Polsko-Japońska Wyższa Szkoła
Technik Komputerowych

Java w Urządzeniach Mobilnych

Autor:
mgr inż. Michał Tomaszewski

2 czerwca 2010

Spis treści

1 Wprowadzenie

W roku 1990 Patrick Naughton, Mike Sheridan i James Gosling rozpoczęli prace nad finansowanym przez firmę SUN ściśle tajnym projektem o kryptonimie „The Green". Celem było zbadanie rynku komputerowego i określenie trendów, na które firma powinna zwrócić szczególną uwagę w najbliższej przyszłości. Dość szybko specjaliści z „Green Team" przeprowadzili niezbędne badania. Z uzyskanych danych wynikało, iż wkrótce istotną częścią rynku stanie się segment cyfrowo sterowanych urządzeń powszechnego użytku. Na nim więc skoncentrowały się wysiłki inżynierów [1]. Zgodnie z deklaracjami twórców tego programu ważne były dwie rzeczy: po pierwsze model biznesowy i produkt końcowy, po drugie zaś technologia wykonania. Między innymi z tego powodu liczba osób biorących udział w projekcie „The Green" stale rosła. I tak w kwietniu roku 1991 do zespołu dołączyły trzy nowe osoby: Chris Warth, Ed Frank oraz Craig Forest.

Efektem prac grupy było urządzenie nazwane *Star7*, wyposażone w :

- pięciocalowy, kolorowy, ciekłokrystaliczny wyświetlacz dotykowy
- bezprzewodowe złącze sieciowe o częstotliwości 900 MHz
- interfejs PCMCIA (ang. Personal Computer Memory Card International Association)
- kodek audio i video

Latem 1992 prace nad tym urządzeniem zostały zakończone i 3 sierpnia ich efekt zaprezentowano kierownictwu firmy. Osoby biorące udział w prezentacji były zachwycone zarówno wyglądem jak i funkcjonalnością produktu. Jednak pomimo bardzo pochlebnych opinii Star7 nigdy nie wdrozono do produkcji (rys. 1.1).

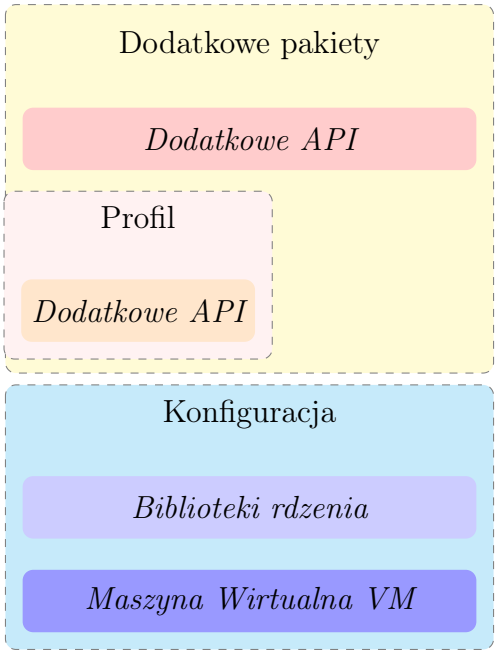
Rysunek 1.1: *Urządzenie *7*

Trzy lata później, w marcu 1995 roku upubliczniono wstępną wersję specyfikacji nowego języka – *Java*. W ciągu kolejnych lat jego rozwój doprowadził do powstania całej rodziny produktów bazujących na wspólnej składni, wśród których wyróżnia się:

- *Java 2 Standard Edition*,
- *Java Enterprise Edition*,
- *Java Micro Edition*.

1.1 Java Micro Edition

Ponieważ spektrum urządzeń dla jakich przeznaczona jest *Java Micro Edition* (*J2ME*) jest bardzo szerokie, architektura musi być elastyczna. Z tego powodu została określona w kontekście *konfiguracji* i *profili*.

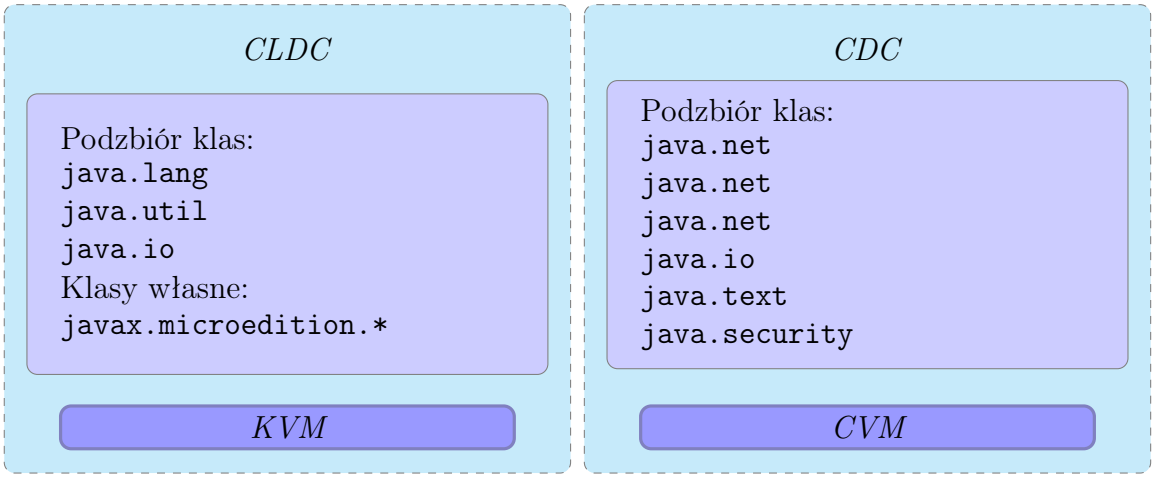


Rysunek 1.2: *Koncepcyjny rozkład warstw w architekturze J2ME*

Jak pokazuje rysunek 1.2, maszyna wirtualna jest częścią *konfiguracji*, natomiast *profil* jest już rozpatrywany jako element dodatkowy. Obecnie dysponujemy dwoma rodzajami konfiguracji:

- *CLDC* (*Connected Limited Device Configuration*) dla urządzeń o ograniczonych zasobach (zaliczamy tu między innymi pagery, telefony komórkowe czy komputery naręczne)
- oraz *CDC* (*Connected Device Configuration*) dla urządzeń o możliwościach większych niż typowe PDA, aczkolwiek nie dorównujących komputerom osobistym.

Każda z wymienionych *konfiguracji* zawiera własną maszynę wirtualną, a ponadto implementuje inny zestaw klas bazowych. Należy zaznaczyć, że w obu przypadkach jest to podzbiór klas zawartych w *J2SE*.



Rysunek 1.3: *Warstwa konfiguracji*

Podział, który obowiązuje dla *konfiguracji* przekłada się na *profile*. I tak w *konfiguracji CLDC* wyróżniano *profil MIDP* oraz *PDAP*. Odpowiednio *konfiguracja CDC* wyróżnia *Foundation Profile, Personal Profile* oraz *RMI Profile*. Szerzej o *konfiguracjach* i *profilach* traktuje rozdział 2, poruszający tematykę wymagań, ograniczeń i zależności pomiędzy obiema warstwami.

1.2 Oprogramowanie

Proces tworzenia oprogramowania, które może zostać uruchomione na platformach mobilnych, wymaga instalacji niezbędnych programów. Za podstawowe można uznać środowisko kompilacyjne pozwalające na analizę leksykalną, składniową i semantyczną plików źródłowych. Jako że tworzone będą aplikacje w języku *Java*, naturalnym wyborem jest oprogramowanie firmy SUN – Java SE Development Kit (JDK)[9].

Następnie zadbać należy o biblioteki, K maszynę wirtualną i programy wspomagające proces tworzenia aplikacji *J2ME*. SUN oferuje na swoich stronach również Java Platform Micro Edition Software Development Kit [8], który umożliwi zarówno rozpoczęcie pracy jak i pozwoli na uruchamianie programów przy pomocy emulatora różnych typów urządzeń mobilnych. Przez długi czas wadą tego produktu była ograniczona liczba dostępnych modeli urządzeń, od wersji 3.0 gamę modeli znacznie poszerzono.

Osoby nieusatysfakcjonowane modelami oferowanymi domyślnie, mogą poszukać stosownego oprogramowania w zasobach sieci globalnej. Ciekawym punktem wyjścia dla samodzielnych poszukiwań może być artykuł 'MIDP Emulators' [10].

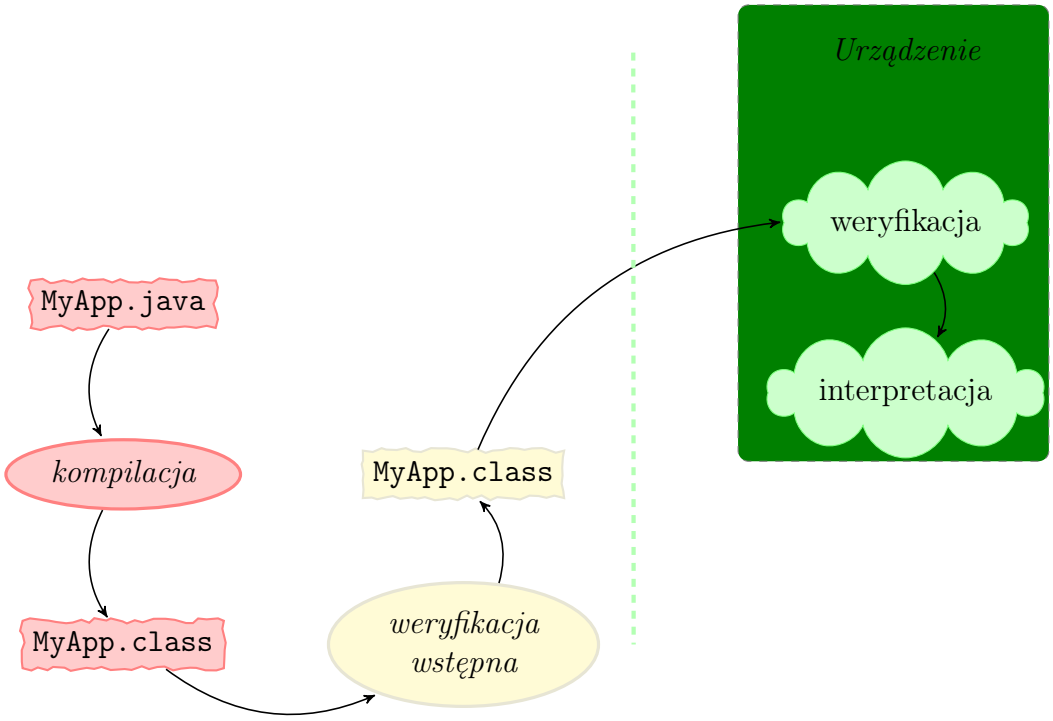
Instalacja wymienionego powyżej oprogramowania pozwoli już na rozpoczęcia procesu tworzenia aplikacji. Jednak pisanie programów przy pomocy zwykłego edytora tekstowego słusznie może zostać uznane za proces czasochłonny i żmudny, dlatego wskazane jest korzystanie z dedykowanego środowiska edycyjnego takiego jak *Eclipse*, wzbogaconego o rozszerzenia wspomagające tworzenie aplikacji mobilnych np.: *Mobile Tools for the Java Platform*.

1.2.1 Proces tworzenia aplikacji

Tworzenie aplikacji w środowisku *Java* sprowadza się do wydania jednego polecenia. W przypadku *J2ME* proces ten jest znacznie bardziej skomplikowany, składają się na niego:

- utworzenie programu,
- kompilacja,
- weryfikacja wstępna,
- *umieszczenie aplikacji w urządzeniu*,
- weryfikacja właściwa,
- interpretacja.

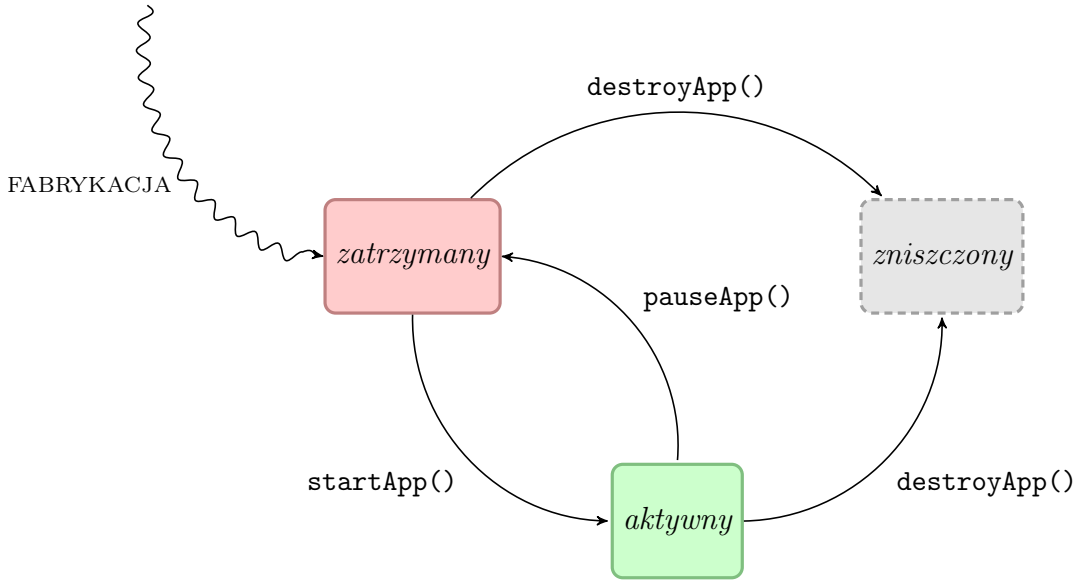
Wszystkie kroki schematu projektowego przedstawia rysunek 1.4, a kolejne rozdziały szczegółowo opisują detale procesu.



Rysunek 1.4: *Cykl tworzenia aplikacji.*

1.2.2 Pierwszy program

Aplikacje przeznaczone dla profilu MIDP są nazywane MIDletami. Podobnie jak aplety są to aplikacje zagnieżdżone w środowisku zarządzającym zasobami urządzenia. Natomiast w odróżnieniu od apletów programem zarządzającym nie jest przeglądarka, a specjalna aplikacja wbudowana w urządzenie, określana mianem *AMS* (*Application Managment Software*). Głównym elementem MIDletu jest klasa dziedzicząca po publicznej i abstrakcyjnej klasie `javax.microedition.midlet.MIDlet`. Ponieważ odziedziczona klasa zawierała metody abstrakcyjne, programista będzie zmuszony do implementacji ich ciał. Będą to `startApp()`, `pauseApp()` i `destroyApp()`. Wymienione metody opisują trzy stany aplikacji w jej cyklu życiowym a ich wizualizację przedstawiono na rysunku 1.5.



Rysunek 1.5: *Cykl życia aplikacji MIDletowych.*

Kod programu

Wiedząc, że uruchomienie MIDletu rozpocznie się od wywołania klasy dziedziczącej po `MIDlet`, można stworzyć przykładowy kod jak w listingu 1.1.

```
package tomPack;

import javax.microedition.midlet.*;

public
    class HelloMidlet
    extends MIDlet{

    public HelloMidlet(){
    }

    public void startApp(){
    }

    public void pauseApp(){
    }

    public void destroyApp(boolean unconditional){
    }
}
```

Listing 1.1: *Podstawowy kod programu MIDlet*

Kompilacja

Po zapisaniu programu opisanego jako listing 1.1 należy poddać go procesowi kompilacji. Operacja ta wymaga określenia klas stanowiących bazę aplikacji, dlatego wykorzystano parametr `-bootclasspath`.

```
javac -d . -classpath . -bootclasspath "C:\Java_ME_platform_SDK_3.0\lib\cldc_1.0.↩
    jar;C:\Java_ME_platform_SDK_3.0\lib\midp_1.0.jar" *.java
```

W wyniku kompilacji kodu programu powstanie bajtkod przechowywany w pliku o rozszerzeniu `*.class`.

Weryfikacja wstępna

Skompilowany bajtkod należy następnie poddać *wstępnej weryfikacji*. Krok ten został wprowadzony do procesu tworzenia aplikacji ponieważ zasoby urządzeń warstwy *CLDC* mogą okazać się nie wystarczające dla przeprowadzenia *weryfikacji* znanej z *J2SE*. Szerzej opisano to zagadnienie w rozdziale 2.2.1. Innym, lecz nie mniej istotnym powodem implementacji tego mechanizmu był fakt, iż *CLDC 1.0* nie przewidywało operacji zmiennoprzecinkowych, zatem umieszczenie typów `float` czy `double` powinno być sygnalizowane.

Operacja *weryfikacji wstępnej* wymaga uruchomienia dedykowanego programu o nazwie `preverify.exe`, dostarczanego przez *J2ME SDK*. Należy zwrócić uwagę aby właściwie określić ścieżkę wyjściową (parametr –d) i lokalizację klas rdzenia (`-classpath`). Przykładowe wywołanie ma postać 1.2.

```
C:\Java_ME_platform_SDK_3.0\bin\preverify -d C:\J2ME\output -classpath "C:\↩
    Java_ME_platform_SDK_3.0\lib\cldc_1.0.jar;C:\Java_ME_platform_SDK_3.0\lib\↩
    midp_1.0.jar" tomPack.HelloMidlet
```

Listing 1.2: *Użycie programu* `preverify.exe`

Działanie programu *weryfikacji wstępnej* skutkuje modyfikacją bajt kodu poprzez umieszczenie w nim dodatkowego atrybutu *stack maps*, opisującego zmienne *MIDletu* i argumenty operacji umieszczanych na stosie interpretera.

Pakietowanie

Efekt *weryfikacji wstępnej* należy połączyć w pojedyncze archiwum `*.jar`, na które będą składać się:

1. pliki aplikacji,
2. multimedia *(pliki graficzne, wideo czy audio)*,
3. plik manifestu.

Aby *zarządca aplikacji* w urządzeniu pozwolił na załadowanie pilku `jar` należy uzupełnić szereg atrybutów, podając:

- nazwę MIDletu,
- numer wersji aplikacji,
- 'producenta',
- klasę uruchomieniową,
- konfigurację,
- profil.

Przykładowo wypełniony plik manifestu zaprezentowano w listingu 1.3.

```
1 MIDlet-Name: FirstMIDlet
2 MIDlet-Version: 1.0
3 MIDlet-Vendor: Empty
4 MIDlet-1: FirstMIDlet, ,tomPack.HelloMIDlet
5 MicroEdition-Configuration: CLDC-1.0
6 MicroEdition-Profile: MIDP-1.0
```

Listing 1.3: *Przykładowa treść pliku MANIFEST.MF*

Opis wszystkich atrybutów manifestu można znaleźć w załączniku (***).

Dysponując wynikiem *weryfikacji wstępnej* i spreparowanym plikiem manifestowym można posłużyć się programem narzędziowym `jar.exe` dostarczonym wraz z pakietem *JDK*. Jego przykładowe wywołanie, które ma na celu utworzenie archiwum `test.jar` ma postać[18]:

```
jar cfm test.jar MANIFEST.MF -C output .
```

Przygotowanie pliku uruchomienia `*.jad`

```
1 MIDlet-Name: FirstMIDlet
2 MIDlet-Vendor: Organization
3 MIDlet-Version: 1.0
4 MIDlet-Jar-URL: test.jar
5 MIDlet-Jar-Size: 1123
```

Listing 1.4: *Przykładowa treść pliku* `jadtest.jad`

Uruchomienie

Tak opracowany program można uruchomić na emulatorze dostarczonym przez firmę SUN.

Uruchomienia emulatora dokonuje się poleceniem:

```
emulator.exe  -Xdevice:DefaultColorPhone -Xdescriptor:jadtest.jad
```

Komenda ta skutkuje wyświetleniem okna z emulacją telefonu komórkowego.

Rysunek 1.6:

1.2.3 Emulacja na urządzeniu Palm

Dysponując tak przygotowanym archiwum można zamienić je na aplikację przeznaczoną dla PalmPilota.

```
java.exe -cp "D:\jdk\j2me\MIDP\midp4palm1.0\Converter\Converter.jar"  com.sun.↩
    midp.palm.database.MakeMIDPApp  v  v  name "Comm test"  outfile OutFirst.prc ↩
     creator 604 "F:\J2ME test1\output\first.jar"
```

Rysunek 1.7: *Aplikacja na emulatorze Palm*

1.2.4 Dodatkowe oprogramowanie

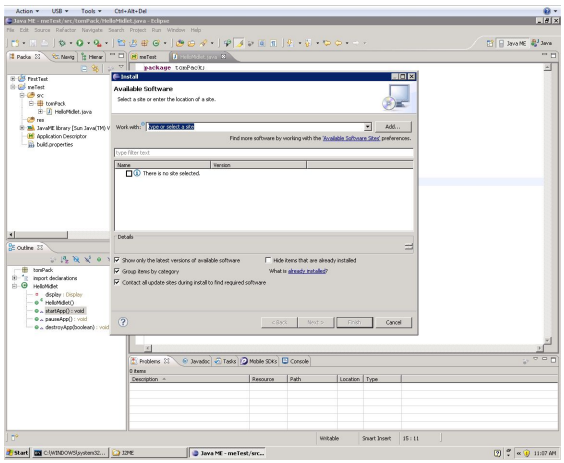
Przedstawione powyżej operacje doprowadzą użytkownika do funkcjonalnej aplikacji, która może zostać umieszczona na urządzeniu docelowym. Zapewne jednak mało który programista zdecyduje się na tworzenie oprogramowania w taki sposób ze względu na brak popularnych udogodnień, jak automatyczne uzupełnianie metod, kontrola typów i składni itp. Dla zwiększenia komfortu projektowania można skorzystać z dodatkowego oprogramowania:

- Eclipse [4] - określany mianem platformy developerskiej,
- Mobile Tools for Java [11] - dodatek dla środowiska Eclipse wykonujący operacje opisane w punktach 1.2.4.

Eclipse i MTJ

Instalacja środowiska Eclipse jest realatywnie prosta, ponieważ sprowadza się do rozpakowania archiwum, uruchomienie programu wymaga zainstalowanej *Javy*.

Rozszerzenie Eclipsa o dodatek *MTJ* rozpoczyna się od wybrania opcji *Install new software...* z menu *Help*, powinno pojawić się okno jak na rysunku 1.8

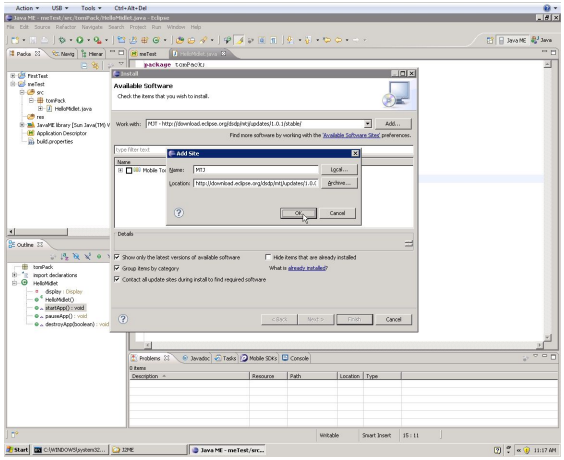


Rysunek 1.8: *Okno Install w środowisku Eclipse.*

Kolejnym krokiem będzie wybranie opcji *Add*, która pokaże okno proszące o wypełnienie:

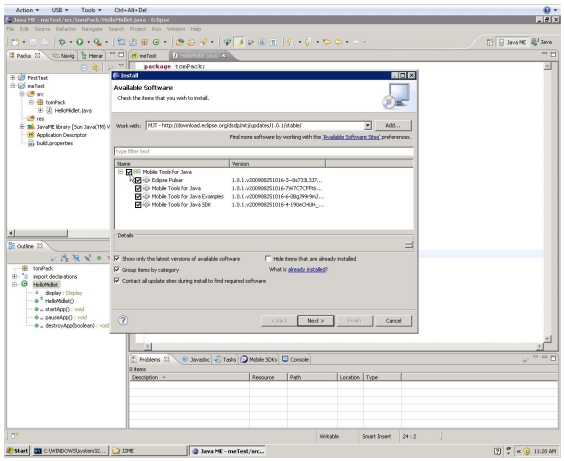
- *Name* - nazwy indentyfikującej źródło update'u,
- *Location* - adres url

Podkreślić należy, że o ile *nazwa* może być dowolnym ciągiem znaków, to *adres* musi pochodzić ze strony o strukturze ściśle określonej przez Eclipsa. W przypadku dodatku *MTJ* może to być: `http://download.eclipse.org/dsdp/mtj/updates/1.0.1/stable/`



Rysunek 1.9: *Okno Dodania nowego adresu.*

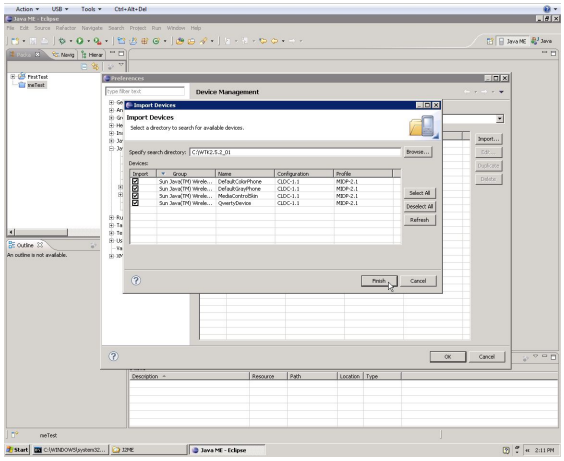
Akceptacja wprowadzonych danych spowoduje podjęcie próby połączenia i pozyskania informacji o dodatku. Jeżeli ta część operacji przebiegnie poprawnie, wówczas na ekranie pojawi się drzewo przedstawiające składowe rozszerzenia, jak jest to widoczne na ilustracji 1.10.



Rysunek 1.10: *Drzewo dodatku.*

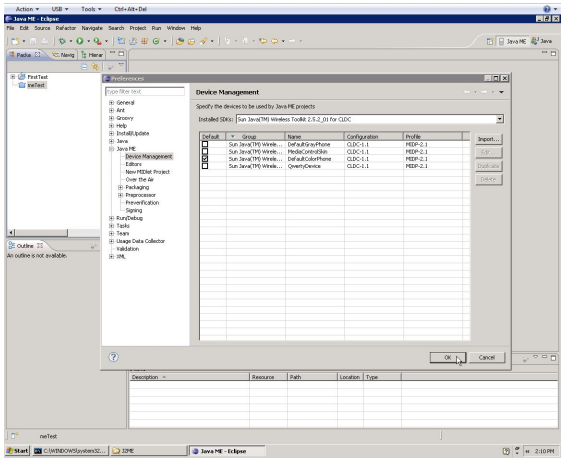
Zaznaczenie dodatku który ma zostać zainstalowany aktywuje przycisk *Next* pozwalający na przejście do kolejnego etapu instalacji. Środowisko dokona analizy, sprawdzając które z zaznaczonych komponentów są już zainstalowane a które wymagają dodatkowych pakietów. Po zakceptowaniu niezbędnych zależności i wciśnięciu przycisku *Finish* nastąpi automatyczna instalacja a w dalszej kolejności prośba o akceptację ponownego uruchomienia środowiska Eclipse.

Kolejnym krokiem będzie powiązanie modułu *MTJ* z *WTK*. W tym celu należy wybrać w środowisku programistycznym opcje *Preferences* z menu *Window*, otrzymując okno konfiguracji jak na rysunku 1.11.



Rysunek 1.11: *Okno ustawień.*

Spośród gałęzi wyświetlonego drzewa należy rozwinąć *Java ME* a następnie zaznaczyć *Device Menagment*. Rozwijana lista *Installed SDKs* będzie domyślnie pusta, dlatego używając przycisku *Import* należy wskazać ścieżkę do *WTK*, zaznaczając urządzenia które będą wykorzystywane w tworzonych aplikacjach. Po akceptacji pozostaje już tylko wskazanie, które z urządzeń powinny być wykorzystywane w projektach. Efekt został przedstawiony na rysunku 1.12.



Rysunek 1.12: *Ekran wyboru urządzeń.*

1.3 Kolejne aplikacje

Aplikacja przedstawiona w rozdziale 1.2 choć poprawnie funkcjonująca, była obarczona poważną wadą – nie można było zaobserwować żadnych efektów jej działania. Dlatego też kolejnym krokiem będzie utworzenie aplikacji komunikującej się z użytkownikiem. W tym celu zaimportowany zostanie dodatkowo pakiet `javax.microedition.lcdui.*`, w którego skład wchodzą komponenty `TextBox` oraz `Command`.

