resources/articles/painless-threading.html
- Interconnecting Emulator Instances:
  http://developer.android.com/guide/developing/devices/emulator.html#connecting

Literatura - Web Services cz. 1 z 2

- Web Service Glossary: http://www.w3.org/TR/ws-gloss/
- Konrad Kunicki: Web Services – Technologie biznesu elektronicznego. Wrocław 2005: http://www.e-informatyka.pl/sens/attach/Webservices260405/webservices260405.pdf
- Google SOAP Search API: http://code.google.com/intl/pl-PL/apis/soapsearch/reference.html
- Developer's Guide (v2): Getting Started - Google Translate API - Google Code: http://code.google.com/intl/pl-PL/apis/language/translate/v2/getting_started.html
- Google Maps API Web Services: http://code.google.com/intl/pl-PL/apis/maps/documentation/webservices/
- Amazon Web Services: http://aws.amazon.com/
- Ed Burnette: Hello, Android. Introducing Google's Mobile Development Platform. United States of America 2008 (str. 146-156)

Literatura - Web Services cz. 2 z 2

- Manifest.permission | Android Developers:
  http://developer.android.com/reference/android/Manifest.permission.html
- AsyncTask | Android Developers:
  http://developer.android.com/reference/android/os/AsyncTask.html
- HttpURLConnection | Android Developers:
  http://developer.android.com/reference/java/net/HttpURLConnection.html

Grafiki

Pozostałe grafiki pochodzą z następujących źródeł:

- http://www.w3.org/2005/Talks/1114-hh-ecows/#(1)
- http://www.ha-systems.pl/mobiledit.html

Politechnika Wrocławska

Pytania?

Politechnika Wrocławska

Content Providers
Komunikacja przez internet
Web Services

Wiktor Ławski
Marcin Lubimow
Mateusz Dziekan
Marcin Filip
Pod opieką dra Pawła Rogalińskiego