Konstruktor `TextBox(String title, String text, int maxSize, int constraints)` wymaga dostarczenia szeregu parametrów. Zmienna typu `String` będzie opisywała nazwę komponentu `TextBox`. Drugi parametr - `text` oczekuje podania ciągu znaków jaki będzie zawarty w tym komponencie. Kolejne parametry opisują dopuszczalny rozmiar ciągu znaków i nałożone ograniczenia. Finalnie realizacja będzie wyglądać w sposób następujący:

```
TextBox tb = new TextBox(
    "Name", "Hello World", 20, 0
);
```

Zaś cały program zamieszczono w listingu 1.5.

```
1  package tomPack;
2  import javax.microedition.midlet.*;
3  import javax.microedition.lcdui.*;
4
5  public
6  class HelloMidlet
7      extends MIDlet{
8      private Display display;
9      public HelloMidlet(){
10         display = Display.getDisplay(this);
11     }
12     public void startApp() {
13         TextBox tb = new TextBox(
14             "Name", "Hello World", 20, 0
15         );
16
17         display.setCurrent(tb);
18     }
19
20     public void pauseApp(){
21     }
22     public void destroyApp(boolean unconditional) {
23     }
24 }
```

Listing 1.5: *Program z* ***TextBoxem***

Niedociągnięciem pokazanego rozwiązania jest brak możliwości wyjścia z aplikacji. Aby to umożliwić wykorzystano obiekt klasy `Command`.

`Command(String label, int commandType, int priority)` konstruktor wymaga dostarczenia nazwy (`label`), rodzaju przycisku(`commandType`) i priorytetu. Ponadto należy obsłużyć ten przycisk, jednak pośród metod którymi dysponuje `Command` nie występuje dodanie odpowiedniego obiektu nasłuchującego. Stosowna metoda `setCommandListener` znajduje się natomiast w klasie `TextBox`, co umożliwia dodanie kilku przycisków które będą obsługiwane przez pojedynczą metodę.

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5
6  public
7  class HelloMidlet
8      extends MIDlet{
9      private Display display;
10
11     public HelloMidlet(){
12         display = Display.getDisplay(this);
13     }
14     public void startApp() {
15         TextBox tb = new TextBox(
16             "Name", "Hello World", 20, 0
17         );
18
19         Command command = new Command(
20             "Exit", Command.EXIT, 0
21         );
22         tb.addCommand(command);
23         tb.setCommandListener(
24             new CommandListener(){
25                 public void commandAction(Command c, Displayable s){
26                     notifyDestroyed();
27                 }
28             }
```

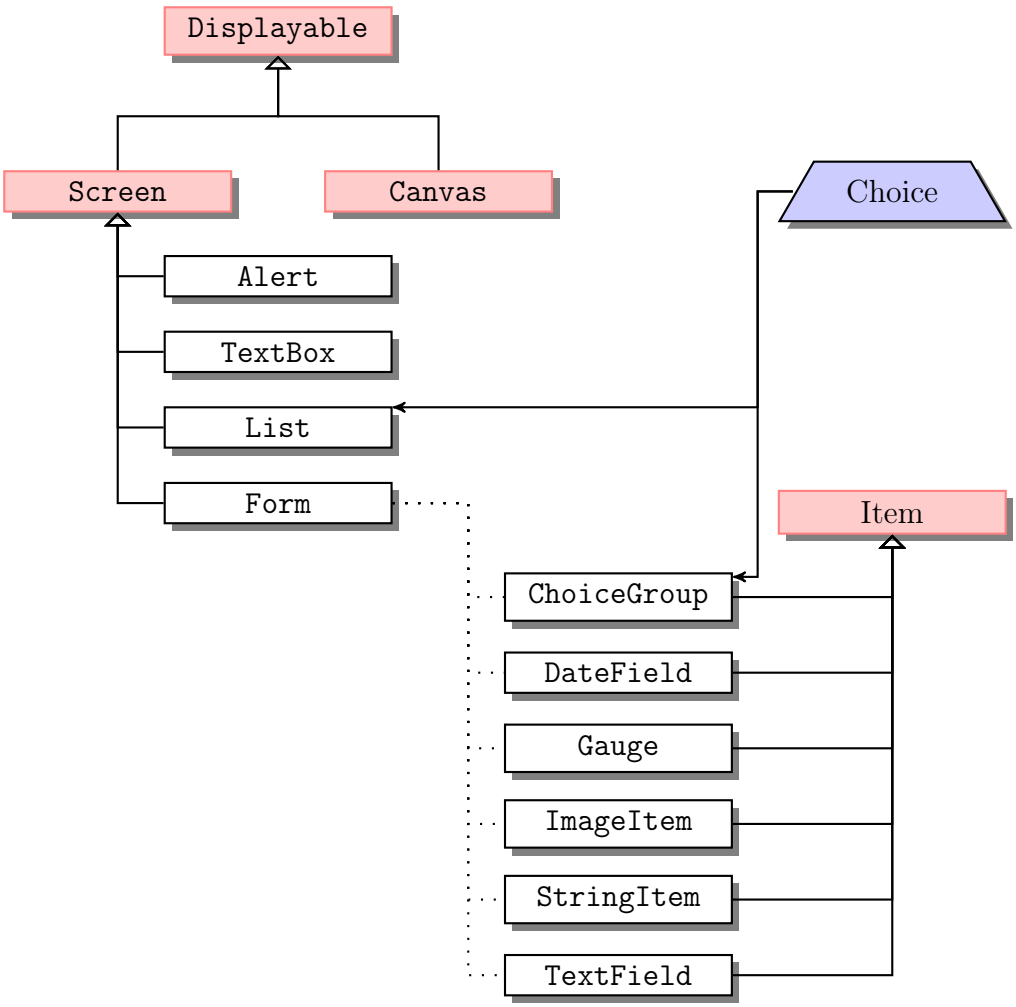
```
29          );
30
31          display.setCurrent(tb);
32      }
33
34      public void pauseApp(){
35      }
36      public void destroyApp(boolean unconditional) {
37      }
38  }
```

Listing 1.6: *Program z przyciskiem* `Command`

1.4 Podstawowe komponenty J2ME

Fizyczny wyświetlacz zamontowany np. w telefonie komórkowym jest odwzorowany w J2ME jako klasa Display. Znajduje się ona w pakiecie `javax.microedition.lcdui`, którego nazwę można rozszyfrować jako *LCD User Interface*. Wyświetlanie danych wymaga 'dostępu' do wyświetlacza, który otrzymuje MIDlet. Wykorzystując zatem statyczną metodę `getDisplay(Midlet)` można uzyskać obiekt reprezentujący wyświetlacz. Dysponując obiektem klasy `Display` programista może określić komponent który będzie wyświetlany. Zrobi to posługując się metodą `setCurrent(Displayable)`.

Widać więc, że wyświetlana może być każda klasa dziedzicząca po `Displayable`, zaś dokumentacja podpowiada że będą to klasy `Screen` lub `Canvas`. Z tego samego źródła możemy dowiedzieć się, że pierwsza z wymienionych klas jest postrzegana jako bazowa dla wszystkich wysokopoziomowych komponentów. Konsekwentnie, klasa `Canvas` będzie wykorzystywana do obsługi zdarzeń niskopoziomowych i graficznych.



Rysunek 1.13: *Schemat klas GUI w profilu MIDP*

Rysunek 1.13 odwzorowuje relacje pomiędzy istniejącymi klasami. Pokazuje on jednocześnie, że do dyspozycji programisty dostarczono kilka wysokopoziomowych komponentów.

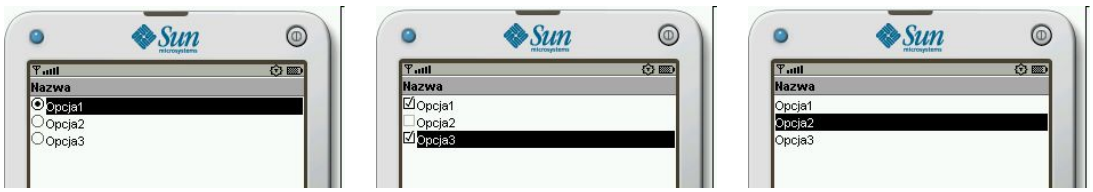
1.4.1 Klasa TextBox

Klasę `TextBox` wykorzystano w programach 1.5 i 1.6 , należy jednak wspomnieć że funkcjonalnością tego komponentu graficznego jest prezentacja i edycja ciągów znakowych.

1.4.2 Klasa List

Nieco bardziej skomplikowaną pod względem funkcjonalności jest klasa `List`. Jej celem jest prezentacja opcji, które mogą być przedstawione jako ciąg znaków, ikona lub połączenie tych dwóch elementów. Istnieją 3 typy list:

- `EXCLUSIVE` w dowolnym momencie może być wybrany tylko jeden element listy,
- `MULTIPLE` pozwalający na wybór kilku opcji,
- `IMPLICIT` stwierdzający że elementem zaznaczonym jest ten który otrzymał celownik (*fokus*).



Exclusive Multiple Implicit

Rysunek 1.14: *Wizualizacje listy.*

Jak widać na rysunku 1.14 w zależności od wybranego typu listy zmienia się jej wizualizacja. Wykorzystanie praktyczne zaprezentowano w listingu 1.7:

```
package tomPack;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public
class HelloMidlet
    extends MIDlet{

    private Display display;

    public void startApp() {

        display = Display.getDisplay(this);

        List list = new List("Nazwa", Choice.EXCLUSIVE);
        list.append( "Opcja1", null);
        list.append( "Opcja2", null);
        list.append( "Opcja3", null);

        display.setCurrent(
            list
        );
    }

    public void pauseApp(){
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Listing 1.7: *Wykorzystanie klasy* `List`

1.4.3 Klasa `Alert`

Klasa `Alert` została stworzona aby sygnalizować błędy lub zdarzenia.

```
Alert(String title, String alertText, Image alertImage, AlertType alertType)
```

Jej konstruktor składa się z informacji wyświetlonej w nagłówku (`title`), treści komunikatu (`alertText`), ilustracji (`alertImage`) i typu zdarzenia, przedstawionego jako obiekt klasy `AlertType`. Może wystąpić jeden z pięciu typów komunikatu:

- ALARM,
- CONFIRMATION,
- ERROR,
- INFO,

- WARNING.

Program pokazany w listingu 1.8 realizuje praktyczne połączenie list i komunikatów. Stosując listę jednokrotnego wyboru `EXCLUSIVE` dokonano klasyfikacji który z komunikatów powinien być wyświetlony. Akceptacja wybranej opcji skutkuje utworzeniem i wyświetleniem komunikatu.

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5
6  public
7  class HelloMidlet
8      extends MIDlet
9      implements CommandListener{
10
11     private Display display;
12     private String[] tab = {
13         "ALARM", "CONFIRMATION", "ERROR", "INFO", "WARNING"
14     };
15     private AlertType[] at = {
16         AlertType.ALARM, AlertType.CONFIRMATION, AlertType.ERROR,
17         AlertType.INFO, AlertType.WARNING
18     };
19     private Command command;
20     private List list;
21
22     public HelloMidlet(){
23         command = new Command( "Approve", Command.OK, 0);
24         list = new List("Alerts", Choice.EXCLUSIVE, tab, null);
25     }
26
27     public void startApp() {
28
29         display = Display.getDisplay(this);
30
31         list.addCommand(command);
32         list.setCommandListener(
33             this
34         );
35
36         display.setCurrent(
37             list
38         );
39     }
40
41     public void commandAction(Command c, Displayable d){
42         if(c == command){
43             int selected = list.getSelectedIndex();
44             Alert alert = new Alert(
45                 "Alert: " + tab[selected],
46                 "Test komunikatu " + tab[selected],
47                 null,
48                 at[selected]
49             );
50             display.setCurrent( alert, d);
51         }
52     }
53
54     public void pauseApp(){
55     }
56
57     public void destroyApp(boolean unconditional) {
58     }
59 }
```

Listing 1.8: *Zastosowanie klasy* `Alert`

Należy zauważyć, że w trakcie działania aplikacja 'odgrywa' dźwięki charakterystyczne dla danego błędu. Dlatego warto zwrócić uwagę na metodę `playSound(Display)` z klasy `AlertType`, za sprawą której realizowana jest ta ope-

racja.

Obserwując działanie aplikacji można zauważyć, że komunikat po pewnym czasie – bez żadnej ingerencji użytkownika – znika. Odpowiedzialna jest za to metoda `setTimeout(int)` klasy `Alert`, parametr `int` określa nieujemną ilość milisekund po których komunikat zostanie ukryty. Wartość tego parametru może przyjmować jeszcze jeden stan, opisany zmienną `Alert.FOREVER`.

1.4.4 Klasa Form

Wszystkie z przedstawionych do tej pory klas stanowią zamknięte kontenery, których wyglądu programista nie może modyfikować. Komponenty o samodzielnie zdefiniowanym wyglądzie tworzyć można dzięki klasie `Form` pozwalającej na łączenie komponentów dziedziczących po klasie `Item`.

W dokumentacji i na rysunku 1.13 można znaleźć następujące klasy spełniające powyższe wymaganie:

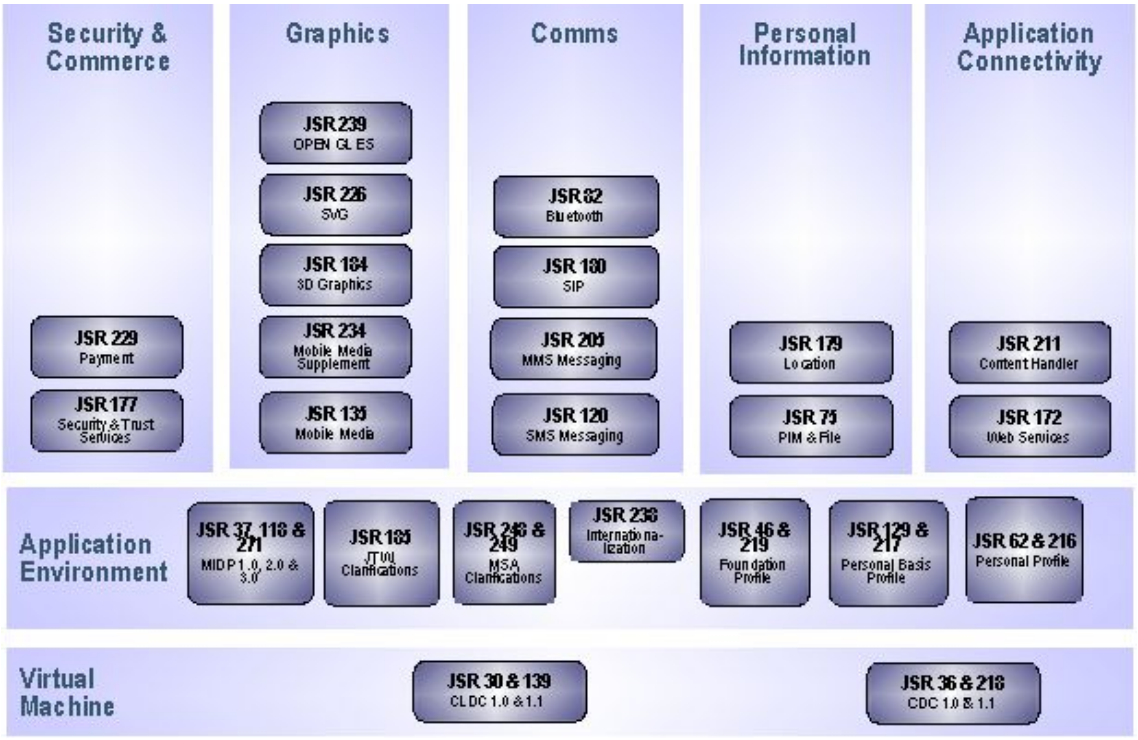
- ChoiceGroup,
- DateField,
- Gauge,
- ImageItem,
- StringItem,
- TextField.

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5
6  public
7  class HelloMidlet
8      extends MIDlet
9      implements CommandListener{
10
11     private Display display;
12
13     private Command command;
14     private Form form;
15     private ChoiceGroup choiceGroup;
16     private TextField textField;
17
18     public HelloMidlet(){
19
20         command = new Command( "OK", Command.OK, 0);
21         form = new Form("Form");
22         choiceGroup = new ChoiceGroup(
23             "DislayText ?",
24             Choice.MULTIPLE,
25             new String[]{ "Yes", "No"},
26             null
27         );
28         textField = new TextField(
29             "EnterText",
30             "", 20,
31             TextField.PASSWORD
32         );
33     }
34
35     public void startApp() {
```

```java

        display = Display.getDisplay(this);

        form.append(choiceGroup);
        form.append(textField);

        form.addCommand(command);
        form.setCommandListener(
            this
        );

        display.setCurrent(
            form
        );
    }

    private String[] tab = {
        "Non is selected",
        "",
        "",
        "YES or NO !!!"
    };

    public void commandAction(Command c, Displayable d){
        if(c == command){
            int wrt = 0;
            if( choiceGroup.isSelected(0))
                wrt += 1;
            if( choiceGroup.isSelected(1))
                wrt += 2;
            switch(wrt){
                case 0:
                    display.setCurrent(
                        new Alert(
                            "Alert",
                            tab[wrt],
                            null,
                            AlertType.ALARM
                        ), d
                    );
                    break;
                case 1:
                    display.setCurrent(
                        new Alert(
                            "Alert",
                            "Entered text:\n\"" + textField.getString()+"\"",
                            null,
                            AlertType.INFO
                        ), d
                    );
                    break;
                case 3:
                    display.setCurrent(
                        new Alert(
                            "Alert",
                            tab[wrt],
                            null,
                            AlertType.ALARM
                        ), d
                    );
                    break;
            }
        }
    }

    public void pauseApp(){
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

2 *MIDlety* i aplikacje

Chociaż korzenie języka *Java* (ówcześnie zwanego *Oak*) sięgają 1993 roku, a urządzenie *Star7* (rys. 1.1) do złudzenia przypomina spotykane obecnie komputery naręczne, to jednak za początek *J2ME* uważa sie rok 2000, ponieważ to właśnie wtedy oficjalnie zakończono prace nad pierwszymi wersjami tej technologii. Od tego czasu minęło 10 lat i rodzina API znacznie się poszerzyła.

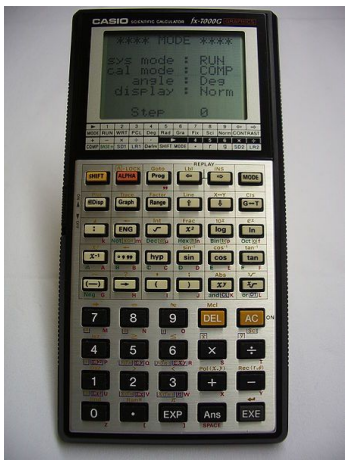


Rysunek 2.1: *Spis specyfikacji JSR dla J2ME*

Jak pokazuje rysunek 2.1 ilość i funkcjonalność *API* pozwala myśleć o tworzeniu bardzo zaawansowanych aplikacji.

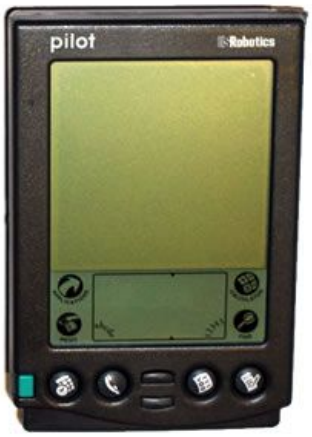
2.1 Zanim pojawiło się *J2ME*.

Pierwsze urządzenia, które można uważać za swego rodzaju komputery naręczne pojawiły sie już w latach 80-tych, były one narzędziami obliczeniowymi, ale posiadały ekran i pozwalały zapisywać własne programy.



Rysunek 2.2: *Kalkulator graficzny.[15]*

Uważa się że urządzenie *Palm* firmy U.S.Robotics jest twórczym rozwinięciem kalkulatorów, a jednocześnie jednym z pierwszych komputerów naręcznych.



Rysunek 2.3: *Kalkulator graficzny.[15]*

Palmtopy zajmowały znaczący segment rynku, również w Polsce. Tworzenie oprogramowania bazowało głównie na języku *C* i *C++*. Pojawiły się jednak produkty, takie jak *Waba Virtual Machine*, które pozwalały tworzyć oprogramowanie w języku o składni zbliżonej do *Javy*. [33]

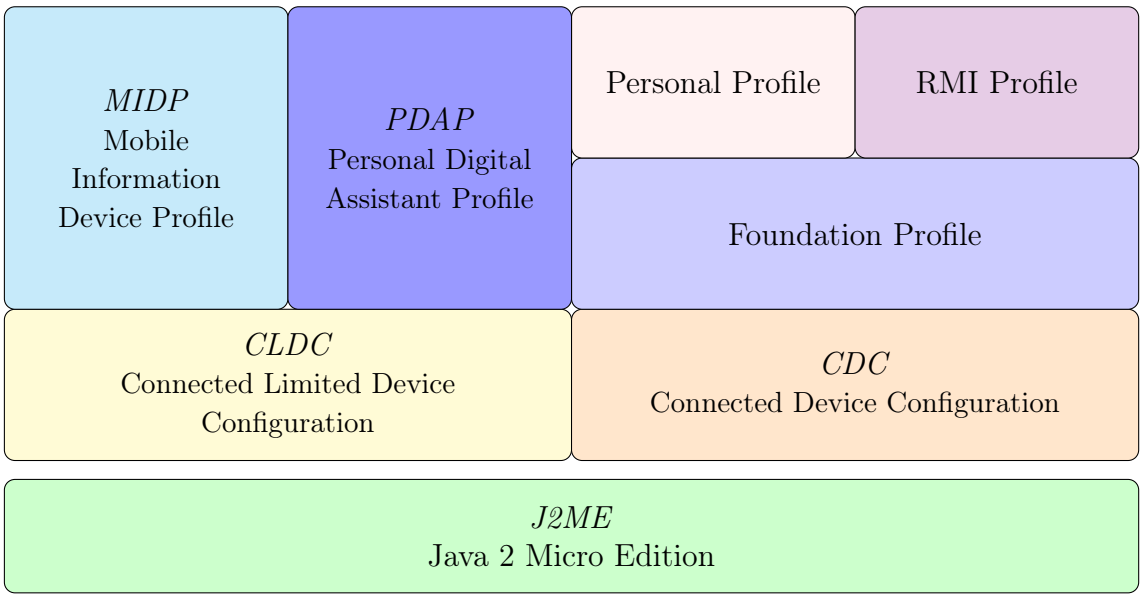
2.2 Architektura *J2ME*

Jak już wspomniano w rozdziale 1.1, *J2ME* obsługiwała bardzo szeroką gamę urządzeń, dlatego architektura musiała być elastyczna, tak jak pokazywał to schemat 1.3. Aby osiągnąć ten cel, wyróżniono warstwy *konfiguracji* i *profilu*.

Podziału na warstwy *konfiguracji* dokonano na podstawie możliwości sprzętowych urządzeń. Podstawowymi kryteriami były w tym wypadku moc obliczeniowa procesora oraz zasobność pamięci, w której należało zmieścić *maszynę wirtualną* i *klasy rdzenia*. W ten sposób wydzielono dwie podstawowe *konfiguracje*: *CLDC* i *CDC*.

Na bazie podziału *konfiguracyjnego* dokonano klasyfikacji *profili*. Zawierają one klasy pozwalające na implementację właściwości urządzeń.

Ustalony podział ilustruje schemat 2.4.



Rysunek 2.4: *Warstwy w J2ME.*

2.2.1 Konfiguracja dla urządzeń o ograniczonych zasobach

Specyfikacja *CLDC 1.0* opisuje następujące wymagania sprzętowe, stawiane przed urządzeniami [27, 25]:

- 128 KB pamięci nieulotnej, która zostanie przydzielona maszynie wirtualnej (*KVM*) i bibliotekom *CLDC*
- 32 KB pamięci ulotnej dedykowanej na potrzeby maszyny wirtualnej podczas uruchomienia programu
- 16 lub 32 bitowy procesor o taktowaniu minimum 25 MHz
- połączenie do wąskopasmowej sieci
- niskie zapotrzebowanie na energię

W praktyce wskazuje to rynek dla którego stworzona była konfiguracja *CLDC*, czyli: pagery, telefony ale również sprzęt audio/wideo czy czytniki kodów paskowych. Niestety ograniczenie zasobów pociągnęło za sobą konieczność ograniczenia funkcjonalności, dlatego:

- ograniczono funkcjonalność języka *Java* i maszyny wirtualnej,
- ograniczono ilość klas rdzenia (`java.lang` i `java.util`),
- zminimalizowano operacje wejścia/wyjścia (`java.io`),
- zminimalizowano operacje sieciowe (`javax.microedition.io`),
- zmodyfikowano model bezpieczeństwa,
- usunięto operacje zmiennoprzecinkowe.

Wymienione powyżej ograniczenia w wymierny sposób odcisnęły się na mechanizmach wykorzystywanych w *J2SE*. *CLDC* pozbawiony został:

- mechanimu refleksji,
- możliwości uruchamiania grup wątków i wątków tła,
- finalizacji,
- możliwości programowania mieszanego, opartego o wywoływanie funkcji rodzimych (*JNI*),
- możliwości definiowania własnych ładowaczy (*loader*).

Konsekwencje zmiany modelu bezpieczeństwa.

Zgodnie z *modelem bezpieczeństwa* przyjętym w *J2SE*, proces weryfikacji realizowany jest przez maszynę wirtualną, która sprawdza bajt kod pod kątem prób przemycenia nielegalnych komend. W warstwie bezpieczeństwa aplikacji zajmuje się tym *menadżer bezpieczeństwa*, którego realizacja wymaga zasobów przekraczających minimalne wymagania określone w *CLDC*. Dlatego zastosowano prostszy model nazwany *piaskownicą (sandbox model)*. Konsekwencją zmiany *modelu* są następujące ograniczenia:

- uniemożliwiono wykorzystanie *JNI* celem zapobiegania odwołaniom do rodzimych funkcjonolności urządzenia z pominięciem udostępnionego API
- użytkownik nie możne definować własnych metod dynamicznie ładujących klasy, gdyż mogłoby to doprowadzić do ominięcia mechnizmu ładowania klas realizowanego przez maszynę wirtualną.

2.2.2 Współczesne aplikacje *CLDC*

Wpółcześnie w procesie tworzenia aplikacji dla urządzeń *konfiguracji CLDC* podstawę stanowi wykorzystnie specyfikacji *JSR-30, JSR-139, JSR-36 i JSR-218*.

CLDC 1.1 i MIDP 2.0

Najważniejsze z różnic pomiędzy *CLDC 1.0* i *CLDC 1.1* to:

- dodano typ zmiennych rzeczywistych
  - dodano klasy `Double` i `Float`
  - zmodyfikowano klasy zależne tak aby wykorzystywały zmienne rzeczywiste
- dodano obsługę *weak references*
- przeprojektowano i dodano klasy `Calendar, Date, TimeZone`
- dodano klasę `NoClassDefFoundError`
- zwiększono ilość pamieci do 192KB

Również *profil MIDP* opisany przez *JSR-218* doczekał się następujących ulepszeń:

- wzbogacony interfejs użytkownika,

- obsługa multimediów (patrz rozdział 6),
- obsługa gier (patrz rozdział 3),
- zwiekszono ilość klas odpowiedzialnych za połączenia sieciowe i bezpieczeństwo (patrz rozdział 8),
- lepiej rozwinięty system dostarczania i instalowania aplikacji.

Zmiany w interfejsie użytkownika

Interfejs użytkownika został wzbogacony o możliwość tworzenia rozwijanych menu poprzez dodanie do klasy `Choice` opcji `POPUP`, której można użyć jako parametru podczas tworzenia obiektu klasy `ChoiceGroup`. Uzyskany efekt został przedstawiony na rysunku 2.5.

Menu zwinięte Menu rozwinięte

Rysunek 2.5: *Wizualizacje rozwijanego menu.*

```
package tomPack;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public
class HelloMidlet
    extends MIDlet{

    private Display display;

    private Form form;
    private ChoiceGroup choiceGroup;

    public HelloMidlet(){

        String tab[] = new String[20];
        for( int i=0; i< tab.length; i++){
            tab[i] = "Opcja " + i;
        }

        form = new Form("Form");
        choiceGroup = new ChoiceGroup(
            "Rozwijane menu:",
            Choice.POPUP,
            tab,
            null
        );
    }

    public void startApp() {

        display = Display.getDisplay(this);

        form.append(choiceGroup);

        display.setCurrent(
```

```
38                form
39            );
40        }
41
42        public void pauseApp(){
43        }
44
45        public void destroyApp(boolean unconditional) {
46        }
47    }
```

Kolejnym dodanym komponentem jest klasa `CustomItem`. Pozwala ona na tworzenie własnych komponentów reagujących na zdarzenia.

Rysunek 2.6: *Wizualizacja komponentu typu `CustomItem`.*

Rysunek 2.6 przedstawia komponent, który reaguje na wciskanie klawiszy przesuwając strzałkę.

```
1   package tomPack;
2
3   import javax.microedition.midlet.*;
4   import javax.microedition.lcdui.*;
5
6   public
7   class HelloMidlet
8       extends MIDlet{
9
10      private Display display;
11
12      private Form form;
13      private MyCustomItem mci;
14
15      public HelloMidlet(){
16
17          String tab[] = new String[20];
18          for( int i=0; i< tab.length; i++){
19              tab[i] = "Opcja " + i;
20          }
21
22          form = new Form("Form");
23
24          mci = new MyCustomItem("Licznik");
25      }
26
27      public void startApp() {
28
29          display = Display.getDisplay(this);
30
31          form.append(mci);
32
33          display.setCurrent(
34              form
```

```
35          );
36      }
37
38      public void pauseApp(){
39      }
40
41      public void destroyApp(boolean unconditional) {
42      }
43  }
44
45  class MyCustomItem
46      extends CustomItem{
47
48      private int stringWidth,
49                  stringHeight,
50                  count;
51      private double step = Math.PI/16;
52
53      public MyCustomItem(String label) {
54          super(label);
55
56          count = 0;
57
58          Font f = Font.getDefaultFont();
59          stringWidth = f.stringWidth(label);
60          stringHeight = f.getHeight();
61
62          setLayout(
63              Item.LAYOUT_CENTER
64          );
65
66      }
67
68      protected int getMinContentHeight() {
69          return stringHeight+10;
70      }
71
72      protected int getMinContentWidth() {
73          return stringWidth+10;
74      }
75
76      protected int getPrefContentHeight(int width) {
77          return this.getMinContentHeight();
78      }
79
80      protected int getPrefContentWidth(int height) {
81          return this.getMinContentWidth();
82      }
83
84      protected void paint(Graphics g, int w, int h) {
85          g.drawString(""+count, w/2, h/2, Graphics.BASELINE|Graphics.HCENTER);
86          g.setColor( 255, 0, 0);
87          g.drawLine(
88              w/2,
89              h/2,
90              w/2+(int)(Math.sin((count%32)*step)*20),
91              h/2+(int)(Math.cos((count%32)*step)*20)
92          );
93          g.setColor( 0, 0, 255);
94          g.drawRect( 0, 0, w-1, h-1);
95      }
96
97      protected void keyPressed(int keyCode){
98          count++;
99          repaint();
100     }
101 }
```

MIDP 3.0

W grudniu 2009 roku zakończono prace nad specyfikacją *JSR-271*, czyli *MIDP 3.0*. Najnowsza wersja będzie poszerzeniem poprzedniej o takie elementy jak:

- możliwość automatycznego uruchomienia *MID*letów,

- pozostawienia uruchomionego *MID*letu w tle,
- wewnętrzena komunikacja pomiędzy *MID*letami.

2.3 Conected Device Configuration

CDC zostało określone przez specyfikację *JSR-36*, zakłada ona:

- 32-bitowy procesor,
- przynajmniej $2MB$ pamięci, dostępnej dla Javy (włączając w to RAM i ROM),
- funkcjonalną *maszyne wirtualną Java 2*,
- urządzenie dysponuje połączeniem do sieci.

Wykorzystanie *maszyny wirtualnej* określonej specyfikacją *Java 2*, implikuje eliminację znacznej części ograniczeń wymienionych w rozdziale 2.2.1. Jednak wymagane zasoby nadal nie pozwalają na załadowanie kompletu klas z *J2SE*, dlatego wybrano następujący podzbiór pakietów [12]:

- `java.io`,
- `java.lang`,
- `java.lang.ref`,
- `java.lang.reflect`,
- `java.lang.math`,
- `java.net`,
- `java.security`,
- `java.security.cert`,
- `java.text`,
- `java.util`,
- `java.util.jar`,
- `java.util.zip`.

W tym miejscu należy odnotować, iż *CDC* jest nadzbiorem *CLDC* w konsekwencji czego część pakietów jest wspólna. Połączone pakiety `javax.microedition.io`, `java.io` i `java.net` pozwalają urządzeniom na odczyt i zapis plików oraz wysyłanie i odbieranie danych w oparciu o *Generic Connection Framework (GCF)*.

Specyfikacja *CDC* wprowadziła również obsługę kilku modeli aplikacji, które najogólniej można podzielić na *wolnostojące* i *zarządzane*.

Model aplikacji wolnostojącej jest powszechnie stosowany w programowaniu *J2SE* i wykorzystuje metodę `main` jako początkowy punkt odniesienia. Natomiast wśród modeli zarządzanych można wyróżnić:

- aplety,
- *Xlet*'y 2.3.2,
- aplikacje kompatybilne z *CLDC*.

2.3.1 Foundation Profile

Na bazie konfiguracji *CDC* utworzono profil nazwany *Foundation*. Dodaje on część klas i metod z *J2ME* do bibliotek *CDC*, których celem jest wspieranie mechanizmów gniazd *(scoket)*, internacjonalizacji i lokalizacji. Ponadto w profilu dodano pakiety:

- `java.security.acl`,
- `java.security.interfaces`,
- `ava.security.spec`.

W literaturze [6, 2] można znaleźć dwa podstawowe zadania stawiane przed tym profilem:

- funkcja organizacyjna *(verticali specification profile)*,
- obsługa urządzeń pozbawionych własnego interfejsu użytkownika, ale obsługujących sieć.

2.3.2 Personal Basic Profile

Personal Basic Profile definiowany przez specyfikację *JSR-129* jest kolejnym nadzbiorem profilu *podstawowego*. Wzbogaca on *CDC* i *FP* o trzy nowe funkcje [12]:

- model aplikacji *Xlet*[1],
- budowanie interfejsu użytkownika w oparciu o lekkie komponenty,
- wywnętrzną komunikację pomiędzy *Xlet*'ami.

Model aplikacji *Xlet*

Xlet jest kolejnym typem aplikacji zarządzanej przez *ASM*[14]. Przypomina on *aplety*, jednak został opracowany jako część pakietu *Java TV API*, a w toku dalszych prac został zaadaptowany przez *Personal Basic Profile* i *Personal Profile*.

Główna klasa *Xlet*u implementuje interfejs `javax.microedition.xlet.Xlet`, który implementuje cztery metody:

- `initXlet()`,
- `startXlet()`,
- `pauseXlet()`,
- `destroyXlet()`.

Cykl życia aplikacji rozpoczyna się w momencie gdy *ASM* fabrykuje *Xlet*, aplikacja znajduje się wówczas w stanie *załadowanym*. Kolejnym krokiem zarządcy jest wywołanie metody `initXlet()`, celem przekazania obiektu `XletContext` do późniejszego użycia. Wraz z zakończeniem procesu inicjalizacji *Xlet* przechodzi w stan *zatrzymania*.

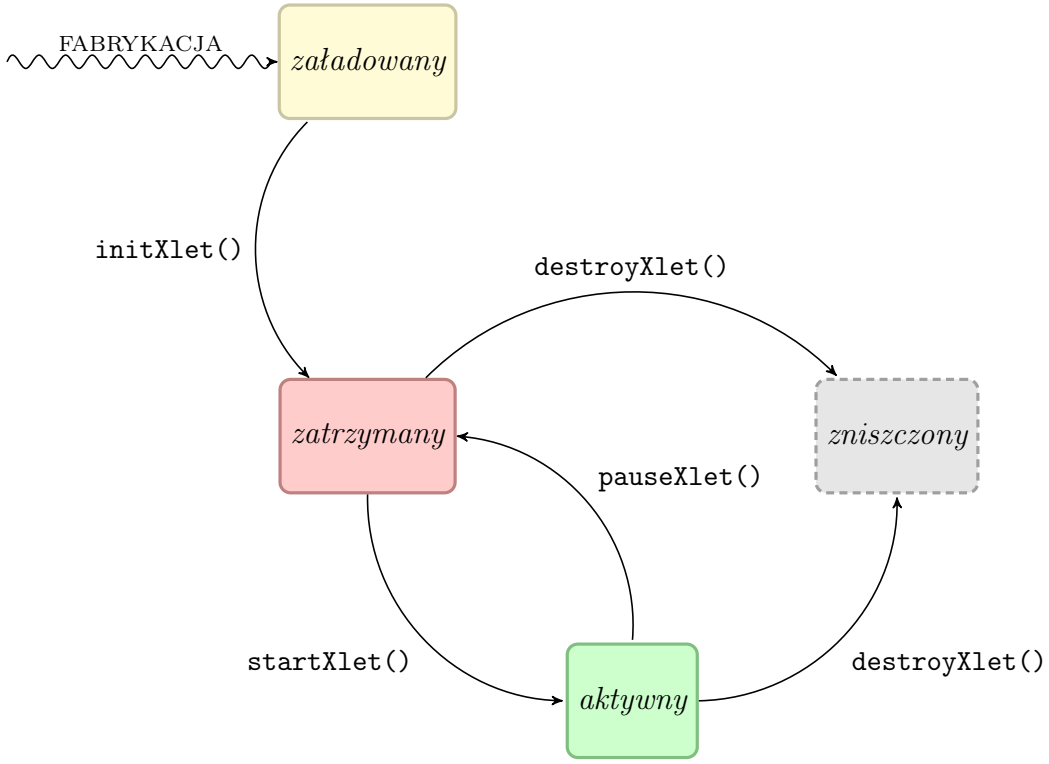
[1] Xlet zaadoptowany z *Java TV API*.

W odpowiedzi na kroki operatora, zarządca wywołuje metodę `startXlet()`, która aktywuje interfejs użytkownika, pozyskuje niezbędne zasoby systemowe i przechodzi do stanu *aktywnego*.

Kolejnym etapem w życiu aplikacji jest przejście ze stanu *atywnego* do stanu *zatrzymania*. Operacja ta może być wykonana przez aplikację *Xlet*'ową lub zarządcę. Gdy *AMS* wymusza zamianę stanu wywołuje metodę `pauseXlet()`, która zwalnia część zasobów systemowych. Natomiast gdy *Xlet* sam przechodzi w stan *zatrzymania*, metoda `pauseXlet()` nie jest wywoływana.

Ten sam mechanizm działa w przypadku zmany stanu na *zniszczony*. Metoda `destroyXlet` zostanie wywołana tylko gdy zarządca żąda takiego przejścia.

Opis ten został zilustrowany rysunkiem 2.7, natomiast przykładową implementację klasy *Xlet*owej można znaleść w listingu 2.1.



Rysunek 2.7: *Cykl życia aplikacji Xletowych.*

Warto zauważyć, że pozyskany w metodzie `initXlet()` obiekt `XletContext` pozwala na interakcję aplikacji z jej środowiskiem uruchomieniowym, służą temu następujące metody:

- `getContainer` - zwraca kontener, w którym powinny być umieszczane komponenty *Xlet*'u,
- `getXletProperty` - zwraca wymienioną własność środowiska.

Środowisko uruchomienia *Xlet*'u można wykorzystać również do poinformowania zarządcy o zmianie stanu, służą temu metody:

- `notifyActive`,
- `notifyPaused`,
- `notifyDestroyed`.

```java
import javax.microedition.xlet.*;

public
    class MainXlet
    implements Xlet{

    private XletContext context;

    public MainXlet(){
    }

    public void initXlet( XletContext context )
        throws XletStateChangeException {

        this.context = context;
    }

    public void destroyXlet( boolean unconditional )
        throws XletStateChangeException {
    }

    public void pauseXlet(){
    }

    public void startXlet()
        throws XletStateChangeException {
    }
}
```

Listing 2.1: *Podstawowy kod aplikacji Xletowej*

Interfejs użytkownika

Jak już zaznaczono, aplikacje w profilu Personal Basic pozwalają na wykorzystanie komponentów *AWT*. Można więc tworzyć aplikacje w oparciu np. o klasę `Component`, która pozwala na przedefiniowanie metody `paint` - tak jak pokazano to w listingu 2.2

```java
package tomPack;

import java.awt.*;
import javax.microedition.xlet.*;

public
    class Main
    extends Component
    implements Xlet{

    private XletContext cont;
    private Container c;

    public void destroyXlet(boolean arg0) throws XletStateChangeException {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public void initXlet(XletContext arg)
        throws XletStateChangeException {
        cont = arg;
        try {
            c = cont.getContainer();
            c.add(this);
        } catch (UnavailableContainerException ex) {
            ex.printStackTrace();
        }
    }

    public void pauseXlet(){
        System.out.println("");
    }
```

```
32
33      public void startXlet() throws XletStateChangeException {
34          c.setVisible(true);
35      }
36
37      public void paint(Graphics g){
38          int width = c.getWidth()/2,
39              height = c.getHeight()/2;
40
41          g.drawString( "Hello Xlet", width, height);
42      }
43  }
```

Listing 2.2: *Aplikacja Xlet'owa*

2.3.3 Zaawansowana grafika i interfesj użytkownika

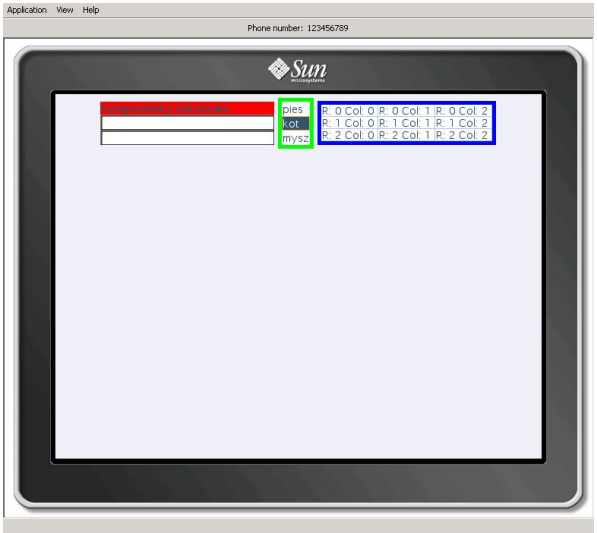
Zadaniem pakietu *Advanced Graphics and User Interface* (*AGUI*) było przeniesienie na platoformę mobilną *J2ME* technik opracowanych na potrzeby *J2SE*. Migracji podlegały kluczowe elementy pakietów *J2SE 1.4.1*, do których zaliczono:

- Swing,
- Java 2D Graphics and Imaging,
- Image I/O,
- Input Method Framework.

Pakiet ten został oznaczony jako *JSR-209* i jest rozszerzeniem profili *Personal Basic* oraz *Personal*. Pomimo że implementacja tego pakietu jest przeniesieniem z *J2SE*, to część mechanizmów została zmodyfikowana:

- wprowadzono ograniczenie w konstruktorach klas `BufferedImage`, `RenderedImage`, `Raster` i `WritableRaster`,
- wprowadzono ograniczenia w komponentach `JFrame`, `JOprionPane` i `JPopupMenu`,
- komponent `JList` nie gwarantuje obsługi zadanego mechanizmu generowania komórek,
- zaleca się aby komponenty ciężke i lekkie nie były mieszane,
- wprowadzono ograniczenia do mechnizmu funkcji `paint`,
- klasy w pakiecie `javax.swing` nie zawierają wyszeregowanych form,
- ograniczono możliwości zmiany marginesów w komponentach rodzimych.

Poglądową implementację komponentów *Swing* wchodzących w skład *AGUI* przedstawiono w listingu 2.3 i rysunku 2.8.



Rysunek 2.8: *Wizualizacja komponentów Swing w AGUI*

```
package tomPack;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;
import javax.swing.JList.*;
import javax.swing.table.*;

public
    class Main
    extends JFrame {

    public static void main(String args[]) {
        EventQueue.invokeLater(
            new Runnable() {
                public void run() {
                    new Main().setVisible(true);
                }
            }
        );
    }

    public Main() {
        super("Okno Swing w AGUI");

        this.setBackground(Color.yellow);

        FlowLayout fl = new FlowLayout();
        this.getContentPane().setLayout(fl);

        JPanel jp1 = new JPanel();
        jp1.setBackground(Color.red);

        jp1.setLayout(
            new BoxLayout( jp1, BoxLayout.Y_AXIS)
        );

        JLabel label = new JLabel("Komponenty tekstowe:");
        jp1.add(label);

        JPasswordField pass = new JPasswordField(10);
        jp1.add(pass);

        JTextField text = new JTextField(10);
        jp1.add(text);

        this.getContentPane().add(jp1);


```

```
51
52          JPanel jp2 = new JPanel();
53          jp2.setBackground(Color.green);
54
55          String data[] = {
56              "pies",
57              "kot",
58              "mysz"
59          };
60          JList list = new JList(data);
61          jp2.add(list);
62          this.getContentPane().add(jp2);
63
64          JPanel jp3 = new JPanel();
65          jp3.setBackground(Color.blue);
66
67          JTable table = new JTable(
68              new AbstractTableModel(){
69
70                  StringBuffer sb = new StringBuffer();
71
72                  public int getColumnCount(){
73                      return 3;
74                  }
75                  public int getRowCount(){
76                      return 3;
77                  }
78                  public Object getValueAt( int row, int col){
79                      sb.delete( 0, sb.length());
80                      sb.append("R: ");
81                      sb.append(row);
82                      sb.append(" Col: ");
83                      sb.append(col);
84                      return sb.toString();
85                  }
86              }
87          );
88          jp3.add(table);
89
90          this.getContentPane().add(jp3);
91
92          this.setVisible(true);
93      }
94
95  }
```

Listing 2.3: *Komponenty Swing w AGUI*

2.3.4 KVM i CVM

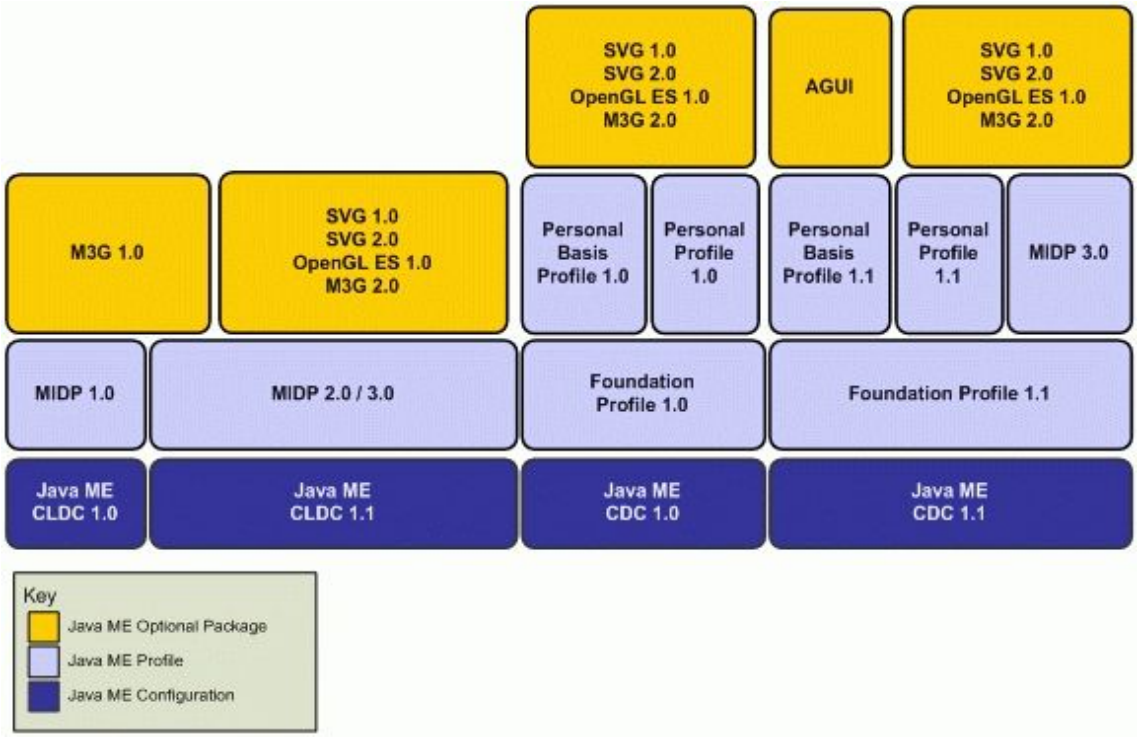
Maszyny wirtualne KVM i *CVM* są ogrniczane przez zasoby urządzeń na których są uruchamiane. Warto jednak zwrócić uwagę na pewien aspekt nazewniczy obu realizacji.

KVM rozwija się jako *Kilo Virtual Machine*, jak więc widać sama nazwa pokazuje urządzenia będące w stanie ją uruchomić.

W przypadku *CVM* samo rozszyfrowanie skrótu jest kłopotliwe, gdyż realizuje tę samą specyfikacje co *JVM*. Jak podaje 'Wireless J2ME Platform Programming' [31], skrót należy rozszyfrować jako *Compact Virtual Machine*. Natomiast sama *maszyna wirtualna* różni sie od *JVM* organizacją która została zoptymalizowana pod kątem jak najlepszego wykorzystania ograniczonych zasobów urządzeń przenośnych.

3 Grafika 2D

Przedstawione w rozdziałach 1 i 2 programy pozwalają na komunikację z użytkownikiem, nie umożliwią jednak stworzenia np. interaktywnej gry ponieważ brakuje w nich grafiki. Rysunek 3.1 przedstawia pakiety wykorzystywane w tworzeniu aplikacji graficznych.



Rysunek 3.1:

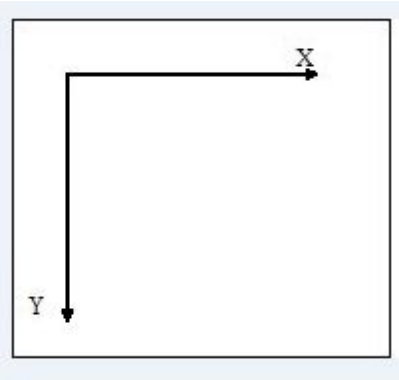
3.1 Grafika w *CLDC 1.0* i *MIDP 1.0*

W *Midlet*ach wersji *1.0* dla potrzeb grafiki przygotowano abstrakcyjną klasę `Canvas`, która definiuje metodę znaną z apletów i aplikacji:

```
protected abstract void paint(Graphics g)
```

Metoda `paint` wywoływana jest, gdy zaistniała potrzeba odświeżenia lub odrysowania fragmentu ekranu. Jako parametr dostarczany jest wykreślacz (obiekt klasy `Graphics`), który pozwala na wyrysowanie kształtów i tekstu w określonym miejscu.

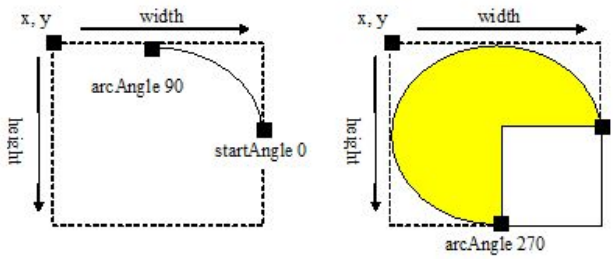
Rysowanie przy użyciu wykreślacza sprowadza się do wywoływania metod z określonymi parametrami np. `drawLine` czy `fillRect`. Układ współrzędnych jest charakterystyczny dla Javy i przedstawia go rysunek 3.2.



Rysunek 3.2: *Osie układu współrzędnych w Javie.*

Podstawowymi metodami klasy są:

- `drawLine(int x1, int y1, int x2, int y2)` — rysuje linię łączącą punkty dostarczone jako parametry,
- `drawRect(int x, int y, int width, int height)` lub `fillRect(...)` – pozwala na wyrysowanie lub wypełnienie prostokąta którego lewy górny róg będzie znajdował się w punkcie (x, y) a boki będą miały szerokość i wysokość parametrów width i height,
- `drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)` lub `fillArc(...)` – rysuje krzywą jak na rysunku 3.3,
- `setColor(int red, int green, int blue)` -– pozwala na określenie składowych RGB koloru jakim będą wykreślane kształty.



Rysunek 3.3: *Mechanika rysowania i wypełniania łuków.*

Wymienione metody pozwalają na stworzenie prostego komponentu, który będzie wyrysowywał grafikę. Przedstawiono go w listingu 3.1

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5
6  public
7  class GraphCanva
8      extends Canvas{
9
10     public void paint(Graphics g){
```

```
11
12          g.setColor( 255, 0, 0);
13          g.fillArc(
14              getWidth()/2 - 20, getHeight()/2 - 20,
15              40, 40,
16              45, 90
17          );
18          g.setColor( 0, 255, 0);
19          g.fillArc(
20              getWidth()/2 - 20, getHeight()/2 - 20,
21              40, 40,
22              135, 90
23          );
24          g.setColor( 0, 0, 255);
25          g.fillArc(
26              getWidth()/2 - 20, getHeight()/2 - 20,
27              40, 40,
28              225, 90
29          );
30          g.setColor( 0, 255, 255);
31          g.drawArc(
32              getWidth()/2 - 20, getHeight()/2 - 20,
33              40, 40,
34              315, 90
35          );
36
37          g.setColor( 0, 0, 0);
38          g.drawLine(
39              0, 0,
40              getWidth(), getHeight()
41          );
42          g.drawLine(
43              getWidth(), 0,
44              0, getHeight()
45          );
46          g.drawRect(
47              0, 0,
48              getWidth() - 1, getHeight() - 1
49          );
50      }
51  }
```

Listing 3.1: *Prosta klasa* `Canvas`

Kolejnym przykładem wykorzystania grafiki może być program wyrysowujący kwiatek przedstawiony na rysunku 3.4.

Rysunek 3.4: *Efekt wykonania programu kwiatek.*

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5
6  public
7  class HelloMidlet
```

```
    extends MIDlet{

    private Display display;

    public HelloMidlet(){
        display = Display.getDisplay(this);
    }

    public void startApp() {
        display.setCurrent(
            new GraphCanva()
        );
    }

    public void pauseApp(){
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

```
package tomPack;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public
class GraphCanva
    extends Canvas{

    int width,
        height,
        r = 30;

    public GraphCanva(){
        width = getWidth();
        height = getHeight();
    }

    int[][] kolor = {
        { 255, 0, 0},
        { 0, 0, 255},
    };

    public void paint(Graphics g){

        g.setColor( 0, 255, 0);
        int szerokoscLodygi = 4;
        for(int i=0; i< szerokoscLodygi; i++){
            g.drawLine(
                (width/2)-(szerokoscLodygi/2) + i, (height/2),
                (width/2)-(szerokoscLodygi/2) + i, height
            );
        }

        for(int i=0; i < 360; i+=25){
            int wrt = i % kolor.length;
            g.setColor(
                kolor[wrt][0], kolor[wrt][1], kolor[wrt][2]
            );
            g.fillArc(
                (width/2) - r, (height/2) - r,
                2*r, 2*r,
                i, 15
            );
        }
        g.setColor( 255, 255, 0);
        g.fillArc(
                (width/2) - (r/4), (height/2) - (r/4),
                2*(r/4), 2*(r/4),
                0, 360
        );
    }
```

```
53 }
```

Listing 3.2: *Program wyrysowujący kwiatek*

3.1.1 Animacja

Wyrysowywanie statycznych obrazów bez wątpienia jest postępem, jednak dopiero animacja pozwoli na tworzenie interaktywnych gier. Efekt przesuwania się elementów można uzyskać przez oszukanie ludzkiego oka, wyświetlając około 25 klatek animacji na sekundę. Oznacza to, że ekran powinien zostać odrysowany jeden raz na ok. 40 milisekund. W dokumentacji znajdujemy metodę `repaint()` z klasy `Canvas`, która odświeża cały ekran.

Odrysowanie ekranu powinno się wykonywać w oddzielnym przepływie sterowania, tak aby nie łączyć tego procesu np. z naciśnięciem klawisza. Każdy przepływ sterowania przez instrukcje programu realizowany jest przez oddzielny wątek, który w języku *Java* można zaimplementować jako klasę implementującą interfejs `Runnable` lub dziedziczącą po klasie `Thread`. Uruchomienie wątku będzie realizowane przez wywołanie metody `start()` na rzecz obiektu mającego w swojej hierarchii dziedziczenia klasę `Thread`. Zatem tworzenie i uruchomienie wątku można przedstawić następująco:

```
public
class ExampleThread
    extends Thread{

    public void run(){
        // Wypisz 100 razy „hello world”
        for(int i=0; i<100; i++){
            System.out.println(“Hello world”);
        }
    }
}

...

new ExampleThread().start();
```

Powyższe zastosowanie jest prawidłowe, jednak niestety nie można go wykorzystać w przypadku programu opisanego listingiem 3.2, ponieważ *Java* nie przewiduje możliwości wielodziedziczenia. Dlatego też należy wykorzystać poniższy sposób:

```
public
class MyThreadClass
    implements Runnable{

    public void run(){
        // Wypisz 100 razy „hello world”
        for(int i=0; i<100; i++){
            System.out.println(“Hello world”);
        }
    }
}

...

new Thread(
    new MyThreadClass()
).start();
```

Implementacja i uruchomienie wątku są konieczne do utworzenia animacji, ale równie ważne jest zadbanie, aby w animacji istniały elementy które będą animowane. Korzystając zatem z listingu 3.2 zaimplementujmy przykładową animację, w której

płatki kwiatu będą naprzemiennie zmieniały swój kolor.

Podstawą dla takiej animacji będzie funkcja run() której celem będzie odświeżenie ekranu, a następnie zaśnięcie (nie wykonywanie żadnych operacji) na pewien okres czasu.

```
    public void run(){
        while(true){
            try{
                repaint();
                Thread.sleep(120);
            }catch(Exception ex){}
        }
    }
```

Aby uzyskac efekt zmiany koloru płatków zdefiniowana zostanie zmienna logiczna `boolean`, która przy każdym wywołaniu funkcji `paint(Graphics)` będzie zmieniać swój stan na przeciwny:

```
stan = !stan;
```

Zależnie od wartości tej zmiennej płatki będą przyjmowały kolor z tablicy `kolory` na pozycji

```
(i+1) % kolory.length
```

lub

```
i % kolory.length
```

Całość programu przedstawiona została w listingu 3.3.

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5
6  public
7  class HelloMidlet
8      extends MIDlet{
9
10     private Display display;
11
12     public HelloMidlet(){
13         display = Display.getDisplay(this);
14     }
15
16     public void startApp() {
17         display.setCurrent(
18             new GraphCanva()
19         );
20     }
21
22     public void pauseApp(){
23     }
24
25     public void destroyApp(boolean unconditional) {
26     }
27 }
```

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5
```

```
6  public
7  class GraphCanva
8      extends Canvas
9      implements Runnable{
10
11     int width,
12         height,
13         r = 30;
14
15     public GraphCanva(){
16         width = getWidth();
17         height = getHeight();
18         new Thread(this).start();
19     }
20
21     int[][] kolor = {
22         { 255, 0, 0},
23         { 0, 0, 255},
24     };
25     private boolean stan = true;
26
27     public void paint(Graphics g){
28
29         g.setColor( 0, 255, 0);
30         int szerokoscLodygi = 4;
31         for(int i=0; i< szerokoscLodygi; i++){
32             g.drawLine(
33                 (width/2)-(szerokoscLodygi/2) + i, (height/2),
34                 (width/2)-(szerokoscLodygi/2) + i, height
35             );
36         }
37
38         stan = !stan;
39
40         for(int i=0; i < 360; i+=25){
41             int wrt;
42             if(stan){
43                 wrt = (i+1) % kolor.length;
44             }else{
45                 wrt = i % kolor.length;
46             }
47             g.setColor(
48                 kolor[wrt][0], kolor[wrt][1], kolor[wrt][2]
49             );
50             g.fillArc(
51                 (width/2) - r, (height/2) - r,
52                 2*r, 2*r,
53                 i, 15
54             );
55         }
56         g.setColor( 255, 255, 0);
57         g.fillArc(
58                 (width/2) - (r/4), (height/2) - (r/4),
59                 2*(r/4), 2*(r/4),
60                 0, 360
61         );
62     }
63
64     public void run(){
65         while(true){
66             try{
67                 repaint();
68                 Thread.sleep(120);
69             }catch(Exception ex){}
70         }
71     }
72 }
```

Listing 3.3: *Animacja płatków kwiatka.*

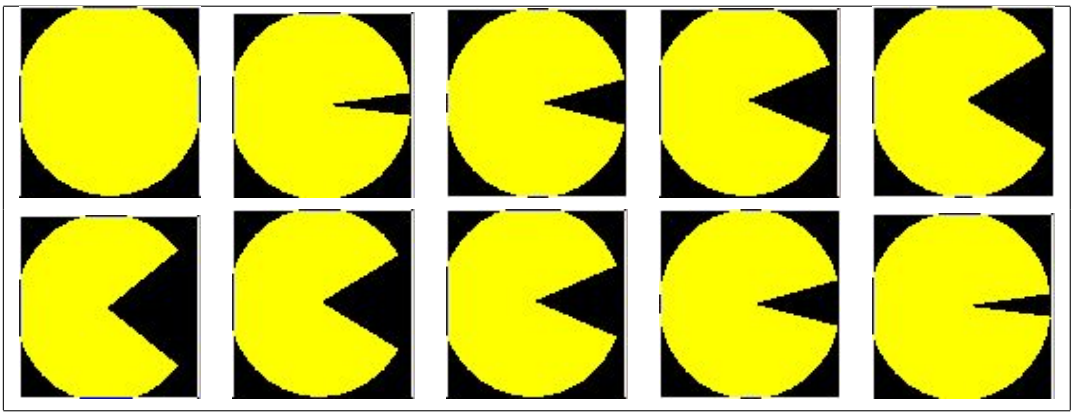
3.1.2 Podwójne buforowanie

Program opisany listingiem 3.3 cechuje kilka poważnych wad:

- odświeżany jest cały ekran, a nie wyłącznie fragment który uległ zmianie
- ciągłe wykreślanie (nawet tych samych kształtów) jest kosztowne obliczeniowo
- pojawia się niezwykle nieprzyjemny dla oka efekt migotania.

W rozwiązaniu wymienionych problemów pomocna będzie klasa `Image`. Stosując technikę podwójnego buforowania można wyeliminować lub bardzo ograniczyć efekt migotania. Idea ta zakłada iż całość operacji graficznych jest wykonywana na obrazie (obiekcie klasy `Image`) przechowywanym w pamięci a dopiero gdy wszystkie operacje zostaną wykonane, wówczas przygotowany obraz zostaje wyrysowany na ekranie.

Rozwiązanie problemu kosztów zakłada, że w scenie istnieje jeden lub kilka charakterystycznych obrazów występujących w określonej kolejności. Przykładem może być *PacMan* widoczny na rysunku 3.5.



Rysunek 3.5: *Klatki animacji.*

Utworzenie obrazu odbywa się za pomocą statycznej metody `createImage` z klasy `Image`, wymaga ona podania wysokości i szerokości tworzonego obrazu. Gdy obraz zostanie utworzony można pobrać jego wykreślacz za pomocą metody `getGraphics()` a następnie wyrysować żądane elementy. W przykładzie opisanym listingiem 3.4 obrazy generowane są przez metodę `drawPacMan(int)`.

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5
6  public
7  class GraphCanva
8      extends Canvas
9      implements Runnable{
10
11     int width,
12         height,
13         r = 30;
14
15     Image tab[];
16
17     public GraphCanva(){
18         width = getWidth();
19         height = getHeight();
20
21         tab = new Image[10];
22
23         for(int i=0; i < tab.length; i++){
24             if(i < tab.length/2)
25                 tab[i] = drawPacMan( 15 * i);
26             else
```

```
27                  tab[i] = drawPacMan( 15 * (tab.length - i));
28          }
29
30          new Thread(this).start();
31      }
32
33      int[][] kolor = {
34          { 255, 0, 0},
35          { 0, 0, 255},
36      };
37
38      public Image drawPacMan(int otwarcie){
39          Image img = Image.createImage( width, height);
40          Graphics g = img.getGraphics();
41          g.setColor( 0, 0, 0);
42          g.fillRect(0,0, width, height);
43          g.setColor( 255, 255, 0);
44          g.fillArc(
45              0, 0,
46              width, height,
47              otwarcie/2, 360 - otwarcie
48          );
49          return img;
50      }
51      int wrt = 0;
52
53      public void paint(Graphics g){
54          g.drawImage(
55              tab[wrt%tab.length],
56              0, 0, Graphics.TOP | Graphics.LEFT
57          );
58
59          if(wrt < tab.length -1){
60              wrt++;
61          }else{
62              wrt = 0;
63          }
64      }
65
66      public void run(){
67          while(true){
68              try{
69                  repaint();
70                  Thread.sleep(120);
71              }catch(Exception ex){}
72          }
73      }
74  }
```

Listing 3.4: *Animacja PacMan.*

3.1.3 Ładowanie obrazów

Dynamiczne tworzenie obrazów jest na pewno wygodnym rozwiązaniem, programista może bowiem dostosować wyrysowywane elementy do rozmiarów ekranu jakimi dysponuje urządzenie. Procedura może jednak okazać się dość czasochłonna, co w przypadku gry na pewno wpłynie na jej atrakcyjność. Dlatego też istnieje możliwość wczytania obrazu z tablicy bajtów lub pliku. Listing programu 3.5 przedstawia klasę realizującą zadanie załadowania obrazów.

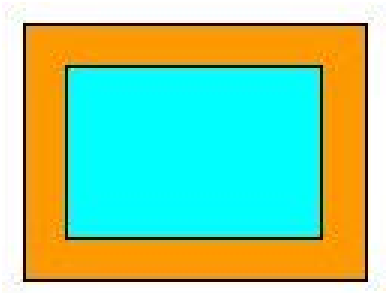
```
1   package tomPack;
2
3   import javax.microedition.midlet.*;
4   import javax.microedition.lcdui.*;
5
6   public
7   class GraphCanva
8       extends Canvas
9       implements Runnable{
10
```

```
11      private int width, height;
12      private Image[] tab;
13
14      public GraphCanva(){
15          width = getWidth();
16          height = getHeight();
17          tab = new Image[5];
18          try{
19              for(int i = 1; i < 6;i++)
20                  tab[i-1] = Image.createImage("/img"+i+".png");
21          }catch(Exception ex){
22          }
23          new Thread(this).start();
24      }
25      int num = 0;
26      public void paint(Graphics g){
27          g.setColor( 255, 255, 255);
28          g.fillRect( 0, 0, tab[0].getWidth(), tab[0].getHeight());
29          g.drawImage( tab[num%tab.length], 0, 0, Graphics.TOP | Graphics.LEFT);
30      }
31
32      public void run(){
33          try{
34              while(true){
35                  num++;
36                  repaint();
37                  Thread.sleep(50);
38              }
39          }catch(Exception ex){
40              System.out.println(ex);
41          }
42      }
43  }
```

Listing 3.5: *Ładowanie obrazów.*

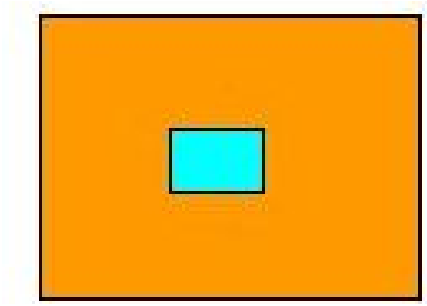
Programy *MIDP 1.0* mogą czytać jedynie pliki z rozszerzeniem `PNG`, co zostało zaznaczone w dokumentacji. Załadowane obrazy mają rozmiary zgodne z plikami dołączonymi do *MIDlet*u, więc ich stosowanie jest obarczone poważną niedogodnością gdyż programy wykorzystujące tę technikę muszą być uruchamiane na urządzeniach o wymiarach wyświetlacza zbliżonych do rozmiaru ekranu urządzenia dla którego pierwotnie utworzono aplikację.

Rozwiązanie 1 Doraźnym rozwiązaniem tego problemu jest rozmieszczenie elementów z zachowaniem dynamicznych marginesów, tak jak to widać na rysunku 3.6.



Rysunek 3.6: *Wizualizacja rozwiązania 1*

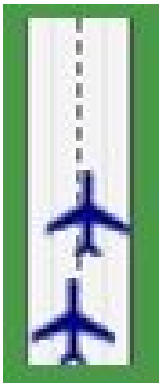
Jednak jak można łatwo spostrzec, w przypadku gdy będziemy przenosić grę z komórki na komputer naręczny doprowadzimy do sytuacji, w której znaczna część ekranu pozostanie nie wykorzystana.



Rysunek 3.7: *Negatywny efekt rozwiązania 1*

Rozwiązanie 2 Alternatywnym rozwiązaniem może być utworzenie kilku lub kilkunastu kompletów obrazów dostosowanych do najczęściej spotykanych modeli urządzeń. W takiej sytuacji znacznie zwiększamy ilość urządzeń prawidłowo uruchamiających program, jednak kosztem problemów związanych z wybraniem właściwego zbioru obrazów i ich prezentacji.

Rysunek 3.8 ilustruje ten problem, wyrysowywując dwa samoloty przesunięte względem siebie o jeden piksel. Który z nich znajduje się na środku?



Rysunek 3.8: *Wizualizacja rozwiązania 2*

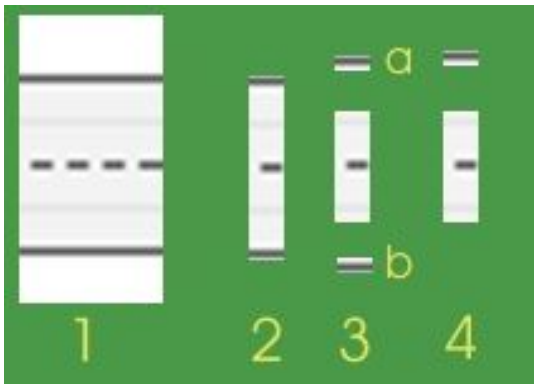
Koncepcja ta ma jeszcze jedną poważną wadę. Ilość telefonów wzrasta w zastraszającym tempie, a co za tym idzie przyrastałaby również ilość potrzebnych kompletów obrazów, a w konsekwencji i rozmiar pliku archiwum `jar`. W ten sposób prędzej czy później doprowadzimy do sytuacji, w której niewielki program będzie miał rozmiar powyżej 500 kb.

Rozwiązanie 3 Skoro niepraktyczne okazało się przechowywanie plików w archiwum, to postarajmy się umieścić je całkowicie poza plikiem `jar`. Skorzystajmy z właściwości urządzenia którym dysponujemy, połączmy się z siecią i w zależności od modelu telefonu ściągnijmy komplet grafiki który nam najlepiej odpowiada.

```
public Image loadImage(String url){
    HttpConnection con = (HttpConnection)Connector.open(url);
    try{
        int length = (int)hc.getLength();
        byte[] data = new byte[length];
        DataInputStream dis = new DataInputStream(hc.openInputStream());
        in.readFully(data);
        return Image.createImage( data, 0, data.lenght);
    }catch(Exception ex){
        con.close();
    }
}
```

Powyższa metoda pozwoli nam dynamicznie pobrać potrzebne obrazy, jednak również to rozwiązanie jest obarczone wadą -– konieczne moze być poniesienie kosztów transmisji danych.

Rozwiązanie 4 – ostateczne Ponieważ żadne z powyższych rozwiązań nie okazało się idealne, zatem stwórzmy hybrydę wykorzystującą mocne strony wymienionych koncepcji.

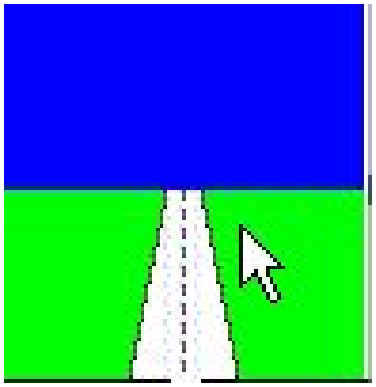


Rysunek 3.9: *Rozkład obrazu na składowe*

Przygotujmy komplet grafik potrzebnych w programie i poszukajmy w nich prawidłowości.

1. Widać, iż pas startowy składa się z kolejno następujących po sobie przerywanych linii.
2. Odrzucamy pobocza, które będą tłem naszej aplikacji
3. Krańce pasa a i b są takie same, zatem jeden można usunąć

W rezultacie dysponujemy jednym krańcem i środkową częścią pasa. Ten sam proces powtarzamy w odwrotnej kolejności, podczas uruchomienia programu. Pozwala to na uzyskanie efektu takiego jak na rysunku 3.10. Mechanizm został zaimplementowany w listingu 3.6.



Rysunek 3.10: *Efekt składania obrazu*

```
1 package tomPack;
2
3 import javax.microedition.midlet.*;
```

```java
import javax.microedition.lcdui.*;

public
class GraphCanva
    extends Canvas
    implements Runnable{

    private int width, height;
    private Image center, rest;

    private int roadStep = 0;

    public GraphCanva(){
        try{
            center = Image.createImage("/road.png");
            rest = Image.createImage("/rest.png");
        }catch(Exception ex){
        }

        width = getWidth();
        if( width%2 != 0)
            width -= 1;

        height = getHeight();

        makeBackground();
        makeRoad(roadStep);

        new Thread(this).start();
    }

    public void paint(Graphics g){
        g.drawImage( back, 0, 0, Graphics.TOP | Graphics.LEFT);
        g.drawImage( road, width/2, height/2, Graphics.TOP | Graphics.HCENTER);
        if(roadStep < 4)
            roadStep++;
        makeRoad(roadStep);
    }

    private Image back;

    private void makeBackground(){
        back = Image.createImage( width, height);
        Graphics g = back.getGraphics();
        g.setColor( 0, 255, 0);
        g.fillRect( 0, 0, width, height);
        g.setColor( 0, 0, 255);
        g.fillRect( 0, 0, width, height/2);
    }

    private Image road;

    private void makeRoad(int step){
        int count = (height/2) / center.getHeight();
        int w = count * step * 2;
        road = Image.createImage( center.getWidth() + w, height/2);

        Graphics g = road.getGraphics();
        g.setColor( 0, 255, 0);
        g.fillRect( 0, 0, road.getWidth(), road.getHeight());



        g.setColor( 255, 255, 255);
        for(int i = 0, j = 0; i < road.getHeight(); i+=center.getHeight(), j += step){
            g.fillRect(
                ((road.getWidth()/2)-(center.getWidth()/2))-j, i,
                center.getWidth()+(2*j), center.getHeight()
            );
            g.drawImage( center, road.getWidth()/2, i, Graphics.TOP | Graphics.HCENTER);
            g.drawImage( rest,
                ((road.getWidth()/2) - (center.getWidth()/2)) - j, i,
                Graphics.TOP | Graphics.HCENTER
            );
```

```
78                g.drawImage( rest,
79                    ((road.getWidth()/2) + (center.getWidth()/2)) + j, i,
80                    Graphics.TOP | Graphics.HCENTER
81                );
82            }
83        }
84
85        public void run(){
86            try{
87                while(true){
88                    repaint();
89                    Thread.sleep(3000);
90                }
91            }catch(Exception ex){
92                System.out.println(ex);
93            }
94        }
95    }
```

Listing 3.6: *Dynamiczne składanie obrazów*

3.2 MIDP 2.0

MIDP 2.0 wprowadziło zmiany zarówno w komponentach inserfejsu użytkownika, jak i w komponentach graficznych. Wprowadzono trzy nowe pakiety dedykowane obsłudze gier i multimediom:

- `javax.microedition.lcdui.game`,
- `javax.microedition.lcdui.media`,
- `javax.microedition.lcdui.media.control`.

Pakiet poświęcony grom wprowadził istotne ułatwienia, pierwszym jest dziedzicząca po `Canvas` klasa `GameCanvas`. Drugim wprowadzenie mechanizmu zarządzania i wyrysowywania wielu warstw, wykorzystanego w klasach:

- `Layer`,
- `LayerManager`,
- `TiledLayer`,
- `Sprite`.

3.2.1 GameCanvas

Klasa `GameCanvas` rozwiązała problem podwójnego buforowania zaznaczony w rozdziale 3.1.2. Jej implementacja przechwytuje również zdarzenia klawiszowe, które są interpretowane i dostarczane programiście w postaci stanu sprawdzanego metodą `getKeyStates()`. Przykładową implementację klasy można znaleść w listingu 3.7.

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4
5  import javax.microedition.lcdui.*;
6  import javax.microedition.lcdui.game.*;
7
8  public
9  class HelloMidlet
10     extends MIDlet{
```

```
11
12      private Display display;
13      private MyGameCanvas mgc;
14
15      public HelloMidlet(){
16          mgc = new MyGameCanvas();
17      }
18
19      public void startApp() {
20          display = Display.getDisplay(this);
21          display.setCurrent(
22              mgc
23          );
24          new Thread(mgc).start();
25      }
26
27      public void pauseApp(){
28      }
29
30      public void destroyApp(boolean unconditional) {
31      }
32  }
33
34  class MyGameCanvas
35      extends GameCanvas
36      implements Runnable{
37
38      private int width, height;
39
40      public MyGameCanvas(){
41          super(true);
42
43          width = getWidth();
44          height = getHeight();
45      }
46
47      public void run(){
48          int keyStates;
49          Graphics g = getGraphics();
50
51          int wLength = width/3,
52              hLength = height /3;
53
54          int tab[][] = {
55              {GameCanvas.GAME_A_PRESSED, GameCanvas.UP_PRESSED, GameCanvas.↩
                    GAME_C_PRESSED},
56              {GameCanvas.RIGHT_PRESSED, GameCanvas.FIRE_PRESSED, GameCanvas.↩
                    LEFT_PRESSED},
57              {GameCanvas.GAME_B_PRESSED, GameCanvas.DOWN_PRESSED, GameCanvas.↩
                    GAME_D_PRESSED}
58          };
59
60          while(true){
61              keyStates = getKeyStates();
62              for(int i=0; i<3; i++){
63                  for(int j=0; j<3; j++){
64                      if((keyStates & tab[i][j]) != 0){
65                      switch(i){
66                          case 0:
67                              g.setColor( 255, i*85, j*85);
68                              break;
69                          case 1:
70                              g.setColor( i*85, 255, j*85);
71                              break;
72                          case 2:
73                              g.setColor( i*85, j*85, 255);
74                              break;
75                      }
76                      }else{
77                          g.setColor(0);
78                      }
79
80                      g.fillRect(i*wLength, j*hLength, wLength, hLength);
81                  }
82              }
83
```

```
84                flushGraphics();
85                try{
86                    Thread.sleep(10);
87                }catch (InterruptedException ie){
88                }
89            }
90        }
91    }
```

Listing 3.7: *Wykorzystanie* `GameCanvas`

3.2.2 Klasy warstw

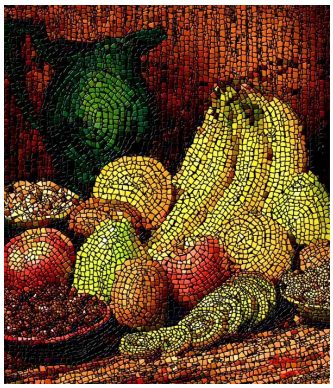
Idea warstw znana jest w grafice od bardzo dawna. Polega ona na utworzeniu szeregu obrazów które wzajemnie będą się uzupełniały np:

- warstwa tła – będzie zawierać obraz pustyni,
- warstwa 1 – będzie zawierać wizerunek rzeki,
- warstwa 0 – będzie zawierać wizerunek palmy.

Po złożeniu wszystkich warstw powstanie obraz łączący w sobie wszsytkie wymienione składowe. W pakiecie *MIDP 2.0* zarządzanie warstwami zostało powierzone obiektowi klasy `LayoutManager`, realizuje on wyrysowywanie zgodnie z kolejnością określoną przez numer warstwy, przy założeniu że warstwa 0 znajduje się najbliżej obserwatora.

Jako warstwy można dodawać obiekty dziedziczące po klasie `Layer`. W przypadku omawianego pakietu będą to obiekty klasy `TiledLayer` i `Sprite`. Obie wymagają obrazu, który będzie zawierał wyświetlane elementy (operacja ładowania obrazów została już przedstawiona w listingu 3.5).

Warstwa mozaikowa



Rysunek 3.11: *Przykładowa mozaika*

Pierwsze mozaiki tworzono już w starożytności, układając z kolorowych kamieni dekoracyjne obrazy. Również w dzisiejszych czasach technika ta znajduje zastosowanie, np. w programowaniu. Obiekt klasy `TiledLayer` jest współczesną wersją takiej realizacji. Zakłada on, że prostokątną warstwę można podzielić na pewną ilość komórek, a następnie każdą z nich wypełnić zadanym wzorem. Konstruktor wymaga zatem dostarczenia informacji o ilości komórek w pionie i poziomie, materiale użytym do wypełnienia i wielkości pojedynczej komórki.

Dysponując obrazem można utworzyć obiekt `TiledLayer` w oparciu o konstruktor:

```
    TilledLayer tl = TiledLayer( kolumny, wiersze, obraz, szerokośćKomórki, ↩
        wysokośćKomórki);
```

Rozmiar warstwy nietrudno obilczyć, mnożąc wielkość pojedynczej komórki przez ich ilość w pionie lub poziomie. Konstruktor dokonuje również indeksowania dostarczonego materiału, przy czym należy zauważyć iż numer 0 jest zarezerwowany dla pustego wypełniena, a dopiero kolejne numery przypadają wypełnieniom z dostarczonych obrazów.

W momencie utworzenia *warstwy mozaikowej* jest ona pusta. Wypełnienia komórek dokonujemy posługując się metodą:

```
    tl.setCell( kolumna, wiersz, numerWypełnienia);
```

Ostatnim etapem użycia tego komponentu będzie jego wyrysowanie poprzez wywołanie metody `paint`:

```
    tl.paint(g);
```

Powyższe operacje zostały przedstawione w programie opisanym listingiem 3.8.

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4
5  import javax.microedition.lcdui.*;
6  import javax.microedition.lcdui.game.*;
7
8  public
9      class MyGameCanvas
10     extends GameCanvas
11     implements Runnable{
12
13     private int width, height;
14     private Image img;
15     public MyGameCanvas(){
16         super(true);
17
18         width = getWidth();
19         height = getHeight();
20
21         try{
22             img = Image.createImage("/new-5.png");
23         }catch(Exception ex){
24             System.out.println(ex);
25         }
26     }
27
28     int cellX = 0,
29         cellY = 0,
30         sCellX = 0,
31         sCellY = 0;
32     boolean selected = false;
33
34
35
36     public void run(){
37         int keyStates;
38         Graphics g = getGraphics();
39
40         TiledLayer tl = new TiledLayer( 12, 8, img, 20, 25);
41         int num = 1;
42         for(int j=0; j<8; j++)
43             for(int i=0; i<12; i++)
44                 tl.setCell( i, j, num++);
```

```
45
46
47          int wLength = width/3,
48              hLength = height /3;
49
50          int tab[][] = {
51              {GameCanvas.GAME_A_PRESSED, GameCanvas.UP_PRESSED, GameCanvas.↩
                    GAME_C_PRESSED},
52              {GameCanvas.RIGHT_PRESSED, GameCanvas.FIRE_PRESSED, GameCanvas.↩
                    LEFT_PRESSED},
53              {GameCanvas.GAME_B_PRESSED, GameCanvas.DOWN_PRESSED, GameCanvas.↩
                    GAME_D_PRESSED}
54          };
55
56          while(true){
57              keyStates = getKeyStates();
58
59              if((keyStates & GameCanvas.UP_PRESSED) != 0){
60                  cellY--;
61              }
62              if((keyStates & GameCanvas.DOWN_PRESSED) != 0){
63                  cellY++;
64              }
65              if((keyStates & GameCanvas.LEFT_PRESSED) != 0){
66                  cellX--;
67              }
68              if((keyStates & GameCanvas.RIGHT_PRESSED) != 0){
69                  cellX++;
70              }
71
72              if(cellX < 0)
73                  cellX = 11;
74              else
75                  cellX %= 12;
76              if(cellY < 0)
77                  cellY = 7;
78              else
79                  cellY %= 8;
80
81              if((keyStates & GameCanvas.FIRE_PRESSED) != 0){
82                  if(selected){
83                      int tmp = tl.getCell(sCellX, sCellY);
84                      tl.setCell( sCellX, sCellY, tl.getCell(cellX, cellY));
85                      tl.setCell( cellX, cellY, tmp);
86                  }else{
87                      sCellX = cellX;
88                      sCellY = cellY;
89                  }
90                  selected = !selected;
91              }
92              tl.paint(g);
93
94              g.setColor(0xff0000);
95              g.drawRect(cellX*20, cellY*25, 20, 25);
96
97              if(selected){
98                  g.setColor(0x00ff00);
99                  g.drawRect(sCellX*20, sCellY*25, 20, 25);
100             }
101
102             flushGraphics();
103             try{
104                 Thread.sleep(100);
105             }catch (InterruptedException ie){
106             }
107         }
108     }
109 }
```

Listing 3.8: *Wykorzystanie warstwy mozaikowej*

Sprite

Klasa `Sprite` opisuje funkcjonalnie te same operacje co *warstwa mozaikowa*, jednak jej cel jest odmienny. O ile klasa `TiledLayer` zajmowała się zawartością aplikacji, o tyle `Sprite` jest poświęcony jej interaktywnym składnikom.

Rysunek 3.12:

Jak pokazuje rysunek 3.12, 12×12 pikseli jest wielkością wystarczającą, aby stworzyć animowaną postać. Animacja tego elementu jest realizowana przez wyświetlanie kolejnych klatek, których wielkość określono w konstruktorze, natomiast kolejnością wyświetlania sterują metody `nextFrame()` i `prevFrame()`.

```
1  package tomPack;
2
3  import javax.microedition.midlet.*;
4
5  import javax.microedition.lcdui.*;
6  import javax.microedition.lcdui.game.*;
7
8  public
9      class MyGameCanvas
10     extends GameCanvas
11     implements Runnable{
12
13     private Image img;
14
15     public MyGameCanvas(){
16         super(true);
17
18         try{
19             img = Image.createImage("/new-3.png");
20         }catch(Exception ex){
21             System.out.println(ex);
22         }
23     }
24
25     public void run(){
26         Graphics g = getGraphics();
27         Sprite sp = new Sprite(img, 12, 12);
28
29         while(true){
30             sp.paint(g);
31             sp.nextFrame();
32             flushGraphics();
33             try{
34                 Thread.sleep(100);
35             }catch (InterruptedException ie){
36             }
37         }
38     }
39 }
```

Listing 3.9: *Wykorzystanie* `Sprite`

3.3 SVG

Rozwiązania przedstawione w *MIDP 1.0* i *MIDP 2.0* pomagają tworzyć aplikacje graficzne. Jednak nie można powiedzieć, że problemy opisane w rozdziale 3.1.3 zostały rozwiązane gdyż ich źródłem jest rastrowy zapis obrazów. Naturalnie istnieje możliwość zapisu w formatach wektorowych, jednak do czasu wprowadzenia

JSR-226 możliwość skalowania i obracania obrazów była ograniczona. Pakiet pozwalający wczytać format wektorowy został nazwany *Scalable 2D Vector Graphics API* i pozwala na wczytywanie plików w formacie *S*calable *V*ector *G*raphics, który przy pomocy *XML*a opisuje obraz. Przykładowy *XML* obrazu pokazuje listing 3.10.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1 Tiny//EN" "http://www.w3.org/Graphics/↩
    SVG/1.1/DTD/svg11-tiny.dtd">
<svg preserveAspectRatio="xMidYMid meet" viewbox="10 10 130 55" width="150.0" ↩
    height="75.0">
    <rect x="25" y="25" transform="translate(75 36.375)
        rotate(15) translate(-70 -35.375)" fill="#00007E"
        width="100" height="22.75" stroke="#000000" stroke-width=".5"/>
    <rect x="25" y="25" fill="#FF9800" width="108.5"
        height="22.75" stroke="#000000" stroke-width=".5"/>
    <text x="30" y="40" fill="#00007E" stroke="#FFFFFF"
        stroke-width=".33" xml:space="preserve">Hello brave world.</text>
</svg>
```

Listing 3.10: *XML obrazu w formacie SVG Tiny [5]*

Jako zalety zastosowania *SVG* wymienia się [7]:

- schemat kodowania dokumentu – *XML*,
- niewielki koszt przetworzenia pliku tekstowego w porównaniu do grafiki rastrowej typu `gif`,
- skalowalność grafiki wektorowej,
- animacja wektorowa,
- obsługa zdarzeń,
- wyraźne oddzielenie interfejsu użytkownika od logiki aplikacji.

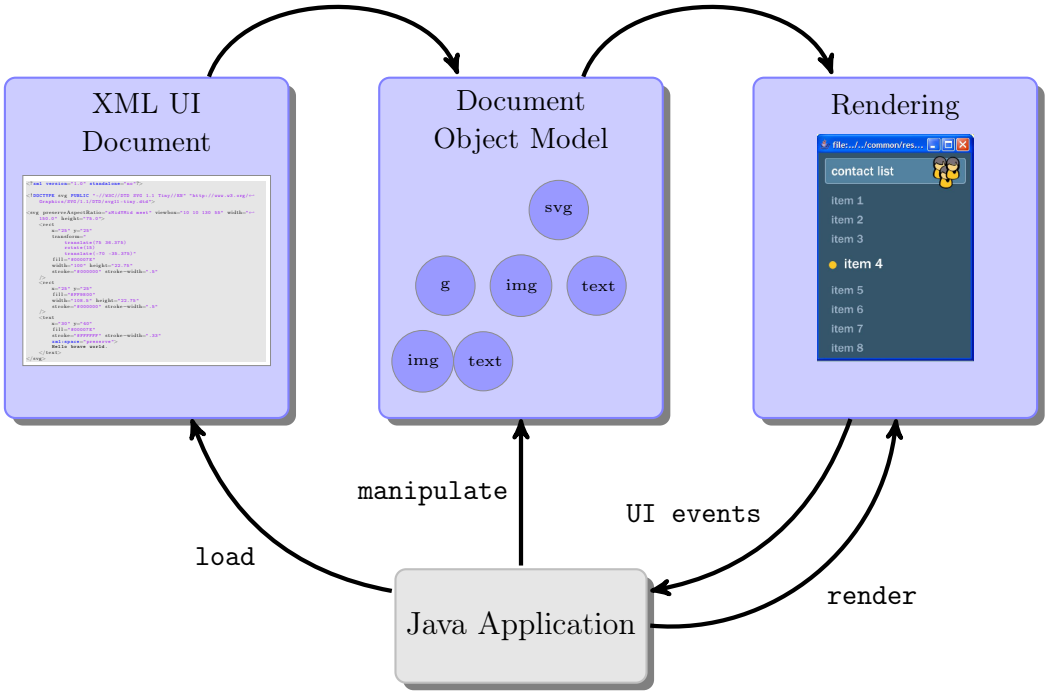
Najprostszą techniką wyświetlenia obrazu *SVG* będzie wykorzystanie klasy `SVGAnimator`, która oczekuje dostarczeniu obiektu klasy `SVGImage`. Problem polega na tym że `SVGImage` nie implementuje metod zdolnych do wczytania obrazu, robi to natomiast jej klasa bazowa `ScalableImage` przy użyciu metody `createImage`. Jak widać w listingu 3.11, konieczna była konwersja obiektu `ScalableImage` na `SVGImage`.

```
1  package tomPack;
2
3  import java.io.InputStream;
4
5  import javax.microedition.midlet.*;
6  import javax.microedition.lcdui.*;
7  import javax.microedition.m2g.*;
8  import javax.microedition.io.*;
9
10 public
11 class HelloMidlet
12     extends MIDlet{
13
14     private Display display;
15     private Canvas c;
16
17     public HelloMidlet(){
18
19         try{
20             InputStream is = getClass().getResourceAsStream("/thumbsUp.svg");
21             SVGImage svgi = (SVGImage)ScalableImage.createImage( is, null);
22
```

```
23                SVGAnimator svga =  SVGAnimator.createAnimator(svgi);
24                c = (Canvas)svga.getTargetComponent();
25                svgi.setViewportHeight(c.getHeight());
26                svgi.setViewportWidth(c.getWidth());
27            }catch(Exception ex){
28                ex.printStackTrace();
29            }
30
31        }
32
33        public void startApp(){
34            display = Display.getDisplay(this);
35            display.setCurrent(c);
36        }
37        public void pauseApp(){
38        }
39
40        public void destroyApp(boolean unconditional) {
41        }
42    }
```

Listing 3.11: *Ładowanie obrazu wektorowego*

Wyświetlenie obrazu to najprostsza operacja jaką można wykonać. Aby przejść do operacji obracania, czy skalowania należy zagłębić się w dokument *XML* i dokonać w nim stosownych zmian. Zanim to jednak nastąpi, warto zwrócić uwagę na rysunek 3.13 gdzie przedstawiono model aplikacji wykorzystującej pakiet *JSR-226* [3].



Rysunek 3.13: *Model aplikacji JSR-226*

Jak widać, model aplikacji składa się z czterech operacji:

- ładowania dokumentu SVG,
- modyfikacji,
- wyrysowania,
- reakcji na zdarzenia z interfejsu użytkownika.

Jako że pierwszy z punktów został już omówiony i aplikacja zawiera już obraz, można zająć się zagadnieniem manipulacji na elementach dokumentu pozyskanego za pomocą metody `getDocument()` wywołanej na rzecz obiektu klasy `SVGImage`. Dysponując obiektem klasy `Document` można pobrać elementy opisane w strukturze *XML*, wskazując je za pomocą identyfikatora.

```
Document doc = svgi.getDocument();
SVGElement svge = (SVGElement)doc.getElementById("Calque_1");
```

Kolejnym krokiem powinno być pozyskanie *macierzy przekształceń*, do czego posłuży metoda `getMatrixTrait`. Ważnym elementem wymienionej metody jest jej parametr. W dokumentacji interfejsu `SVGElement` można znaleźć opis dopuszczalnych atrybutów, jednym z nich jest `transform`, który opisuje przekształcenia wykonywane na danym elemencie.

```
SVGMatrix m = svge.getMatrixTrait("transform");
m.mTranslate( fx, fy);
m.mRotate(10);
m.mTranslate( -fx, -fy);
svge.setMatrixTrait("transform", m);
```

Dysponując *macierzą* można przeprowadzać operacje matematyczne na obrazach *SVG*.

Twórcy *JSR-226* przewidzieli również możliwość dynamiczego tworzenia dokumentów. Proces ten można podzielić na dwie części:

- utworzenia dokumentu, za pomocą polecenia `SVGImage.createEmptyImage`,
- dodania nowych elementów, które sprowadza się do utworzenia elementu o określonym atrybucie metodą `createElementNS` a następnie dodania go do elementu w dokumencie za pomocą metody `appendChild`.

```
SVGImage svgi = SVGImage.createEmptyImage(null);
Document d = svgi.getDocument();
SVGSVGElement svge = (SVGSVGElement)d.getDocumentElement();

SVGRect vb = svge.createSVGRect ();
vb.setWidth (100);
vb.setHeight (100);
svge.setRectTrait ("viewBox", vb);

SVGElement circle = (SVGElement) d.createElementNS(
    "http://www.w3.org/2000/svg", "circle"
);
SVGRGBColor fgColor = svge.createSVGRGBColor(0, 0, 255);
circle.setRGBColorTrait("fill", fgColor);
circle.setFloatTrait("cx", 50);
circle.setFloatTrait("cy", 50);
circle.setFloatTrait("r", 20);
svge.appendChild(circle);
```

4 Założenia grafiki 3D

Tworzenie światów 3D na potrzeby urządzeń mobilnych wyposażonych w *Jave ME* zostało określone przez specyfikację *JSR-184*, nad którą prace zostały ukończone w 2003 roku. Celem tej specyfikacji było zdefiniowanie API pozwalającego na rendering trójwymiarowej grafiki w interaktywnym środowisku, opisanym przez graf [22]. Klasy tej specyfikacji zostały określone w pakiecie `javax.microedition.m3g`, a użycie wybranych zostało przedstawione w tym rozdziale.

W założeniach specyfikacji *JSR-184* autorzy określili urządzenia docelowe jako:

- urządzenia klasy CLDC,
- CPU o niskiej wydajności,
- niewielka ilość pamięci,
- brak sprzętowego wspomagania 3D,
- oraz brak typów opisujących liczby rzeczywiste.

Zastrzegli jednak iż API może być skalowane w górę, tak aby urządzenia o lepszych parametrach mogły wykorzystać swoje atuty.

Specyfikacja określa również potęcjalne aplikacje wśród których wymienia:

- gry,
- mapy,
- interfejsy użytkownika,
- wygaszacze ekranu.

Aby sprostać wymaganiom jak najszerszej grupy programów, a zarazem dbać o wygodę programistów, API zostało podzielone na dwie części: implementację niskiego poziomu opartego o specyfikację *OpenGL ES* określoną przez *Khronos*, na którą nadbudowano *graf sceny* realizujący implementację wysokiego poziomu.

Tak samo jak większość dodatkowych pakietów, tak i *M3D* opiera się na *MIDP*. Dlatego zastrzeżono, że implementacja nie powinna przekraczać 150kB wliczając w to obsługę grafiki (*graphics engine*), pliki rdzenia oraz pliki włączane.

Grafika *3D* już od bardzo długiego czasu wykorzystuje strukturę *drzewa* do opisu sceny. Tak jest również w pakiecie *M3D*, gdzie można wyróżnić trzy rodzaje obiektów sceny:

- węzeł główny,
- węzły grup,
- liście.

Najniżej w hierarchii drzewa stoją liście, którymi mogą być takie obiekty jak kamera, oświetlenie czy kształt.

Obiekty grupujące, wiążące inne obiekty mogą zostać porównane do gałęzi i konarów. Ich zadaniem jest połączenie lub nadanie pewnych dodatkowych właściwości strukturom znajdującym się w hierarchii niżej od nich.

Ostatnim komponentem drzewa jest pień, rolę tę spełniają obiekty określające położenie grup oraz rodzaj środowiska.

Wszystkie wymienione elementy drzewa znajdują odzwierciedlenie w implementujących je klasach. I tak, *pniem* będzie klasa `World`, *węzłami grupującymi* będą klasy `Group` a liśćmi `Camera`, `Sprite3D`, `Light`, `Mesh`, `MorphingMesh` i `ShinnedMesh`.

4.1 Pierwszy program

Aby rozpocząć tworzenie jakiejkolwiek sceny, potrzebny jest obiekt który obsłuży wyświetlanie. Dokumentacja dostarcza szablon przedstawiony w listingu 4.1:

```
public
    class MyCanvas
    extends Canvas{

    Graphics3D ;

    public MyCanvas(){
        myG3D = Graphics3D.getInstance();
    }

    public void paint(Graphics g) {
        try {
            myG3D.bindTarget(g);
             ... wprowadź zmniany w scenie ...
             ... wyrysuj scenę ...
        } finally {
            myG3D.releaseTarget();
        }
}
```

Listing 4.1: *Szablon w oparciu o* `Canvas`

Powyższy listing obarczony jest istotnym niedopracowaniem – nic nie wyświetla. Dzieje się tak, ponieważ scena nie zawiera żadnych elementów. Naturalnym rozwiązaniem tego problemu jest dodanie jakiegoś przykładowego obiektu, np. sześcianu, cylindra czy kuli. Jednak w pakiecie przeznaczonym dla urządzeń mobilnych nie przewidziano żadnych predefiniowanych obiektów, zatem dostępnymi rozwiązaniami są:

- utworzenie nowego obiektu,
- załadowanie modelu z pliku.

Oczywiście w przypadku pierwszego rozwiązania konieczne jest wprowadzenie kilku terminów i instrukcji, które uczyniłyby kod mniej przejrzystym. Dlatego wybrano drugie rozwiązanie, którego konsekwencją jest konieczność znalezienia metody załadowania pliku.

Jak podaje specyfikacja pakietu, przewidzano możliwość załadowania plików o rozszerzeniu `m3g`. Znaczna część narzędzi do tworzenia grafiki 3D posiada możliwość eksportowania do wymienionego formatu. W przykładach dostarczonych ze specyfikacją pakietu można znaleść plik `skaterboy.m3g` na przykładzie którego zostanie pokazana technika ładowania i wyświetlania, istniejących scen 3D.

Rysunek 4.1: *Efekt wczytania obrazu* `skaterboy.m3g`

Wykorzystana zostanie klasa `Loader` z jej statyczną metodą `load(String)` przyjmującą ścieżkę do obrazu. Wynikiem wczytania będzie tablica obiektów `Object3D`, której pierwszym elementem będzie pień drzewa. Informacje te pozwolą na utworzenie programu przedstawionego w listingu 4.2, w którym widać konwersję obiektu `Object3D` na `World` a następnie wyrysowanie zawartość drzewa metodą `render()` klasy `Graphics3D`. Uzyskany rezultat widoczny jest na rysunku 4.1.

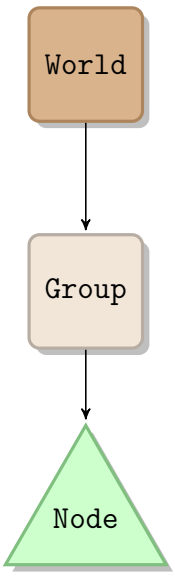
```
package tomPack;

import javax.microedition.lcdui.game.*;
import javax.microedition.m3g.*;

public
    class MyCanvas
    extends GameCanvas{

    Graphics3D myG3D = Graphics3D.getInstance();
    World root;
    public MyCanvas(){
        super(true);

        Object3D[] objects;
        try {
            objects = Loader.load("/skaterboy.m3g");
            root = (World)objects[0];
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    public void paint(Graphics g) {
        try {
            myG3D.bindTarget(g);
            myG3D.render(root);
        } finally {
            myG3D.releaseTarget();
        }
    }
}
```

Listing 4.2: *Wczytanie sceny 3D*

4.2 Struktura sceny

Jak już zaznaczono we wstępie, scena jest strukturą drzewiastą. Podstawowym elementem jest obiekt klasy `World`, a podstawowym elementem grupującym będzie obiekt klasy `Group`. Rolę *liści* pełnią obiekty klas `Sprite3D`, `Mesh`, `Light` lub `Camera`.

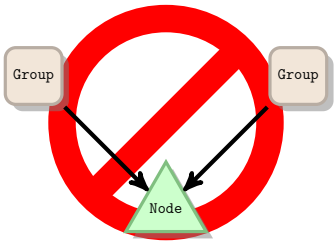
Tworzenie sceny można więc strywializować do operacji wywołania metody `addChaild` na rzecz obiektu rodzica. Efektem takiej operacji byłby diagram przedstawiony na rysunku 4.2.



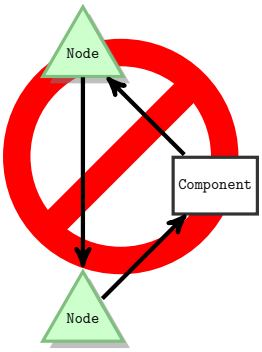
Rysunek 4.2: *Struktura prostej sceny w M3G.*

Ilość *grup* i *węzłów* jest ograniczona jedynie wydajnością urządzenia, jednak struktura sceny nie pozwala dopuścić do dwóch sytuacji:

- węzły nadrzędne drzewa nie mogą współdzielić węzłów podrzędnych



- drzewo nie może zawierać cykli



4.3 Przekształcenia afiniczne

Tworzenie wielopoziomowej struktury ma na celu ułatwienie nawigacji i uporządkowanie świata. Wynika to z podejścia nazywanego *bottom-up*, które zakłada że najprościej jest stworzyć małe elementy sceny a następnie poskładać je w coś większego. Przykładem może być tu mechanizm składania choinki, jak wiadomo *choinka* zbudowana jest z pnia, od którego odrastają gałęzie. Gałęzie rozdzielają się na

mniejsze gałązki, które z kolei wypuszczają jeszcze mniejsze gałązki aż wreszcie na samym końcu z małych gałązek wyrastają igły. Naturalnie można utworzyć program, który wygeneruje całą choinkę. Jednak praktyka pokazuje że prościej jest napisać oddzielny fragmencik kodu dla generowania *igieł*, inny dla generowania *gałązek* o różnej grubości, a następnie połączyć wyniki działania tych fragentów.

Oczywiście wyniki składania będzie należało przedstawić odpowiednio w przestrzeni, dlatego potrzebne są operacje pozwalające na *przesuwanie*, *obracanie* i *skalowanie*. Zgodnie z definicją: *przekształcenie afiniczne* jest to takie odwzorowanie jednego zbioru punktów na drugi, że funkcja odwzorowująca jest funkcją liniową z przesunięciem $y = ax + b$.

Matematyka dawno temu dostarczyła już narzędzi, które pozwalają na wymienione operacje. Wyprowadzono odpowiednie macierze:

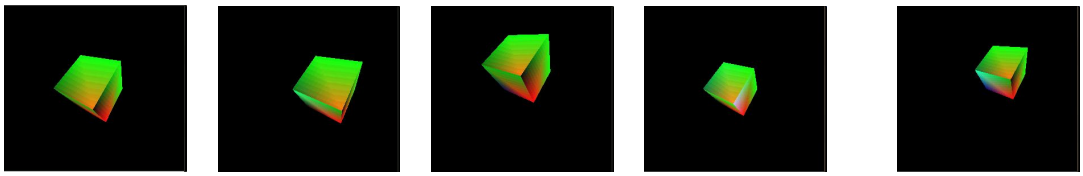
- *przesunięcia* – T,
- *obrotu* – R,
- *skalowania* – S,

które pozwalają sprawnie i szybko dokonać żądanej operacji lub dowolnego ich złożenia.

Scena *3D* nie mogłaby się obejść bez wymienionych przekształceń, dlatego w hierarchii dziedziczenia węzłów można znaleść klasę `Transformable` implementującą nastepujące metody:

- `setTranslation(...)`,
- `setOrientation(...)`,
- `setScale(...)`.

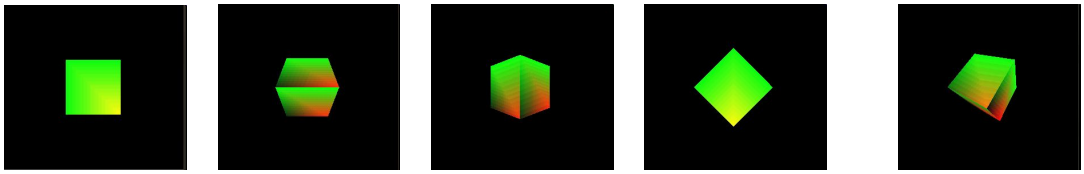
Metoda `setTranslation` oczekuje dostarczenia trzech parametrów, określających o ile jednostek na każdej z osi należy przesunąć punkt. Modyfikacje poszczególnych parametrów przedstawiono na rysunku 4.3.



Obraz wejściowy | $X + 0.5$ | $Y + 0.5$ | $Z - 0.5$ | Suma wszystkich operacji

Rysunek 4.3: *Operacje przesunięcia.*

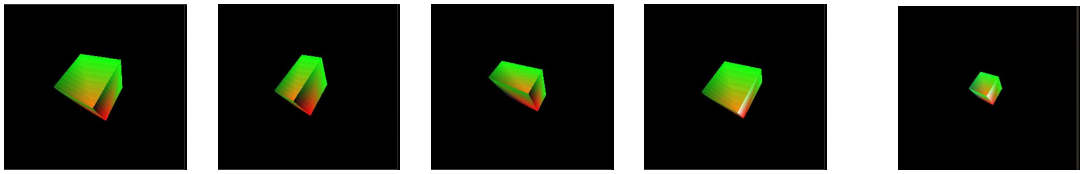
Metoda `setOrientation` wymaga czterech parametrów, są to odpowiednio kąt *(podany w stopniach)* o jaki należy dokonać obrotu i skalarny udział osi w obrocie. Wyniki cząstkowe i całościowe zostały przedstawione na rysunku 4.4.



Obraz wejściowy | obrót w osi X | obrót w osi Y | obrót w osi Z | Suma wszystkich obrotów

Rysunek 4.4: *Operacje obrotu.*

Metoda `setScale` wymaga trzech parametrów, które odpowiadają za osie X, Y, Z. Wartość osi 1.0 określa że element sceny nie ulegnie zmianie, przeskalowanie o 0.5 spowoduje zmniejszenie rozmiaru o połowę. Efekty zastosowania tej metody przedstawiono na rysunku 4.5



Obraz wejściowy | osi X w skali 0.5 | osi Y w skali 0.5 | osi Z w skali 0.5 | Wszystkie osie w skali 0.5

Rysunek 4.5: *Operacje skalowania.*

Istnieje również możliwość użycia metody `setTransform`, która jako parametr oczekuje obiektu klasy `Transform`. Klasa ta implementuje macierz liczb rzeczywistych o rozmiarach 4x4, która opisuje przekształcenia jakie należy wykanać.

Przekształcenia wykonane na najniższym węźle nie przenoszą się na węzły wyższego rzędu. Dokładnie przeciwnie jest w sytuacji odwrotnej – gdy węzły wyższego rzędu przekazują parametry otoczenia węzłom pochodnym.

4.4 Tworzenie modeli

W programie 4.2 udało się uniknąć tworzenia własnego obiektu, zastępując go załadowanym modelem. Wczytany obiekt swoją strukturę zawdzięczał środowisku do edycji grafiki *3D* co znacznie ułatwiło pokazanie pierwszego programu. Bazując na technice ładowania obiektów można stworzyć również bardzo zaawansowane programy, jednak należy wiedzieć że elementy sceny można również tworzyć w programie.

4.4.1 Modelowanie i szkieletowanie

Tworzenie elementu, wymaga dwóch niezbędnych kroków:

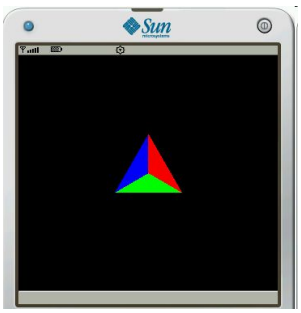
- niezbędne jest przygotowanie tablicy wierzchołków,
- połączenie wierzchołków.

Ponadto, można oczekiwać że tworzony element będzie miał jakiś kolor, własną charakterystykę dystrybucji światła czy teksturę. W tym celu należy dodatkowo dodefiniować tablice reprezentujące:

- tablice kolorów,
- tablicę normalnych,
- tablicę rozpięcia.

Dążąc do efektu przedstawionego na rysunku 4.6, utworzona zostanie klasa `Ostroslup` dziedzicząca po klasie `Group`.

Analizę wymagań dla tworzenia własnych brył należy rozpocząć od klasy która jest przystosowana do przedstawiania trójwymiarowych elementów sceny - `Mesh`. Utworzenie obiektu tej klasy wymaga dostarczenia trzech parametrów:



Rysunek 4.6: *Efekt utworzenia własnej bryły.*

- `VertexBuffer`,
- `IndexBuffer`,
- `Appearance`.

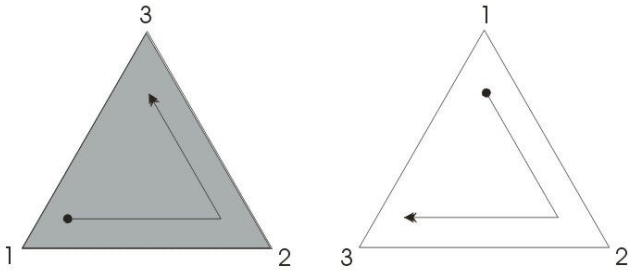
Klasa `VertexBuffer` przechowuje informacje o pozycjach, kolorach, normalnych i współrzędne tekstur, które są reprezentowane jako obiekty klasy `VertexArray`. Aby utworzyć obiekt tej klasy należy podać: ile wierzchołków będzie one przechowywać, ile wartości będzie składało się na opis oraz ile bajtów zostało przewidzianych na każdą współrzędną.

```
VertexArray vaPositions = new VertexArray(
    wierzcholki.length/3, 3, 2
);
```

Dysponując już obiektem można wprowadzić dane używając metody `set`, podając jako parametry indeks pierwszej współrzędnej, liczbę elementów które należy zastąpić oraz tablicę zawierającą dane.

Ponieważ ostrosłup będzie składał się z czterech ścian, a każda z nich będzie opisywana przez trzy punkty, z których każdy ma trzy współrzędne zgodne z osiami, łatwo policzyć że będzie potrzebna tablica o rozmiarze $4 * 3 * 3 = 26$ elementów.

Należy jednak wyjaśnić, iż w obiekcie szkieletowym każda ściana jest wykreślana (renderowana) tylko z jednej strony. Mechanizm ten ma na celu oszczędniejsze zarządzanie dostępnymi zasobami maszyny. W związku z taką implementacją użytkownik musi zwrócić uwagę na sposób, w jaki wpisuje informacje o wierzchołkach. Zgodnie z ogólną zasadą, iż kolejne punkty powinny być umieszczane przeciwnie do ruchu wskazówek zegara tak jak pokazano to rysunku 4.7.



Rysunek 4.7: *Zasada trójkątowania.*

Zatem definicja ściany powinna wyglądać następująco:

```
    private short[] wierzcholki = {
        -86, -50, -100,
         86, -50, -100,
          0,   0,  100,

         ...
    };
```

Adekwatna tablica tworzona jest dla kolorów, z tą różnicą że każde pole reprezentuje składową koloru.

Dysponując dwoma obiektami `VertexArray`, można połączyć je w obiekcie `VertexBuffer`. Dopełnienie tej czynności kończy etap przygotowania danych dla pierwszego argumentu klasy `Mesh`.

Obiekty klasy `IndexBuffer` odpowiadają za przygotowanie szkieletu obiektu. W przypadku przedstawianego programu wykorzystany zostanie mechanizm szkieletowania trójkątowego. Zdefiniowany w klase `TriangleStripArray` oczekuje dostarczenia informacji z ilu ścian będzie składał się renderowany element oraz ile wierzchołków przypada na jedną ścianę. Dane te przekazywane są jak drugi argument konstruktora.

```
    IndexBuffer ib = new TriangleStripArray( 0, new int[]{ 3, 3, 3, 3});
```

Ostatnim argumentem jest obiekt klasy `Appearance`, którego wykorzystanie zostanie omówione w punkcie 4.4.2.

```
1  lass Ostroslup
2      extends Group{
3
4      private short[] wierzcholki = {
5          -86, -50, -100,
6           86, -50, -100,
7            0,   0,  100,
8
9           86, -50, -100,
10           0, 100, -100,
11           0,   0,  100,
12
13           0, 100, -100,
14         -86, -50, -100,
15           0,   0,  100,
16
17           0, 100, -100,
18          86, -50, -100,
19         -86, -50, -100
20     };
21
22     private int[] grupowanie = {
23         3, 3, 3, 3
24     };
25
26     private final byte ff = (byte)0xff;
27
28     private byte[] kolory = {
29          0, ff, 0,
30          0, ff, 0,
31          0, ff, 0,
32
33         ff, 0, 0,
34         ff, 0, 0,
35         ff, 0, 0,
36
37          0, 0, ff,
38          0, 0, ff,
39          0, 0, ff,
40
41         ff, 0, ff,
42         ff, 0, ff,
43         ff, 0, ff
```

```
44      };
45
46      public Ostroslup(){
47
48          VertexArray vaPositions = new VertexArray(
49              wierzcholki.length/3, 3, 2
50          );
51          vaPositions.set( 0, wierzcholki.length/3, wierzcholki);
52
53          VertexArray vaColors = new VertexArray(
54              kolory.length/3, 3, 1
55          );
56          vaColors.set( 0, kolory.length/3, kolory);
57
58          VertexBuffer vb = new VertexBuffer();
59          vb.setPositions( vaPositions, 0.01f, null);
60          vb.setColors(vaColors);
61
62          IndexBuffer ib = new TriangleStripArray( 0, grupowanie);
63
64
65          Appearance ap = new Appearance();
66
67          Mesh m = new Mesh( vb, ib, ap);
68          this.addChild(m);
69      }
70  }
```

Listing 4.3: *Kompleksowa realizacja klasy* `Ostroslup`

Rozwiązanie przedstawione w listingu 4.3 było prostym zapisem bryły. Istnieje jednak możliwość uzyskania zbliżonego efektu niższym kosztem.

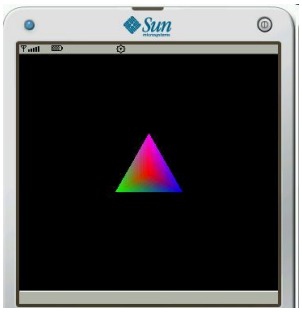
```
1   class Ostroslup
2       extends Group{
3
4       private short[] wierzcholki = {
5             0,   0,  100,
6           -86, -50, -100,
7            86, -50, -100,
8             0, 100, -100,
9       };
10
11      private int[] numerowanie = {
12          0, 1, 2,
13          0, 2, 3,
14          0, 3, 1,
15          1, 2, 3
16      };
17
18      private int[] grupowanie = {
19          3, 3, 3, 3
20      };
21
22      private final byte ff = (byte)0xff;
23
24      private byte[] kolory = {
25           ff,  0,  0,
26            0, ff,  0,
27            0,  0, ff,
28           ff,  0, ff,
29      };
30
31      public Ostroslup(){
32          VertexArray vaPositions = new VertexArray(
33              wierzcholki.length/3, 3, 2
34          );
35          vaPositions.set( 0, wierzcholki.length/3, wierzcholki);
36
37          VertexArray vaColors = new VertexArray(
38              kolory.length/3, 3, 1
39          );
40          vaColors.set( 0, kolory.length/3, kolory);
41
42
```

```
43          VertexBuffer vb = new VertexBuffer();
44          vb.setPositions( vaPositions, 0.01f, null);
45          vb.setColors(vaColors);
46
47          IndexBuffer ib = new TriangleStripArray( numerowanie, grupowanie);
48
49          Appearance ap = new Appearance();
50
51          Mesh m = new Mesh( vb, ib, ap);
52          this.addChild(m);
53      }
54  }
```

Listing 4.4: *Alternatywna realizacja klasy* `Ostroslup`

Analiza porównawcza listingów 4.3 i 4.4 słusznie pokazuje że można znacznie zredukować ilość danych zapisanych w tablicy wierzchołków (przecież ostrosłup o podstawie trójkąta ma tylko cztery wierzchołki). Jednak redukcja ta jest obarczona kosztem utworzenia nowej tablicy opisującej połączenia:

```
private int[] numerowanie = {
    0, 1, 2,
    0, 2, 3,
    0, 3, 1,
    1, 2, 3
};
```



Rysunek 4.8: *Efekt uruchomienia programu z zmodyfikowaną klasą* `Ostrosłup`.

Jak widać na rysunku 4.8 kolory przypisane do ścian uległy rozlaniu i wymieszaniu – wszak usunięto podział na ściany.

4.4.2 Powierzchnia modelu

Rysunki 4.6 i 4.8 przedstawiają efekt uzyskany przez domyślne wartości przewidziane dla klasy `Appearance` (dokumentacja pokazuje że domyślną wartością dla wszstkich pól jest `null`). Jednak pozostawienie wszystkich wartości w tym stanie ukryje przed programistą jedną z najbardziej interesujących klas pakietu *M3G* dostarczającą szereg mechanizmów wykorzystywanych podczas generowania wyglądu elementu [24].

Klasa `PolygonMode`

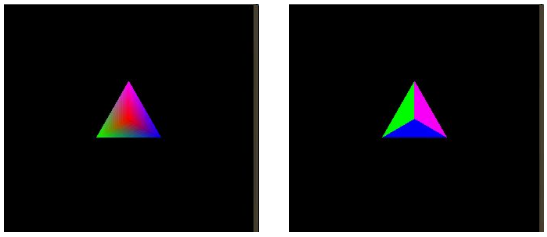
`PolygonMode` jest klasą odpowiedzialną za interpretację i obsługę danych geometrii. Można wyróżnić trzy główne zastosowania tego obiektu:

- *winding* – czyli dopasowanie systemu wyznacznia frontu trójkątów. Zgodnie z domyślnym mechanizmem opisanym w rozdziale 4.4 indeksowanie w kolejności przeciwnej do wskazówek zegara oznacza ustawienie trójkąta widoczną

powierzchnią w stronę obserwatora. Stan ten opisuje zmienna `WINDING_CCW`. Wykorzystując metodę `setWinding` i zmienne `WINDING_CW` można odwrócić domyślne ustawienia.

- *culling* – pozwala określić która ze stron trójkątu zostanie pominięta podszas *rastryzacji*. Możliwe są trzy opcje: zadna `CULL_NONE`, skierowana przodem do odbiorcy `CULL_FRONT` i skierowana tyłem do odbiorcy `CULL_BACK`.

- *shading* – określenie zasady zgodnie z którą będzie wypełniony będzie trójkąt. Przewidziano dwa stany wypełnienia: `SHADE_SMOOTH` i `SHADE_FLAT`.W pierwszym przypadku kolor jest interpolowany dla każdego piksela trójkątu, natmiast w drugim uznaje się że trójkąt należy wypełnić kolorem trzeciego wierzchołka.



Materiał

Klasa `Material` opisuje zachowanie światła padającego na daną powierzchnię, dlatego też dane zapisane w polach wchodzą w interakcje tylko gdy w scenie zostało zdefiniowane światło. Materiał został opisany przez cztery pola:

- `AMBIENT` - jest składową w równaniu które definiuje w jaki sposób powstaje odbite światło rozproszone,

- `DIFUSE` - określa kolor obiektu w każdych warunkach oświetleniowych,

- `SPECULAR` - w przypadku, gdy kąt pomiędzy źródłem światła a powierzchnią odbijającą jest bliski kątowi pomiędzy tą powierzchnią a osobą patrzącą, ma wpływ na tworzenie efektu połysku gładkiej powierzchni oraz na stopień rozproszenia koloru,

- `EMISSIVE` - określa rodzaj i natężenie światła emanującego z obiektu, niezależnie od liczby zewnętrznych źródeł oświetlenia istniejących w scenie.

Przewidziano również metody:

- `setShininess` – określającą stopień wygładzenia powierzchni oraz stopień jej połysku/matowości,

- `setVertexColorTrackingEnable` – pozwajającą uwzględnić kolor przypisany do wierzchołków.

```
Material mat = new Material();
mat.setColor(Material.AMBIENT, 0x00000080);
mat.setColor(Material.DIFFUSE, 0xff0000ff);

mat.setColor( Material.SPECULAR, 0xffffff);
mat.setColor( Material.EMISSIVE, 0x101010);

mat.setShininess(10.0f);
```

Teksturowanie

Proces modelowania geometrycznego opisany rezdziale 4.4.1 pozwala na utworzenie praktycznie każdej bryły czy płaszczyzny. Jednak algorytmiczne opisanie wszystkich elementów może być bardzo kosztowne obliczeniowo. Również projekcja takich obiektów na urządzeniach mobilnych może nastręczyć nie małych problemów.

Dlatego tworząc model zaleca się pominięcie detali, zaś na siatkę zgrubnie oddającą element należy "naciągnąć" obraz prezentujący powierzchnię. Efekty takiej operacji przedstawiono na rysunku 4.9.

Rysunek 4.9: *Efekt teksturowania modelu.*

Teksturowanie modeli w *M3G* jest realizowane przez zmianę wyglądu wyświetlanej powierzchni za którą odpowiedzialna jest klasa `Appearance`. Aby uruchomić teksturowanie modelu wystarczy więc dodać do obiektu klasy `Appearance` obiekt klasy `Texture2D`. Jednak aby utworzyć obiekt `Texture2D`, potrzebny jest obraz przechowywany pod postacią `Image2D`, która zostanie załadowana z pliku (listing 4.2).

```
        Image2D img2d = null;
        try {
            img2d = (Image2D) Loader.load("/textureImage.png")[0];
        } catch (IOException e) {
            e.printStackTrace();
        }
        Texture2D texture = new Texture2D(img2d);

        ...

        Appearance app = new Appearance();
        app.setTexture( 0, texture);
```

Dysponując teksturą należy zaznaczyć, że potrzebuje ona punktów, na których będzie rozpięta. Dlatego należy dodać charakterystykę rozpięcia tekstury do `VertexBuffera`. Operacja ta będzie wyglądać podobnie do definiowania kolorów czy wierzchołków modelu, a rozpocznie się od utworzenia tablicy przypisującej współrzędne tekstury do wierzchołków:

```
    private short[] texCoord = {
          0, 255,
        255, 255,
        255,   0,
          0,   0
    };
```

W kolejnym kroku utworzony zostanie obiekt `VertexArray`, do którego przypisana zostanie tablica koordynat:

```
        VertexArray vaTex = new VertexArray(
```

```
            texCoord.length/2, 2, 2
        );
        vaTex.set( 0, texCoord.length/2, texCoord);
```

W końcu obiekt `VertexArray` zostanie dodany do `VertexBuffera`.

```
VertexBuffer vb = new VertexBuffer();
...
vb.setTexCoords( 0, vaTex, 1.0f/256.0f, null);
```

Rezultat został przedstawiony na rysunku 4.10, a program

Płaszczyzna bez tekstury | Nakładana tekstura | Efekt końcowy

Rysunek 4.10: *Efekt nakładania tekstury na płaszczyznę.*

```
1  package tomPack;
2
3  public
4      class Plaszczyzna
5      extends Group{
6
7      private short[] wierzcholki = {
8          -50, -50, 0,
9           50, -50, 0,
10          50,  50, 0,
11         -50,  50, 0
12     };
13
14     private int[] numerowanie = {
15         0, 1, 3, 2,
16     };
17
18     private int[] grupowanie = {
19         4
20     };
21
22     private final byte ff = (byte)0xff;
23
24     private byte[] kolory = {
25           0,  0, ff,
26          ff,  0,  0,
27           0, ff,  0,
28           0, ff, ff,
29     };
30
31
32
33     private short[] texCoord = {
34         0, 255,
35         255, 255,
36         255, 0,
37         0, 0
38     };
39
40     private Image2D img2d = null;
41
42     public Plaszczyzna(){
43
44         try {
45             img2d = (Image2D) Loader.load("/texture.png")[0];
```

```
46          } catch (IOException e) {
47              e.printStackTrace();
48          }
49
50          VertexArray vaPositions = new VertexArray(
51              wierzcholki.length/3, 3, 2
52          );
53          vaPositions.set( 0, wierzcholki.length/3, wierzcholki);
54
55          VertexArray vaColors = new VertexArray(
56              kolory.length/3, 3, 1
57          );
58          vaColors.set( 0, kolory.length/3, kolory);
59
60          VertexArray vaTex = new VertexArray(
61              texCoord.length/2, 2, 2
62          );
63          vaTex.set( 0, texCoord.length/2, texCoord);
64
65          VertexBuffer vb = new VertexBuffer();
66          vb.setPositions( vaPositions, 0.01f, null);
67          vb.setColors(vaColors);
68          vb.setTexCoords( 0, vaTex, 1.0f/256.0f, null);
69
70          IndexBuffer ib = new TriangleStripArray( numerowanie, grupowanie);
71
72          Texture2D texture = new Texture2D(img2d);
73
74          Appearance ap = new Appearance();
75          ap.setTexture( 0, texture);
76
77          Mesh m = new Mesh( vb, ib, ap);
78          this.addChild(m);
79      }
80  }
```

Listing 4.5: *Realizacja teksturowania.*

Zamglenie

Kolejnym elementem wyglądu, który można sparametryzować jest `Fog`. Jego wpływ na wyniki generowania obrazu przypomina mgłę. Aby dodać ten efekt należy:

```
Fog f = new Fog(); // utworzyć obiekt Fog
...
ap.setFog(f); // ustawić go dla Appearance
```

Rezultat można zobaczyć na rysunku 4.11, zaś program realizujący ten problem można znaleźć w listingu 4.6.

Bez nałożonego zamglenia | Domyślne wartości zamglenia | Zamglenie zdefiniowane przez użytkownika

Rysunek 4.11: *Efekt zamglenia.*

Jak widać standardowe ustawienia zamglenia spowodowały zanik obiektów oddalonych więcej niż jednostkę od kamery. Dopiero modyfikacja zamglenia przez ustawienie:

- typu propagacji zamglenia `setMode`,

- gęstości `setDensity` i,
- koloru `setColor`

pozwoliły na zwizualizowanie zmian.

```
1  package tomPack;
2
3  import java.io.*;
4
5  import javax.microedition.midlet.*;
6  import javax.microedition.lcdui.*;
7  import javax.microedition.lcdui.game.*;
8
9  import javax.microedition.m3g.*;
10
11
12 public
13 class HelloMidlet
14     extends MIDlet{
15
16     private Display display;
17     private Canvas c;
18
19     public HelloMidlet(){
20         c = new MyCanvas();
21     }
22
23     public void startApp(){
24         display = Display.getDisplay(this);
25         display.setCurrent(c);
26
27     }
28     public void pauseApp(){
29     }
30
31     public void destroyApp(boolean unconditional) {
32     }
33 }
34
35 class MyCanvas
36     extends GameCanvas{
37
38     Graphics3D g3d = Graphics3D.getInstance();
39     World root;
40
41     Group g;
42     Camera c;
43
44     public MyCanvas(){
45         super(true);
46
47         Plaszczyzna p1 = new Plaszczyzna();
48         Plaszczyzna p2 = new Plaszczyzna();
49         Plaszczyzna p3 = new Plaszczyzna();
50
51         p1.setTranslation( -0.5f, 0.0f, 0.75f);
52         p2.setTranslation(  0.0f, 0.0f, 0.0f);
53         p3.setTranslation(  0.5f, 0.0f,-0.5f);
54
55         g = new Group();
56         g.addChild(p1);
57         g.addChild(p2);
58         g.addChild(p3);
59
60         root = new World();
61         root.addChild(g);
62
63         c = new Camera();
64         c.setPerspective( 90f, ((float)getWidth()/(float)getHeight()), 0.1f, 50.f↩
               );
65         c.setTranslation( 0f, 0f, 1.5f);
66
67         root.addChild(c);
```

```
68          root.setActiveCamera(c);
69      }
70
71      public void paint(Graphics g) {
72          g3d.bindTarget(g, true, Graphics3D.DITHER | Graphics3D.TRUE_COLOR);
73          g3d.setCamera( c, null);
74          try{
75              g3d.render(root);
76          }catch(Exception ex){
77              ex.printStackTrace();
78          }
79          g3d.releaseTarget();
80      }
81  }
82
83  class Plaszczyzna
84      extends Group{
85
86      private short[] wierzcholki = {
87          -50, -50, 0,
88           50, -50, 0,
89           50,  50, 0,
90          -50,  50, 0
91      };
92
93      private int[] numerowanie = {
94          0, 1, 3, 2,
95      };
96
97      private int[] grupowanie = {
98          4
99      };
100
101     private final byte ff = (byte)0xff;
102
103     private byte[] kolory = {
104           0,  0, ff,
105          ff,  0,  0,
106           0, ff,  0,
107           0, ff, ff,
108     };
109
110     private Image2D img2d = null;
111
112     private short[] texCoord = {
113         0, 255,
114         255, 255,
115         255, 0,
116         0, 0
117     };
118
119     public Plaszczyzna(){
120
121         try {
122             img2d = (Image2D) Loader.load("/texture.png")[0];
123         } catch (IOException e) {
124             e.printStackTrace();
125         }
126
127         VertexArray vaPositions = new VertexArray(
128             wierzcholki.length/3, 3, 2
129         );
130
131         vaPositions.set( 0, wierzcholki.length/3, wierzcholki);
132
133         VertexArray vaColors = new VertexArray(
134             kolory.length/3, 3, 1
135         );
136         vaColors.set( 0, kolory.length/3, kolory);
137
138         VertexArray vaTex = new VertexArray(
139             texCoord.length/2, 2, 2
140         );
141         vaTex.set( 0, texCoord.length/2, texCoord);
142
143
```

```
144         VertexBuffer vb = new VertexBuffer();
145         vb.setPositions( vaPositions, 0.01f, null);
146         vb.setColors(vaColors);
147         vb.setTexCoords( 0, vaTex, 1.0f/256.0f, null);
148 
149         IndexBuffer ib = new TriangleStripArray( numerowanie, grupowanie);
150 
151         Texture2D texture = new Texture2D(img2d);
152 
153         Fog f = new Fog();
154         f.setMode(Fog.EXPONENTIAL);
155         f.setDensity(0.75f);
156         f.setColor(0xff8866);
157 
158         Appearance ap = new Appearance();
159         ap.setTexture( 0, texture);
160         ap.setFog(f);
161 
162 
163         Mesh m = new Mesh( vb, ib, ap);
164         this.addChild(m);
165     }
166 }
```

Listing 4.6: *Program przedstawiający efekt zamglenia powierzchni.*

4.5 Oświetlenie

Następnym elementem sceny, który powinien zostać opisany jest klasa `Light`. Jej definicja jest bezpośrednią implementacją oświetlenia opisanego w standardzie *OpenGL*. Dlatego zarówno mechanizmy jak i uzyskiwane rezultaty będą zbliżone.

Korzystając ze standardowego konstruktora tworzy się obiekt.

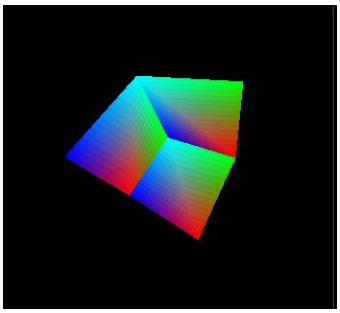
```
    Light l = new Light();
```

Następnie zostaje on umieszczony w strukturze sceny, np:

```
    root.addChild(l);
```

Tak spreparowany węzeł tworzy światło kierunkowe o białym zabarwieniu, które domyślnie skierowane jest przeciwnie do kierunku osi Z. Zmiana ukierunkowania światła *(jeśli jest to konieczne)* sprowadza się do zaaplikowania odpowiedniego przekształcenia, jako że klasa `Light` ma w swojej hierarchii dziedziczenia klasę `Transformable`.

Po stworzeniu sceny z trzema ścianami i domyślnym oświetleniem, rezultat (rys. 4.12) przedstawia scenę która niczym nie różni się od sceny nie oświetlonej.



Rysunek 4.12: *Obraz bazowy oświetlenia sceny.*

Dzieje się tak, ponieważ domyślnie utworzona powierzchnia jako `Material` ustawia wartość `null`. Dlatego utworzono materiał:

```
Material mat = new Material();

mat.setColor(Material.AMBIENT, 0x00000080);
mat.setColor(Material.DIFFUSE, 0xff0000ff);

mat.setColor( Material.SPECULAR, 0xffffff);
mat.setColor( Material.EMISSIVE, 0x101010);

mat.setShininess(10.0f);
```

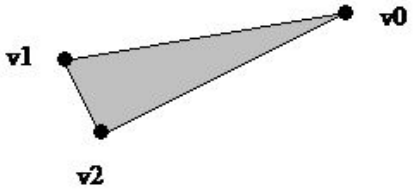
który umieszczono jako część powierzchni każdej z płaszczyzn.

Rysunek 4.13: *Ściany oświetlone bez normalnych.*

Jak pokazuje rysunek 4.13, dodanie materiału spowodowało pewien regres. Wprawdzie ściany są widoczne, co zawdzięczamy ustawionemu w materiale polu `EMISSIVE`, to jednak ściany straciły kolory. Spowodowane jest to faktem iż podczas tworzenia modelu nie zdefiniowano *wektorów normalnych*, skutkiem czego oświetlenie nie może wyliczyć swojego udziału w poszczególnych wierzchołkach.

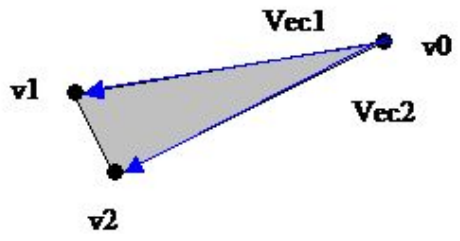
4.5.1 Wektory normalne

Wektor normalny definiowany jest jako wektor prostopadły do płaszczyzny stycznej do powierzchni w danym punkcie. Dysponując pojedynczym trójkątem, tak jak na rysunku 4.14, najeży określić wierzchołek dla którego będzie liczony wektor normalny.



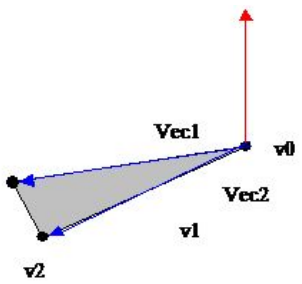
Rysunek 4.14: *Trójkąt do wyliczenia wektora normalnego.*

W ilustrowanym przypadku wybrano punkt $v0$. Na bazie punktów $v0v1$ i $v0v2$, wyznaczono wektory $Vec1$ i $Vec2$.



Rysunek 4.15: *Wyznaczone wektory $Vec1$ i $Vec2$.*

Następną operacją będzie wyliczenie iloczynu kartezjańskiego wektorów $Vec1$ i $Vec2$ wynikiem czego będzie wektor $VecW$



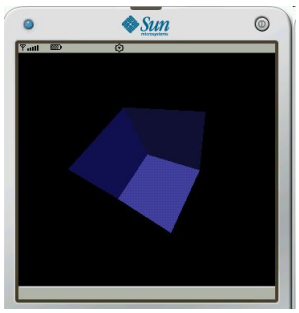
Rysunek 4.16: *Wyznaczony wektor normalny.*

Ostatnia operacja to znormalizowanie wektora $VecW$.

W przypadku trójkąta rozpiętego na płaszczyźnie XY nie trudno obliczyć że będzie to wektor skierowany zgodnie z osią Z. W związku tym modyfikacjom poddany zostanie obiekt klasy `VertexBuffer`.

```
private byte[] normalne = {
    0, 0, 127,
    0, 0, 127,
    0, 0, 127,
    0, 0, 127
};

//...

VertexArray vaNormals = new VertexArray(
    normalne.length/3, 3, 1
);
vaNormals.set( 0, normalne.length/3, normalne);

//...

VertexBuffer vb = new VertexBuffer();
vb.setNormals(vaNormals);
```

Po wprowadzeniu niezbędnych poprawek, efekt działania programu został przedstawiony na rysunku 4.17.

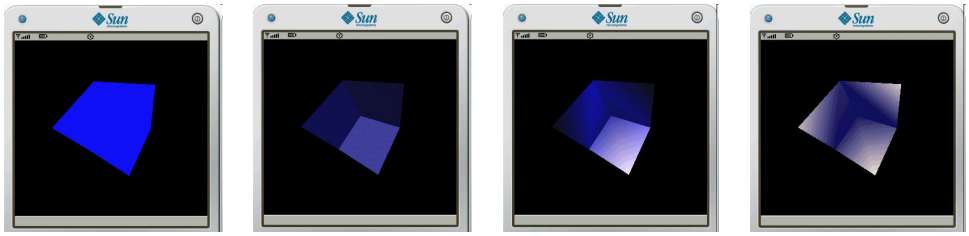


Rysunek 4.17: *Domyślne światło.*

Węzeł `Light` może występować w jednym z czterech typów:

- `AMBIENT` – światło tego typu pozornie dobiega ze wszystkich kierunków równomiernie i nie jest możliwe zlokalizowanie jego źródła,
- `DIRECTIONAL` – reprezentujący światło ukierunkowane,
- `OMNI` – światło punktowe,
- `SPOT` – światło reflektorowe.

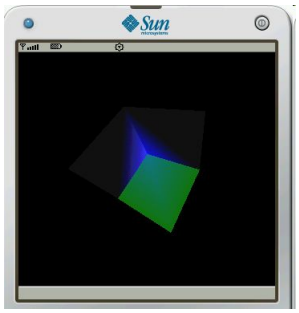
Efekty jaki dają różne rodzaje światła przedstawiono na rysunku 4.18.



AMBIENT DIRCTIONAL OMNI SPOT

Rysunek 4.18: *Efekt działania różnych rodzajów światła.*

Może się jednak okazać że w scenia będzie więcej niż jedno źródło i typ światła. W przykładzie opisanym listingiem 4.7 połączono opisywane światła, a uzyskany efekt pokazuje rysunek 4.19.



Rysunek 4.19: *Połączone światła.*

```
package tomPack;

import java.io.*;

```

```
5  import javax.microedition.midlet.*;
6  import javax.microedition.lcdui.*;
7  import javax.microedition.lcdui.game.GameCanvas;
8
9  import javax.microedition.m3g.*;
10
11 import org.w3c.dom.*;
12 import org.w3c.dom.svg.*;
13
14 public
15 class HelloMidlet
16     extends MIDlet{
17
18     private Display display;
19     private Canvas c;
20
21     public HelloMidlet(){
22         c = new MyCanvas();
23     }
24
25     public void startApp(){
26         display = Display.getDisplay(this);
27         display.setCurrent(c);
28
29     }
30     public void pauseApp(){
31     }
32
33     public void destroyApp(boolean unconditional) {
34     }
35 }
36
37 class MyCanvas
38     extends GameCanvas {
39
40     Graphics3D g3d = Graphics3D.getInstance();
41     World root;
42
43     Group g;
44     int kat;
45     float X, Y, Z;
46
47
48     Light s1, s2, s3;
49     Transform t;
50
51     Camera c;
52
53     public MyCanvas(){
54         super(true);
55
56         Plaszczyzna p1 = new Plaszczyzna();
57         p1.setOrientation( 0f, 0, 0, 1);
58         p1.setTranslation( 0.0f, 0.0f, -0.5f);
59
60         Plaszczyzna p2 = new Plaszczyzna();
61         p2.setOrientation( 90f, 0, 1, 0);
62         p2.setTranslation( -0.5f, 0.0f, 0.0f);
63
64         Plaszczyzna p3 = new Plaszczyzna();
65         p3.setOrientation( 90f, 1, 0, 0);
66         p3.setTranslation( 0.0f, 0.5f, 0.0f);
67
68         g = new Group();
69         g.addChild(p1);
70         g.addChild(p2);
71         g.addChild(p3);
72
73
74         g.setTranslation( 0, 0, -0.0f);
75         g.setOrientation( 45f, -1.0f, -1.0f, -1.0f);
76
77
78         s1 = new Light();
79         s1.setMode(Light.AMBIENT);
80         s1.setColor(0x500000);
```

```
81          s1.setIntensity(100.0f);
82
83          s2 = new Light();
84          s2.setMode(Light.DIRECTIONAL);
85          s2.setColor(0x00ff00);
86          s2.setIntensity(2.f);
87
88          s3 = new Light();
89          s3.setTranslation( 0.5f,-0.5f, 0.5f);
90          s3.setOrientation( 45f, 1.f, 1.f, 1.f);
91          s3.setMode(Light.SPOT);
92          s3.setColor(0xFFEE88);
93          s3.setIntensity(5.0f);
94          s3.setSpotAngle(20.0f);
95          s3.setSpotExponent(1.0f);
96          s3.setAttenuation( 0.0f, 1.0f, 0.0f);
97
98          Light s4 = new Light();
99          s4.setTranslation( 0.f, 0.f, 0.f);
100         s4.setMode(Light.OMNI);
101         s4.setColor(0x00FFFF);
102         s4.setIntensity(0.01f);
103
104         root = new World();
105         root.addChild(g);
106         root.addChild(s1);
107         root.addChild(s2);
108         root.addChild(s3);
109
110         c = new Camera();
111         c.setPerspective( 90f, ((float)getWidth()/(float)getHeight()), 1.0f, 50.f↩
                );
112         c.setTranslation( 0f, 0f, 1.5f);
113
114         root.addChild(c);
115         root.setActiveCamera(c);
116     }
117
118     public void paint(Graphics g) {
119         g3d.bindTarget(g, true, Graphics3D.DITHER | Graphics3D.TRUE_COLOR);
120         g3d.setCamera( c, null);
121         try{
122             g3d.render(root);
123         }catch(Exception ex){
124             ex.printStackTrace();
125         }
126         g3d.releaseTarget();
127     }
128 }
129
130 class Plaszczyzna
131     extends Group{
132
133     private short[] wierzcholki = {
134         -50, -50, 0,
135          50, -50, 0,
136          50,  50, 0,
137         -50,  50, 0
138     };
139
140     private int[] numerowanie = {
141         0, 1, 3, 2,
142     };
143
144     private int[] grupowanie = {
145         4
146     };
147
148     private final byte ff = (byte)0xff;
149
150     private byte[] kolory = {
151           0,  0, ff,
152          ff,  0,  0,
153           0, ff,  0,
154           0, ff, ff,
155     };
```

```
156
157     private byte[] normalne = {
158         0, 0, 127,
159         0, 0, 127,
160         0, 0, 127,
161         0, 0, 127
162     };
163
164     public Plaszczyzna(){
165
166         VertexArray vaPositions = new VertexArray(
167             wierzcholki.length/3, 3, 2
168         );
169
170         vaPositions.set( 0, wierzcholki.length/3, wierzcholki);
171
172         VertexArray vaColors = new VertexArray(
173             kolory.length/3, 3, 1
174         );
175         vaColors.set( 0, kolory.length/3, kolory);
176
177         VertexArray vaNormals = new VertexArray(
178             normalne.length/3, 3, 1
179         );
180         vaNormals.set( 0, normalne.length/3, normalne);
181
182         VertexBuffer vb = new VertexBuffer();
183         vb.setPositions( vaPositions, 0.01f, null);
184         vb.setColors(vaColors);
185         vb.setNormals(vaNormals);
186
187         IndexBuffer ib = new TriangleStripArray( numerowanie, grupowanie);
188
189         Material mat = new Material();
190
191         mat.setColor(Material.AMBIENT, 0x00000080);
192         mat.setColor(Material.DIFFUSE, 0xff0000ff);
193
194         mat.setColor( Material.SPECULAR, 0xffffff);
195         mat.setColor( Material.EMISSIVE, 0x101010);
196
197         mat.setShininess(10.0f);
198
199         Appearance ap = new Appearance();
200         ap.setMaterial(mat);
201
202         Mesh m = new Mesh( vb, ib, ap);
203         this.addChild(m);
204     }
205 }
```

Listing 4.7: *Program przedstawiający efekt oświetlenia powierzchni.*

5 Przechowyanie danych

Urządzenia przenośne technologii *J2ME* we wszystkich swoich profilach miały ograniczone zasoby pamięci. Z racji mobilności nie można było zastosować tracycyjnego dysku, na którym zapisywane byłyby dane. To uwarunkowania techniczne i energetyczne spowodowały, że wydzielono pamięć trwałą i ulotną.

Oczywiście systemy operacyjne musiały zezwolić na zapis pewnej ilości informacji, ponieważ taki dodatek jak np. książka telefoniczna musiał przecież przechowywać informacje o numerach. Powszechnie stosowanym rozwiązaniem jest domyślne umieszczanie tych informacji na karcie *SIM*, która jednak cechuje się bardzo długim czasem dostępu. Często po włączeniu urządzenia i próbie dostępu do książki telefonicznej należy odczekać nawet kilka minut aż odczytane zostaną potrzebne informacje.

Kolejną niedogodnością wynikającą z braku standaryzacji jest mnogość systemów organizacji danych w pamięci. Praktycznie każdy liczący się wytwórca posiada własne patenty w tej dziedzinie.

Jednakże *Java* dążąca do spełnienia postulatu '*napisz raz, uruchamiaj wszędzie*', musiała pozwalać na zunifikowany dostęp do danych bardzo szerokiej rodzinie urządzeń. Dlatego odniesiono się do idei relacyjnych baz danych i ich konkurencji. O implementacji pełnej wersji *RDBMS* nie mogło być mowy, jednak niewielki podzbiór opisu bazodanowego został uznany za dobry pomysł. [28, 23]

5.1 System zarządzania rekordami

Record Managment System(*RMS*) został wprowadzony jako mechanizm pozwalający na trwałe przechowywanie informacji. W literaturze [23] można znaleźć bezpośrednie porównania do pakietu *Java DataBase Connection*, i rzeczywiście ideowo są one bardzo zbliżone, jednak praktyka pokazuje że *RMS* jest mechanizmem znacznie prostszym.

Podstawą jednostkę w systemie *RMS* stanowi mała baza danych nazywana *Record store*, a reprezentuje ją klasa `javax.microedition.rms.RecordStore`. Informacje przechowywane w bazie noszą nazwę *rekordów*, które są reprezentowane jako tablice bajtów.

Zadaniem dowolnej realizacji *RMS* jest lokalizacja bazy danych niezależnie od fizycznego miejsca przechowywania informacji. Proces ten jest realizowany na podstawie nazwy składającej się z ciągu znaków o maksymalnej długości 32.

Lokalizacja bazy nie gwarantuje jednak jej otwarcia. Kluczowym elementem są uprawnienia, dostęp do bazy może być:

- ograniczony – w przypadku gdy *MIDlet* utworzy bazę w obrębie swojego pakietu,
- współdzielony – gdy nie stanowi części pakietu.

Uzyskanie dostępu do jednej z baz nie ogranicza możliwości otwarcia kolejnej.

Na otwartej bazie można wykonywać operacje na rekordach, każda modyfikacja będzie skutkować zapisaniem dnia i godziny w postaci wartości uzyskanej z metody `System.currentTimeMilis()`. Należy zaznaczyć że pakiet *RMS* nie zawiera żadnych mechnizmów zapewniających spójność bazy, zadanie to jest w całości delegowane na platformę uruchamiającą aplikację.

Ostatnim etapem życia bazy danych będzie jej usunięcie. Może to nastąpić w dwojakich okolicznościach, gdy:

- doprowadzono do wykonania metody usuwającej,
- usunięto *MIDlet*.

Niezależnie od wyboru metody wszelkie dane powinny zostać usunięte i podobnie jak to miało miejsce z integralnością bazy, to realizacja jest odpowiedzialna za wykonanie tego procesu.

5.1.1 Podstawowa aplikacja przechowująca dane.

Utworzenie lub otworzenie bazy danych jest wykonywane przez tę samą statyczną metodę `openRecordStore(String rsName, boolean createIfNecessary)` z klasy `RecordStore`. Pierwszy z parametrów określa nazwę poszukiwanej bazy, natomiast drugi warunkuje logicznie utworzenie nowej bazy w przestrzeni *MIDlet*u gdy lokalizacja istniejącej bazy się nie powiodła.

```
try {
    RecordStore rs = RecordStore.openRecordStore("MojaBaza", true);
}catch(Exception ex){
}
```

Listing 5.1: *Podstawowe utworzenie lub otworzenie bazy RMS*

Współdzielenie bazy *RMS*

Jak już wspomniano, istnieje możliwość współdzielenia baz *RMS*. Analiza dokumentacji wykaże, że przedstawiona w listingu 5.1 metoda `openRecordStore` pochodzi z *MIDP 1.0*. Dopiero kolejna wersja profilu wprowadziła możliwość współdzielenia, osiągnięto to dodając dwie metody:

- `openRecordStore`,
- `setMode`.

Obie przyjmują jedną z dwóch wartości `AUTHMODE_ANY` lub `AUTHMODE_PRIVATE`, które odpowiednio opisują dostęp wolny i restrykcyjny. Ostatnim parametrem wymagającym opisania jest wartość logiczna przyjmująca stan `true` jeżeli do bazy zapisu może dokonywać inny *MIDlet* i `false` w przeciwnym przypadku.

Rekordy bazy *RMS*

Operacje na bazie można wykonywać tylko jeżeli została ona poprawnie otworzona, w przeciwnym wypadku zwrócony zostanie wyjątek `RecordStoreNotOpen`.

Jako że pusta baza danych nie będzie zawierała żadnych *rekordów*, to należy dodać jakąś informację. Posłuży do tego metoda `addRecord(byte[] data, int`

`offset, int numBytes)`, która jako pierwszy parametr oczekuje tablicy bajtów. Pozyskanie obiektu typu `byte[]` jest prostą operacją gdy dysponujemy obiektem np. typu `String`.

```
String str = "Hello world.";
byte[] data = str.getBytes();

int wrt = rs.addRecord( data, 0, data.length);
```

Listing 5.2: *Dodanie nowego rekordu do bazy RMS*

Wynikiem poprawnie zakończonej operacji dodania rekordu, będzie zwrócenie zmiennej `int` pokazującej id rekordu. Może on posłużyć do pozyskania tego rekordu:

```
byte[] newData = new byte[rs.getRecordSize(wrt)];
rs.getRecord( wrt, newData, 0);
String newStr = new String(newData);
```

Listing 5.3: *Pobranie rekordu z bazy RMS*

Enumeracja

Należy jednak postawić pytanie jak często programista będzie spotykać się z sytuacją w której konieczne będzie pozyskanie dopiero co zapisanego rekordu? Najczęstszą odpowiedzą będzie – niezwykle rzadko, a ta prawdziwa odpowiedz implikuje sobą konieczność pozyskania informacji o ilości zapisanych rekordów i ich identyfikatorach. Posłużyć temu może metoda `getNumRecords()`, która zwróci ilość rekodów przechowywanych w bazie. Następnie odwołując się do kolejnych identyfikatorów można uzyskać wszystkie rekordy.

Istnieje również mechanizm pozwalający na wyszeregowanie rekordów. Służy do tego metoda `enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated)`, która zwróci obiekt klasy `RecordEnumeration`. Zanim to jednak nastąpi wymagane jest dostarczenie parametrów.

Pierwszym argumentem jest interfejs `RecordFilter`, który wymaga zaimplementowania metody `matches(byte[] candidate)`. W listingu 5.4 przedstawiono implementację anonimowej klasy wewnętrznej, która sprawdza czy pozyskany rekord rozpoczyna się od słowa “Hello”.

```
RecordFilter rf = new RecordFilter(){

    byte[] req = "Hello".getBytes();

    public boolean matches(byte[] candidate){
        if(candidate.length < req.length){
            return false;
        }else{
            for(int i=0; i<req.length; i++){
                if(candidate[i] != req[i])
                    return false;
            }
            return true;
        }
    }
};
```

Listing 5.4: *Implementacja interfejsu `RecordFilter`*

Drugim parametrem jest obiekt utworzony na podstawie interfejsu `RecordComparator`. Podobnie jak to miało miejsce w opracowaniu pierwszego

parametru, tak samo i tu konieczne jest zaimplementowanie wymaganej metody. Jako klasyfikator przyjęto w tym wypadku długość dostarczonej tablicy bajtów.

```
RecordComparator rc = new RecordComparator(){
    public int compare(byte[] rec1, byte[] rec2){
        if(rec1.length < rec2.length)
            return RecordComparator.PRECEDES;
        else
            if(rec1.length > rec2.length)
                return RecordComparator.FOLLOWS;

        return RecordComparator.EQUIVALENT;
    }
};
```

Listing 5.5: *Implementacja interfejsu* `RecordComparator`

Dysponując argumentami pozwalającymi na enumerację, można przystąpić do pozyskania i wykorzytania samej enumeracji.

```
RecordEnumeration re = rs.enumerateRecords( rf, rc, true);
while(re.hasNextElement()){
    String str = new String(re.nextRecord());
}
```

Listing 5.6: *Pozyskanie enumeratora.*

5.1.2 MIDMedic

Program ten został stworzony dla potrzeb ludzi chorujących na cukrzycę, zmuszonych do prowadzenia regularnych pomiarów poziomu glukozy we krwi oraz odnotowywania ich wyników, również wówczas gdy chory przebywa poza domem. Telefon komórkowy jako urządzenie stale noszone przez właściciela okazuje się być doskonałym nośnikiem do gromadzenia takich zapisów.

Dane wprowadzane z klawiatury zostają opatrzone markerem czasowym, a następnie zapisywane są do bazy *RMS*. Można je odczytać, a odczytane zaprezentować w postaci graficznego wykresu (z zaznaczonym zakresem prawidłowych wartości) lub w formie tabeli z kolumnami 'Data', 'Stan' oraz 'Ocena'.

Pierwszym ekranem widocznym dla uzytkownika po uruchomieniu aplikacji jest ekran wyboru. Utworzono go przy użyciu obiektu klasy `List`, w której zdefiniowano trzy opcje wyboru. Należy zaznaczyć, iż implementacja tego ekranu została dodatkowo wzbogacona o obiekt nasłuchujący zdarzenia, przez co wybór którejkolwiek opcji implikował podjęcie przypisanej mu akcji. Użytkownik dysponuje zatem trzema możliwościami:

- 'Dane' -– otwiera ekran dedykowany wprowadzaniu danych,
- 'Statystyka' – rysuje wykres przedstawiający wahania poziomu cukru we krwi w okresie ostatnich 50 badań,
- 'Tabela' – przedstawienie zachowanych danych w postaci tabeli.

Moduł 'Dane' był jednym z prostszych – należało określić sposób wprowadzania danych. Ze względów estetycznych zrezygnowno tu z standardowego komponentu TextBox – zajmował on cały ekran i nie informował należycie użytkownika, jakich danych oczekuje aplikacja. Dlatego też użyto klasy Form, do obiektu której dodano komórkę z możliwością wprowadzania danych. Wykorzystano klasę `TextField` określając tekst jakim będzie poprzedzona graficzna reprezentacja komponentu i

jednocześnie definiując iż komponent może przyjąć tylko 3 cyfry. Do tak wizualnie przygotowanej klasy dodano jeszcze obsługę w postaci dwóch przycisków (`Command`) które pozwalają odpowiednio na powrót do poprzedniego ekranu albo dodanie wprowadzonych danych do bazy.

Moduł 'Statystyka' był nieco bardziej pracochłonny, gdyż wyświetlane dane winny dostosować się do obszaru okna, jakim dysponuje użytkownik w swoim urządzeniu. Przy implementacji klasy `MyStatisticForm` wykorzystano bazową klasę `Canvas`, pozwalającą programiście na samodzielne definiowanie komponentów graficznych.

Ostatnim modułem była 'Tabela'.

tryb wyświetlania wykresu

tryb wyświetlania tabeli

Rysunek 5.1: *Aplikacja MIDMedic.*

```
*********************************
*    HelloMidlet.java
*********************************
package mtPack;

import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;

public
class HelloMidlet
    extends MIDlet{

    public Display display;

    public HelloMidlet(){
    }

    public void startApp(){
        display = Display.getDisplay(this);
        display.setCurrent(new MyList(this));
    }

    public void pauseApp(){
    }

    public void destroyApp(boolean flag){
    }
}


*********************************
*    MyList.java
*********************************
package mtPack;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;

class MyList
    extends List
    implements CommandListener{

    Command doneCommand;
    Command enterCommand;
    HelloMidlet parent;

```

```
47      public MyList(HelloMidlet hellomidlet){
48          super( "Wstepny wybor:", List.IMPLICIT,
49              new String[] {
50                  "Dane", "Statystyka", "Tabela", "Komunikacja"
51              }, null
52          );
53          parent = hellomidlet;
54          doneCommand = new Command("Wyjdz", 7, 1);
55          enterCommand = new Command("Wejdz", 1, 1);
56          addCommand(enterCommand);
57          addCommand(doneCommand);
58          setCommandListener(this);
59      }
60
61      public void commandAction(Command command, Displayable displayable){
62          if(displayable == this)
63              if(command == doneCommand) {
64                  parent.destroyApp(false);
65                  parent.notifyDestroyed();
66              } else
67                  if(command == enterCommand)
68                      switch(getSelectedIndex()){
69                          case 0:
70                              parent.display.setCurrent(new MyText(this));
71                              break;
72
73                          case 1:
74                              parent.display.setCurrent(New MyStatisticForm(this));
75                              break;
76
77                          case 2:
78                              parent.display.setCurrent(new MyTableForm1(this));
79                              break;
80                          case 3:
81                              parent.display.setCurrent(new MyCommunication(this));
82                              break;
83                      }
84      }
85  }
86
87  ********************************
88  *    MyStatisticForm.java
89  ********************************
90  package mtPack;
91
92  import java.io.PrintStream;
93  import java.util.Vector;
94  import javax.microedition.lcdui.*;
95  import javax.microedition.rms.RecordStore;
96  import javax.microedition.rms.RecordStoreException;
97
98  public
99  class MyStatisticForm
100     extends Canvas
101     implements CommandListener{
102
103     private MyList parent;
104     private Vector vec;
105     private RecordStore rs = null;
106     private Command backCommand;
107     private int wys = 0;
108     private int sze = 0;
109     private String tomStr;
110
111     public MyStatisticForm(MyList mylist){
112         parent = mylist;
113         backCommand = new Command("Powrot", 2, 1);
114         addCommand(backCommand);
115         setCommandListener(this);
116         vec = new Vector();
117         try{
118             rs = RecordStore.openRecordStore(Names.dbName, false);
119             for(int i = 1; i <= rs.getNumRecords(); i++)
120                 vec.addElement(Wpis.setValues(rs.getRecord(i)));
121         }catch(RecordStoreException recordstoreexception) {
122             System.out.println("Allert" + recordstoreexception);
```

```
        }
        wys = getHeight();
        sze = getWidth();
    }

    public void paint(Graphics g){
        g.setColor(255, 255, 255);
        g.fillRect(0, 0, wys, sze);
        g.setColor(0, 0, 0);
        g.drawLine(0, wys - 3, sze, wys - 3);
        g.drawLine(3, 0, 3, wys);
        for(int i = 0; i < sze; i += 3)
            g.drawLine(i, wys - 3, i, wys);

        for(int j = wys - 3; j > 0; j -= 3)
            g.drawLine(0, j, 3, j);

        int k = 0;
        for(int l = 0; l < vec.size(); l++){
            Wpis wpis = (Wpis)vec.elementAt(l);
            int i1 = wpis.getWartosc() / 10;
            g.drawLine((l + 1) * 3, wys - 3 - k, (l + 2) * 3, wys - 3 - i1);
            k = i1;
        }
    }

    public void commandAction(Command command, Displayable displayable){
        if(displayable == this && command == backCommand)
            parent.parent.display.setCurrent(parent);
    }
}

*********************************
*    MyTableForm1.java
*********************************
package mtPack;

import java.io.PrintStream;
import java.util.Calendar;
import java.util.Vector;
import javax.microedition.lcdui.*;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;


class MyTableForm1
    extends Canvas
    implements CommandListener{

    public MyTableForm1(MyList mylist){
        wys = 0;
        sze = 0;
        rs = null;
        max = new int[3];
        vScroll = false;
        hScroll = false;
        cal = Calendar.getInstance();
        startDisp = 0;
        xTrans = 0;
        parent = mylist;
        backCommand = new Command("Powrot", 2, 1);
        addCommand(backCommand);
        setCommandListener(this);
        vec = new Vector();
        font = Font.getFont(0, 0, 0);
        for(int i = 0; i < nazwy.length; i++)
            max[i] = font.stringWidth(nazwy[i]);

        try{
            rs = RecordStore.openRecordStore(Names.dbName, false);
            for(int j = 1; j <= rs.getNumRecords(); j++){
                Wpis wpis = Wpis.setValues(rs.getRecord(j));
                String s = cal.get(1) + "/" + (cal.get(2) + 1) + "/" + cal.get(5)↩
                    + " " + cal.get(10) + ":" + cal.get(12) + " " + (cal.get(9) ↩
                    != 0 ? "PM" : "AM");
                int k = font.stringWidth(s);
```

```
                if(max[0] < k)
                    max[0] = k;
                k = font.stringWidth(wpis.getWartosc() + "");
                if(max[1] < k)
                    max[1] = k;
                vec.addElement(wpis);
            }

        }catch(RecordStoreException recordstoreexception) {
            System.out.println("Allert" + recordstoreexception);
        }
        wys = getHeight();
        sze = getWidth();
        if(max[0] + 3 + max[1] + 3 + max[2] > sze){
            hScroll = true;
            wys -= 5;
        }
        if((vec.size() + 1) * font.getHeight() > wys)
            vScroll = true;
        fontHeight = font.getHeight();
        ilosc = (wys - fontHeight) / fontHeight;
    }

    public void paint(Graphics g){
        g.setColor(255, 255, 255);
        g.fillRect(0, 0, wys, sze);
        g.translate(xTrans, 0);
        g.setFont(font);
        g.setColor(0, 0, 0);
        int i = 0;
        for(int j = 0; j < nazwy.length; j++){
            g.drawString(nazwy[j], i + max[j] / 2 + j * 3, 0, 17);
            i += max[j];
            g.drawLine(i + j * 3 + 1, 0, i + j * 3 + 1, wys);
        }

        int k = 1;
        i = 0;
        int l = 0;
        for(int i1 = vec.size(); i1 > 0; i1--)
            if(vec.size() - startDisp >= i1 && l < ilosc){
                l++;
                Wpis wpis = (Wpis)vec.elementAt(i1 - 1);
                cal.setTime(wpis.getDate());
                String s = cal.get(1) + "/" + (cal.get(2) + 1) + "/" +
                    cal.get(5) + " " + cal.get(10) + ":" + cal.get(12) + " " +
                   (cal.get(9) != 0 ? "PM" : "AM");
                g.drawString(s, max[0] / 2, fontHeight * k, 17);
                s = "" + wpis.getWartosc();
                g.drawString(s, max[0] + 3 + max[1] / 2, fontHeight * k, 17);
                int j1 = wpis.getWartosc();
                if(j1 > 150)
                    s = "^";
                else
                    if(j1 < 120)
                        s = "v";
                    else
                        s = "o";
                g.drawString(s, max[0] + 3 + max[1] + 3 + max[2] / 2,
                    fontHeight * k++, 17
                );
            }

        g.translate(-xTrans, 0);
        g.drawLine(0, fontHeight, sze, fontHeight);
        if(hScroll){
            g.setColor(255, 255, 255);
            g.fillRect(0, wys, sze, 5);
            g.setColor(0, 0, 0);
            g.drawLine(0, wys, sze, wys);
            if(xTrans != 0)
                g.fillRect(3, wys + 2, 3, 3);
            if(max[0] + max[1] + max[2] + 6 > sze + xTrans * -1)
                g.fillRect(sze - 4, wys + 2, 3, 3);
        }
        if(vScroll){
```

```
273             g.setColor(255, 255, 255);
274             g.fillRect(sze - 5, 0, sze, wys);
275             g.setColor(0, 0, 0);
276             g.drawLine(sze - 5, fontHeight, sze - 5, wys);
277             if(startDisp != 0)
278                 g.fillRect(sze - 3, fontHeight + 4, 3, 3);
279             if(vec.size() > ilosc + startDisp)
280                 g.fillRect(sze - 3, wys - 4, 3, 3);
281         }
282     }
283 
284     protected void keyPressed(int i){
285         switch(getGameAction(i)){
286             case 1:
287                 if(startDisp > 0)
288                     startDisp--;
289                 break;
290             case 2:
291                 if(xTrans < 0)
292                     xTrans += 20;
293                 break;
294             case 5:
295                 if(max[0] + max[1] + max[2] + 6 > sze + xTrans * -1)
296                     xTrans -= 20;
297                 break;
298             case 6:
299                 if(startDisp + ilosc < vec.size())
300                     startDisp++;
301                 break;
302             default:
303                 break;
304 
305         }
306         repaint();
307     }
308 
309     public void commandAction(Command command, Displayable displayable){
310         if(displayable == this && command == backCommand)
311             parent.parent.display.setCurrent(parent);
312     }
313 
314     int wys;
315     int sze;
316     Command backCommand;
317     MyList parent;
318     Vector vec;
319     RecordStore rs;
320     private Font font;
321     String nazwy[] = {
322         "Data", "Stan", "Ocena"
323     };
324     int max[];
325     boolean vScroll;
326     boolean hScroll;
327     Calendar cal;
328     int startDisp;
329     int xTrans;
330     int fontHeight;
331     int ilosc;
332 }
333 ********************************
334 *    MyText.java
335 ********************************
336 package mtPack;
337 
338 import java.io.PrintStream;
339 import javax.microedition.lcdui.*;
340 import javax.microedition.rms.RecordStore;
341 import javax.microedition.rms.RecordStoreException;
342 
343 class MyText
344     extends Form
345     implements CommandListener{
346 
347     private Command enterCommand;
348     private Command backCommand;
```

```
349     private MyList parent;
350
351     private TextField tf;
352
353     public MyText(MyList mylist){
354         super("Wprowadzanie danych");
355         tf = new TextField("Stan: ", "", 3, TextField.NUMERIC);
356         append( tf);
357         enterCommand = new Command("Dodaj", 1, 1);
358         backCommand = new Command("Powrot", 2, 1);
359         addCommand(enterCommand);
360         addCommand(backCommand);
361         setCommandListener(this);
362         parent = mylist;
363     }
364
365     public void commandAction(Command command, Displayable displayable){
366         if(displayable == this){
367             if(command == enterCommand){
368                 Object obj = null;
369                 try{
370                     RecordStore recordstore = RecordStore.openRecordStore(
371                         Names.dbName, true
372                     );
373                     Wpis wpis = new Wpis(Integer.parseInt(tf.getString()));
374                     byte abyte0[] = wpis.getBytes();
375                     recordstore.addRecord(abyte0, 0, abyte0.length);
376                 }catch(RecordStoreException recordstoreexception){
377                     System.out.println("Allert" + recordstoreexception);
378                 }
379             }
380             parent.parent.display.setCurrent(parent);
381         }
382     }
383 }
384 ********************************
385 *    Names.java
386 ********************************
387 package mtPack;
388
389
390 class Names{
391     public static String dbName = "Medical";
392 }
393 ********************************
394 *
395 *    Wpis.java
396 *
397 ********************************
398 package mtPack;
399
400 import java.util.Date;
401
402 class Wpis{
403
404     private Date data;
405     private int wartosc;
406
407     public Wpis(int i){
408         wartosc = i;
409         data = new Date(System.currentTimeMillis());
410     }
411
412     private Wpis(Date date, int i){
413         wartosc = i;
414         data = date;
415     }
416
417     public int getWartosc(){
418         return wartosc;
419     }
420
421     public Date getDate(){
422         return data;
423     }
424
```

```
425     public byte[] getBytes(){
426         StringBuffer stringbuffer = new StringBuffer();
427         stringbuffer.append((new Long(data.getTime())).toString());
428         stringbuffer.append(" ");
429         stringbuffer.append((new Integer(wartosc)).toString());
430         return stringbuffer.toString().getBytes();
431     }
432 
433     public static Wpis setValues(byte abyte0[]){
434         String s = new String(abyte0);
435         Date date = new Date(
436             Long.parseLong(
437                 s.substring(0, s.indexOf(' '))
438             )
439         );
440         int i = Integer.parseInt(
441             s.substring( s.indexOf(' ') + 1, s.length())
442         );
443         return new Wpis(date, i);
444     }
445 }
446 ********************************
447 *    MyCommunication.java
448 ********************************
449 package mtPack;
450 
451 import java.io.PrintStream;
452 import java.util.Calendar;
453 import java.util.Vector;
454 import javax.microedition.lcdui.*;
455 import javax.microedition.rms.RecordStore;
456 import javax.microedition.rms.RecordStoreException;
457 
458 import javax.microedition.io.*;
459 import java.io.*;
460 
461 
462 class MyCommunication
463     extends Form
464     implements CommandListener{
465 
466     Command enterCommand;
467     Command backCommand;
468     MyList parent;
469 
470     private TextField ip;
471     private TextField st;
472 
473     public MyCommunication(MyList parent){
474         super("Wysylanie danych:");
475         this.parent = parent;
476 
477         ip = new TextField( "IP: ", "192.168.0.72", 15, TextField.ANY);
478         st = new TextField( "Communication: ", "", 300, TextField.ANY);
479 
480         append(ip);
481         append(st);
482 
483         backCommand = new Command("Powrot", 2, 1);
484 
485         enterCommand = new Command("Wyslij", 1, 1);
486 
487         addCommand(enterCommand);
488         addCommand(backCommand);
489         setCommandListener(this);
490 
491     }
492 
493     public void commandAction(Command command, Displayable displayable){
494         if(displayable == this)
495             if(command == backCommand){
496                 System.out.println("TU");
497                 parent.parent.display.setCurrent(parent);
498             }else{
499                 StreamConnection sc = null;
500                 RecordStore rs = null;
```

```
501
502                 try{
503                     st.setString("Connecting...");
504                     sc = (StreamConnection)Connector.open(
505                         "http://"+ip.getString()+":1977"
506                     );
507                     st.setString("Connection establish.\nRecordStore opening...")↩
                            ;
508                     rs = RecordStore.openRecordStore(Names.dbName, false);
509                     st.setString("Connection establish.\nRecordStore opened.");
510                     OutputStream os = sc.openOutputStream();
511                     int numer = rs.getNumRecords();
512                     for(int i = 1; i <= numer; i++){
513                         st.setString("Sending "+i+" of "+numer);
514                         os.write( rs.getRecord(i));
515                         os.flush();
516                     }
517                     st.setString("Send."+numer);
518                     os.close();
519                 }catch(Exception ex){
520                     st.setString("Exception\n"+ex);
521                 }
522             }
523     }
524 }
```

Listing 5.7: *Implementacja programu MIDMedic.*

5.2 File Connection API *JSR-75*

W miarę upływu czasu *RMS* przestał być rozwiązaniem wystarczającym. Wiele nowoczesnych urządzeń, takich jak aparaty fotograficzne czy telefony, wprowadziło różnego rodzaju hierarchiczne systemy plików. Dane przechowywane są w plikach które stanowią część katalogów. Katalogi mogą być zawarte w innych katalogach lub w katalogu głównym *root*.

Opcjonalny pakiet *JSR-75* opisuje zbiór klas pozwalających na dostęp do systemu plikowego urządzenia. W praktyce odczyt i zapis plików wykorzystuje strumienie z klas pakietu `java.io`, pozwala jednak odnieść się do nośników takich jak karta pamięci.

Należy zaznaczyć że każda aplikacja korzystająca z tego *API* musi zostać podpisana cyfrowo, operacja ta wymaga dodatkowych zasobów czasowych i finansowych. [26]

5.2.1 Czy urządzenie obsługuje *FileConnection API*?

Ponieważ pakiet *FileConnection* jest z założenia dodatkiem, warto sprawdzić czy dane urządzenie obsługuje to *API*. Operacja ta sprowadzi się do odpytania systemu, jakie wersje pakietu obsługuje.

```
String ver = System.getProperty("microedition.io.file.FileConnection.version" )
```

Pakiet nie jest obsługiwany jeżeli zostanie zwrócona wartość `null`, w przeciwnym wypadku dostarczony zostanie ciąg znaków reprezentujący wersję pakietu np. '1.0'.

5.2.2 Odwołanie do pliku w oparciu o *GCF*

Jak już zaznaczono, operacje zapisu i odczytu są realizowane przez strumienie z pakietu `java.io`. Aby uzyskać dowiązanie do pliku wykorzystana zostanie metoda `open` z klasy `Conector`. Jako parametru oczekuje ona dostarczenia zmienej typu

Rysunek 5.2: *Ekran rozpoczynający*

`String` zapisanej zgodnie z formatem *URL* (RFC 2396). Ogólny zapis to

```
{scheme}:[{target}][{params}]
```

gdzie:

- `{scheme}` jest opisem ptotokołu np. `http` czy `file`,
- `{target}` jest lokalizacją np. adresem siecowym, `CFCard/` lub `MemoryStick/`,
- `{params}` jest przypisaniem wartości do parametrów np. `type=a`.

Zatem przykładowy opis lokalizacji pliku może mieć postać:

```
String url = "file:///root1/photos/zdjecie1.png";
```

Utworzenie dowiązania wymaga konwersji zgodnej z:

```
FileConnection fc = (FileConnection)Connector.open(url);
```

Dysponując obiektem klasy `FileConnection` można uzyskać strumień wejścia/wyjścia przez wywołanie metody `openInputStream` lub `openOutputStream`.

5.2.3 Praktyczne wykorzystanie - 'Mobilny rozkład jazdy komunikacji miejskiej'

Pakiet *JSR-75* znalazł praktyczne zastosowanie w klasach klienckich aplikacji 'Timetable'[16]. Zadaniem aplikacji była dystrybucja rozkładu jazdy komunikacji miejskiej na urządzenia mobilne, umożliwiające użytkownikowi odczytanie informacji o planowanym przyjeździe autobusu czy tramwaju. Moduł odpowiedzialny za zapisanie informacji do plików zrealizowano w oparciu o pakiet `FileConnection`.

```
public
    class Client
    implements Runnable {
    /** Ścieżka do danych aplikacji */
    private static String FILE_LINE;
    /** Obiekt typu wektor przechowuje linie do aktualizacji */
    private static Vector linesToUpdate;
    /** Obiekt typu wektor przechowuje linie do usunięcia */
    private static Vector linesToDelete;
    /** Obiekt reprezentujący wyświetlacz urządzenia */
    private Display display;
    /** Obiekt reprezentujący wątek */
    private Thread timer;
    /** Obiekt reprezentujący klasę NewList */
    private NewList myList;
    /** Obiekt reprezentujący klasę BTServer */
```

```java
    private BTServer myServer;

    /**
     * Konstruktor klasy Client przypisuje wartości
     * @param myDisplay
     * @param path
     * @param myList
     * @param myServer
     */
    public Client(Display myDisplay, String path, NewList myList,BTServer ↩
        myServer) {

        this.display = myDisplay;
        Client.FILE_LINE = path;
        this.myList = myList;
        this.myServer = myServer;
    }

    /**
     * Funkcja przeprowadza aktualizacje danych z serwera tworząc strumień ↩
         połączenia.
     * Wykorzystana jest przy update przez TCP/IP.
     */
    public void makeUpdate() {

        Image image = null;
        try {
            image = Image.createImage("/info.png");
        } catch (Exception e) {
            display.setCurrent(new Alert("Informacja", "Błąd podczas ładwoania ↩
                obrazka!", null, AlertType.WARNING));
        }

        try {

            StreamConnection connection = (StreamConnection) Connector.open("↩
                socket://localhost:4445", Connector.READ_WRITE);
            makeUpdate(connection, readVersion());

        } catch (IOException e) {
            display.setCurrent(new Alert("Informacja", "Serwer nie odpowiada!", ↩
                image, AlertType.ERROR));
        }
    }

    /**
     * Przeprowadza aktualizacje danych z serwerem przez podany stumień.
     * Stworzona na potrzeby Bluetooth
     * @param myDisplay zmienna typu Display za pomcą niej wyświetlane są ↩
         komunikaty.
     * @param myStreamConnection zmienna typu StreamConnection strumień ↩
         połączenia
     * @param path zmienna typu String przechowuje scieżke plików z danymi
     */
    public void makeUpdate(StreamConnection myStreamConnection,int version) {

        String dataIn = null;
        linesToUpdate = new Vector();
        linesToDelete = new Vector();
        try {
            Image image = Image.createImage("/info.png");
            myList.setShowUpdateIco(true);
            DataOutputStream os = myStreamConnection.openDataOutputStream();
            DataInputStream is = myStreamConnection.openDataInputStream();
            PrintStream print = new PrintStream(os);
            InputStreamReader in = new InputStreamReader(is);
            myStreamConnection.close();
            try {

                print.println("Hellow");
                dataIn = Read.readLine(in);
                print.println(version);
                int versionOnServer = Integer.parseInt(dataIn);
                if (versionOnServer > version) {

                    print.println("getData");
```

```
                    //Update files
                    String countAddFile = Read.readLine(in);
                    for (int i = 0; i < Integer.parseInt(countAddFile); i++) {
                        String fileName = Read.readLine(in);
                        dataIn = Read.readLine(in);
                        writeFile(fileName + ".tmp", is, Integer.parseInt(dataIn)↩
                            );
                        linesToUpdate.addElement(fileName);
                    }
                    //Delete files
                    String countDeleteFile = Read.readLine(in);

                    for (int i = 0; i < Integer.parseInt(countDeleteFile); i++) {
                        dataIn = Read.readLine(in);
                        linesToDelete.addElement(dataIn);
                    }
                    print.println("Close");
                    updateFiles(linesToUpdate);
                    deleteFiles(linesToDelete);

                    myList.setShowUpdateIco(false);
                    this.display.setCurrent(new Alert("Informacja", "Aktualizaja ↩
                        przebiegła pomyślnie, zmieniono: " + countAddFile + " ↩
                        usunieto: " + countDeleteFile, Image.createImage("/info.↩
                        png"), AlertType.INFO));
                    this.display.vibrate(1000);
                } else if (versionOnServer < version) {
                    myList.setShowUpdateIco(false);
                    this.display.setCurrent(new Alert("Informacja", "Posiadasz ↩
                        nowsza wersje!", image, AlertType.INFO));
                    print.println("Close");
                } else {
                    print.println("Close");
                    myList.setShowUpdateIco(false);
                    this.display.setCurrent(new Alert("Informacja", "Posiadasz ↩
                        aktualną wersje!", image, AlertType.INFO));
                }

            } finally {
                linesToUpdate.removeAllElements();
                linesToDelete.removeAllElements();
                in.close();
                print.close();
                is.close();
                os.close();
                myStreamConnection.close();
                myServer.reset();

            }
        } catch (Exception e) {
            this.display.setCurrent(new Alert("Informacja", "Problem z ↩
                aktualizacja błąd połączenia", null, AlertType.ERROR));
            cleanUpFiles(linesToUpdate);
        }
    }

    /**
     * Funkcja odbiera dane (Plik o wielkości fileSize) ze strumienia in i ↩
         wpisuje je do pliku o podanej scieżce path;
     * @param in zmienna DataInputStream, strumień z którego odbieramy dane.
     * @param fileSize zmienna int StreamConnection, wielkość pliku tworzonego w ↩
         byte.
     * @param path zmienna typu String przechowuje scieżke plików z danymi.
     * @throws IOException gdy nie można stworzyc pliku lub odczytać danych.
     */
    public void writeFile(String path, DataInputStream in, int fileSize) throws ↩
        IOException {
        byte[] data = new byte[fileSize];
        in.read(data);
        FileConnection file = (FileConnection) Connector.open(FILE_LINE + path, ↩
            Connector.READ_WRITE);
        if (!file.exists()) {
            file.create();
        }
        OutputStream out = file.openOutputStream();
        out.write(data);
```

```
        out.flush();
        out.close();
        file.setHidden(true);
        file.close();

    }

    /**
     * Usuwa pliki o nazwach znajdujących się w wektorze dodając rozszerzenie *.↩
       tmp.
     * @param v wektor w którym przekazujemy nazwe plików
     */
    private void cleanUpFiles(Vector v) {

        for (int i = 0; i < v.size(); i++) {
            deleteFile((String) v.elementAt(i) + ".tmp");
        }

    }

    /**
     * Stworzenie wątku i uruchomienie metody run()
     */
    public void start() {
        timer = new Thread(this);
        timer.start();
    }
    /**
     * Usuwa plik o podanej nazwie name
     * @param name typu String należy w niej podać nazwe pliku z rozszerzeniem.
     */
    private void deleteFile(final String name) {
        new Thread() {
            public void run() {
                super.run();
                try {
                    FileConnection file = (FileConnection) Connector.open(↩
                        FILE_LINE + name, Connector.WRITE);
                    file.delete();
                    file.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }.start();
    }

    /**
     * Usuwa pliki o nazwach znajdujących się w wektorze z rozszerzeniem *.data
     * @param v typu Vector przechowuję nazwy plików.
     */
    private void deleteFiles(Vector v) {
        for (int i = 0; i < v.size(); i++) {
            deleteFile((String) v.elementAt(i) + ".data");
        }
    }

    /**
     * Aktualizuje dane z plików tymczasowych do plików używanych przez
     * aplikacje o nazwach z wektora v, następnie kasuje tymczasowe pliki
     * @param v typu Vector przechowuje nazwy plików
     */
    private void updateFiles(Vector v) {
        try {
            for (int i = 0; i < v.size(); i++) {
                ranameFile((String) v.elementAt(i));
                deleteFile((String) v.elementAt(i) + ".tmp");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Przepisuje dane z podanego pliku o scieżce name z rozszerzeniem *.tmp do
     * pliku o tej samej nazwie lecz zmienionym
```

```
     * rozszerzeniu na *.data. (Zamienia tymczasowe pliki na własciwe używane w ↩
       aplikacji.)
     * @param name typu String - scieżka pliku.
     */
    public void ranameFile(final String name) {
        new Thread() {

            public void run() {
                try {
                    FileConnection fileR = (FileConnection) Connector.open(↩
                        FILE_LINE + (String) name + ".tmp");
                    DataInputStream in = fileR.openDataInputStream();
                    writeFile((String) name + ".data", in, (int) fileR.fileSize()↩
                        );
                    in.close();
                    fileR.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }.start();
    }

    /**
     * Funkcja czyta aktualną wersje z pliku log.data
     * @return int numer posiadana wersja.
     */
    public static int readVersion() {

        int ver = 0;
        String[] tmp = null;
        String[] tmp2 = null;
        try {
            tmp = Read.read(Read.openFile(FILE_LINE + "log.data"));
            tmp2 = Read.split(tmp[0], ';');
            ver = Integer.parseInt(tmp2[0]);
        } catch (Exception e) {
        }
        return ver;
    }
    /**
     * Uruchomienie wątku
     */
    public void run() {
        makeUpdate();
    }
}
```

Listing 5.8: *Implementacja części klienckiej aplikacji Timetable*

5.3 Zarządzanie lokalnymi bazami danych

Specyfikacja *JSR-75* opisuje jeszcze jednen pakiet – *PIM*. Część urządzeń mobilnych przechowuje informacje o otrzymanej poczcie, zapisanych kontaktach czy listach zadań do wykonania. Informacje te mogą rzucić światło na codzienne życie użytkownika, a co za tym idzie – aplikacja mogłaby zasugerować np. iż warto się wybrać do fryzjera gdyż od ostatniej wizyty upłynęło już dużo czasu. Również sieci społecznościowe znajdą odpowiednie zastosowanie dla takich danych. Dlatego dostarczono programistom pakiet pozwalający na zarządzanie informacjami osobistymi (*Personal Information Managment*). Klasy zostały zgrupowane pakiecie `jaax.microedition.pim`.[19, 26, 32]

Podstawą operacji w pakiecie jest umiejętność zarządzania listami danych *PIM*. Przewidziano trzy rodzaje danych do których można się odwołać jak do list:

- kontakty – informacje o adresach, numerach telefonów;
- zdarzenia – nadchodzące spotkania, przypomnienia czy ważne daty;

- zadania do wykonania.

Urządzenie realizujące ten pakiet może nie implementować dostępu do wszystkich trzech typów danych. Wybór obsługiwanej funkcjonalności jest zależny od producenta.

5.3.1 Pozyskanie informacji

Aby uzyskać dostęp do danych, w pakiecie zaimplementowano klasę `PIM`, która działa zgodnie z wzorcem *singleton*.

```
PIM pim = PIM.getInstance();
```

Wywołanie statycznej metody `getInstance` dostarczy obiekt klasy `PIM`. Korzystając z metody `openPIMList` można pozyskać jeden z obiektów implementujących `PIMList`, czyli `ContactList`, `EventList` lub `ToDoList`. Pierwszy argument metody `openPIMList` określa jakiego typu listę programista chce pozyskać. Adekwatnie do przedstawionych już rezultatów można żądać dostarczenia:

- `PIM.CONTACT_LIST`,
- `PIM.EVENT_LIST`,
- `PIM.TODO_LIST`.

Drugi argument określa prawa na których zostanie ustanowiony dostęp do list. Przewidziano trzy rodzaje dostępu:

- `PIM.READ_ONLY` – tylko odczyt,
- `PIM.WRITE_ONLY` – tylko zapis,
- `PIM.READ_WRITE` – odczyt i zapis.

Pozyskanie listy można więc przedstawić jako

```
    EventList el = (EventList) pim.openPIMList(
        PIM.EVENT_LIST,
        PIM.READ_ONLY
    );
```

Należy zwrócić uwagę na fakt, że podczas wywołania tej metody może pojawić się wyjątek `SecurityException`. Środowisko uruchomieniowe *J2ME* może nałożyć ograniczenia w dostępie danych wykorzystując model bezpieczeństwa producenta urządzenia.

Jeżeli jednak stosowne pozwolenia zostaną przyznane, można przystąpić do odczytu poszczególnych zdarzeń:

```
    Enumeration enum = el.items();
    while( enum.hasMoreElements()){
        Event evt = (Event)enum.nextElement();
    }
```

5.3.2 Utworzenie nowego zapisu

Dane pozyskane w poprzednim rozdziale mogą już byc swobodnie przetwarzane. Po odpowiedniej obróbce serwis typu facebook dostarczy informacji o wspólnych kolegach, z imprezy której nie bardzo pamiętamy. Dane te należy jednak gdzieś przychować, i tu pojwaia się możliwość wykorzystania *PIM*.

Operacja zapisu będzie składała się z trzech kroków:

- utworzenia nowego składnika,
- uzupełnienia pól,
- i zaaplikowania zmian.

Pierwszy z wymienionych kroków jest relatywnie prosty, gdyż sprowadza się do wywołania metody `create` w wersji odpowiedniej dla typu tworzonej informacji. Dla zdarzeń będzie to:

```
PIM pim = PIM.getInstance();
EventList el = (EventList) pim.openPIMList(
    PIM.EVENT_LIST,
    PIM.READ_WRITE
);

Event evt = el.createEvent();
```

Krok drugi będzie nieco bardziej skomplikowany. Każdy z typów informacji, czyli `Contact`, `Event` i `ToDo`, zawiera własny zestaw pól. Kontakt może być opisany przez 18 różnych pól, np: `NAME`, `EMAIL`, `BIRTHDAY`, `PHOTO`. Ponadto z każdym polem jest związany typ danych który będzie go opisywał, np: `PIMItem.STRING_ARRAY`, `PIMItem.DATE` czy `PIMItem.INT`. W przypadku wydarzenia, pól będzie mniej a całą operację można przedstawić jako:

```
evt.addString(Event.SUMMARY, PIMItem.ATTR_NONE, "Spotkanie z Olą");
evt.addDate(Event.START, PIMItem.ATTR_NONE, date.getTime());
```

Jednak mogą wystąpić pola których długość jest zależna od urządzenia, dzieje się tak gdy zaistnieje potrzeba wprowadzenia informacji o kontakcie. Chcąc wprowadzić pole `NAME` czy `ADDR` konieczne jest utworzenie tablic, które będą miały pewną długość. Dlatego proces ten przebiegać będzie następująco:

```
ContactList contacts = //...
Contact contact = contacts.createContact();

String[] name = new String[contacts.stringArraySize(Contact.NAME)];
name[Contact.NAME_FAMILY] = "Public";
name[Contact.NAME_GIVEN] = "John";

contact.addStringArray(Contact.NAME, PIMItem.ATTR_NONE, name);
```

Aplikacja informacji, czyli ostatni i najprostszy krok, sprowadza się do wywołania metody `commit`:

```
evt.commit();
```

6 Obraz i dźwięk

W rozdziale 3 pokazano już jak można odczytać i umieścić obrazy w aplikacjach profilu *MIDP*. `JPRG`, `PNG` i `SVG` stanowią jednak tylko prezentacje formatów graficznych, a niezdefiniowane pozostają formaty audio i video.

Profil *MIDP 1.0* praktycznie nie uwzględniał faktu istnienia dzwięków czy treści wideo. Sytuacja diametralnie zmieniła się wraz z pojawieniem *MIDP 2.0*. Ponadto udostępniono dodatkowy pakiet *Mobile Media API* (*MMAPI*), który rozszerzał funkcjonalność oferowanych interfejsów programistycznych [20].

6.1 Co wprowadziło *MIDP*

Począwszy od pierwszych komputerów klasy *PC*, pojedyncze piknięcie głośnika oznajmiało poprawne załadowanie biosu. Przenosząc ten sam mechanizm na urządzenia mobilne, można powiedzieć że od zawsze wciśnięcie klawisza skutkowało wywołaniem dzwięku. Niestety programując urządzenie mobilne w języku *Java* twórcy nie przewidzieli możliwości odegrania choćby prostych dzwięków. Błąd ten poprawiono przy pierwszej nadarzającej się okazji.

```
Manager.playTone(60, 500, 100);
```

Metoda `playTone` ma za zadanie odegrać dźwięk określony trzema argumentami:

- rodzaj dźwięku jaki należy odtworzyć,
- czas odtwarzania,
- głośność.

Oczywiście możliwość odegrania całej melodii dawała znacznie większe możliwości urozmaicenia gier czy programów. Dlatego w pakiecie `javax.microedition.media.control` umieszczono interfejs `ToneControl`, a dokumentacja pokazuje jak należy z niego korzystać.

```
21      byte tempo = 30; // set tempo to 120 bpm
22      byte d = 8;      // eighth-note
23
24      byte C4 = ToneControl.C4;;
25      byte D4 = (byte)(C4 + 2); // a whole step
26      byte E4 = (byte)(C4 + 4); // a major third
27      byte G4 = (byte)(C4 + 7); // a fifth
28      byte rest = ToneControl.SILENCE; // rest
29
30      byte[] mySequence = {
31          ToneControl.VERSION, 1,   // version 1
32          ToneControl.TEMPO, tempo, // set tempo
33          ToneControl.BLOCK_START, 0,   // start define "A" section
34          E4,d, D4,d, C4,d, E4,d,       // content of "A" section
35          E4,d, E4,d, E4,d, rest,d,
36          ToneControl.BLOCK_END, 0,     // end define "A" section
```

```
37          ToneControl.PLAY_BLOCK, 0,     // play "A" section
38          D4,d, D4,d, D4,d, rest,d,      // play "B" section
39          E4,d, G4,d, G4,d, rest,d,
40          ToneControl.PLAY_BLOCK, 0,     // repeat "A" section
41          D4,d, D4,d, E4,d, D4,d, C4,d   // play "C" section
42      };
43
44      try{
45          Player p = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR);
46          p.realize();
47          ToneControl c = (ToneControl)p.getControl("ToneControl");
48          c.setSequence(mySequence);
49          p.start();
50      } catch (IOException ioe) {
51      } catch (MediaException me) { }
```

Listing 6.1: *Użycie* `ToneControl`

Łatwo można wyróżnić tablicę `mySequence` z poszczególnymi dzwiękami, które należy odegrać. Widać również że po utworzeniu obiektu `Player` konwertujemy go na `ToneControl`, aby zapisać w nim przygotowaną sekwencję. Następnie obiekt klasy `Player` przechodzi ze stanu `realize` do stanu `start`, co powoduje odtworzenie muzyki.

Jednak era odgrywania dzwięków powoli przemijała i zwykłe bipery były zastępowane głośnikami polifonicznymi. Dźwięk jaki można było na nich odtworzyć miał znacznie lepszą jakość więc i formaty obsługujące go musiały być doskonalsze. Urządzenia z głośnikami polifonicznymi mogły odtwarzać następujące formaty audio:

```
52      InputStream in = getClass().getResourceAsStream("/signs_m.wav");
53      Player p = Manager.createPlayer(in, "audio/x-wav");
54      p.start();
```

Listing 6.2: *Odtwarzanie strumienia audio.*

Pliki audio dostarczone w pakietach *MIDlet*owych doskonale nadawały się jako tło w licznych grach. Jak widać ich użycie było bardzo proste, wystarczyło dostarczyć strumień i format, a odtwarzaniem zajmował się już player.

6.2 MMAPI

Specyfikację pakietu *MMAPI* tworzono z myślą o dopasowaniu do jak największej ilości multimediów. Założenia obejmowały obsługę podstawowych funkcji audio w urządzeniach o ograniczonych zasobach i jednocześnie możliwość obsługi zaawansowanych multimediów w urządzeniach pozbawionych ograniczeń.

Dane mogą być dostarczane z różnych źródeł: począwszy od pakietu `jar` przez system plików, strumienie sieciowe aż po urządzenia akwizycyjne takie jak kamery i mikrofony. Koncepcja pakietu zakładała możliwość obsługi wszystkich możliwych lokalizacji i protokołów bez konieczności utraty elastyczności .

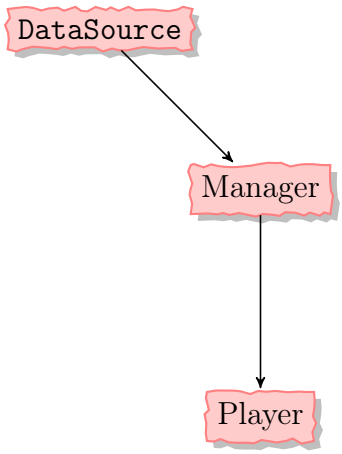
Pozyskane dane powinny zostać przetworzone a w następnym etapie wyrysowane na urządzeniu. *MMAPI* jest pakietem dla programowania wysokopoziomowego, dlatego nie musi dostarczać implementacji tych procesów. Wymaga jednak dostarczenia stosownej obsługi obejmującej nie tylko interpretację strumieni danych, ale również ich dekodowanie i sprawowanie kontroli.[21]

Aby spełnić te założenia wprowadzono dwa interfejsy, `DataSource` i `Player`. Celem pierwszego jest zapewnienie dostępu do plików multimedialnych, natomiast drugi jest odpowiedzialny za szeroko rozumianą obsługę odtwarzania.

Zadanie interfejsu `DataSource` jest realizowane za pomocą klas strumieniowych z pakietu `javax.microedition.media.protocol`. Pojedynczy `SourceStream` jest

przypisany do jednego typu danych, więc połączenie części audio i wideo wymaga połączenia strumieni. Praktyka pokazuje jednak że tak długo jak nie istnieje konieczność zdefinowania własnego protokołu, można operować na plikach multimedialnych nie odwołując się bezpośrednio do `DataSource`.

Pomiędzy `DataSource` a `Player`em istnieje jeszcze jedna, bardzo ważna klasa – `Manager`. Jej zadaniem jest dostarczyć programiście obiekt klasy `Player`. Wykorzystanie tej klasy przedstawiono w przykładach 6.1 i 6.2.



Rysunek 6.1: *Klasa `Manager`.*

6.3 Praktyczne wykorzystanie MMAPI

Celem przedstawionej tu pracy było zademonstrowanie złożoności procesu tworzenia zaawansowanej aplikacji mobilnej oraz jego wyjaśnienie w oparciu o dostępną wiedze teoretyczną. Autorzy pragnęli pokazać jakim wyzwaniem jest stworzenie złożonej aplikacji mobilnej przy ograniczonych zasobach ludzkich (2 osoby) oraz ograniczonym czasie (3 miesiące)[29].

Ekran zapisu zdjęcia Ekran zapisu dźwięku Ekran nawigacji

Rysunek 6.2: *Operacje przesunięcia.*

6.3.1 Założenia wstępne dotyczące aplikacji

Pierwszym etapem tworzenia projektu było określenie założeń wstępnych dotyczących samej aplikacji i jej funkcjonalności. Pierwotnym założeniem było zrealizowanie jedynie części mobilnej projektu czyli aplikacji Java dla urządzenia mobilnego. Umożliwiać ona miała śledzenie przebytej przez użytkownika trasy oraz zbieranie fotografii zgromadzonych w jej trakcie. Część serwerowa miała być potraktowana jedynie teoretycznie, jako wykraczająca poza podstawowy projekt. Pomyślny rozwój projektu oraz trafność stosowanych rozwiązań pozwoliła Autorom na implementację całości systemu, mieszcząc się w wyznaczonych ramach czasowych.

W trakcie implementacji Autorzy postanowili wzbogacić aplikację o szereg funkcji. Wśród nich znalazła się obsługa rejestrowania plików muzycznych i wideo. Implementacja ustawień aplikacji zapamiętywanych w pamięci telefonu. Implementacja animowanego interfejsu, czy systemu przeglądania plików multimedialnych.

6.3.2 Wybrane rozwiązania

```
1  package view;
2
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5  import javax.microedition.lcdui.game.Sprite;
6  import control.Controller;
7  import control.HttpController;
8  import control.RmsController;
9  import control.FController;
10 import java.io.*;
11 import javax.microedition.media.*;
12 import javax.microedition.media.control.*;
13
14 public class AudioCanvas extends Canvas implements Runnable, CommandListener, ↩
       PlayerListener {
15
16     private Controller controller;
17     private Font displayFont = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, ↩
           Font.SIZE_LARGE);
18     private final int WIDTH, HEIGHT;
19     private boolean active = true;
20     private boolean initialized, recording, recorded, playingAudio;
21     private Image screen, recButton, leftButton, rightButton, diode, microphone;
22     private Sprite spriteRec, spriteLeft, spriteRight, spriteDiode, spriteMicro;
23     private Command backCommand;
24     private Player player, playback;
25     private RecordControl recordControl;
26     private ByteArrayOutputStream output;
27     private byte[] recordedSoundArray;
28     private FController fileController;
29     private String name;
30
31     public AudioCanvas(Controller c) {
32         controller = c;
33
34         setFullScreenMode(true);
35         setCommandListener(this);
36
37         try{
38             screen = Image.createImage("/images/PhotoScreen.png");
39             recButton = Image.createImage("/images/RecButton.png");
40             leftButton = Image.createImage("/images/LeftButton.png");
41             rightButton = Image.createImage("/images/RightButton.png");
42             diode = Image.createImage("/images/RecDiode.png");
43             microphone = Image.createImage("/images/microphone.png");
44         }catch(Exception e){}
45
46         spriteRec = new Sprite(recButton, 60, 80);
47         spriteRec.setPosition(90,190);
48         spriteRec.setFrame(1);
49
50         spriteLeft = new Sprite(leftButton, 50, 60);
51         spriteLeft.setPosition(17,190);
52         spriteLeft.setFrame(0);
53
54         spriteRight = new Sprite(rightButton, 50, 60);
55         spriteRight.setPosition(173,190);
56         spriteRight.setFrame(0);
57
58         spriteDiode = new Sprite(diode, 20, 20);
59         spriteDiode.setPosition(188,32);
60         spriteDiode.setFrame(0);
61
```

```
62          spriteMicro = new Sprite(microphone, 176, 144);
63          spriteMicro.setPosition(32,32);
64          spriteMicro.setFrame(0);
65
66          WIDTH = getWidth();
67          HEIGHT = getHeight();
68
69          backCommand = new Command("Back", Command.BACK, 4);
70          addCommand(backCommand);
71
72          initializeRecorder();
73
74          Thread t = new Thread(this);
75          t.setPriority(Thread.MAX_PRIORITY);
76          t.start();
77      }
78
79      public void paint(Graphics g){
80          if(!initialized){
81              //controller.gpsCanvas.getMap().paint(g);
82              g.drawImage(screen,0,0,Graphics.LEFT | Graphics.TOP);
83              spriteMicro.paint(g);
84              spriteRec.paint(g);
85              initialized = true;
86          }
87          if(recording){
88              spriteMicro.paint(g);
89              spriteRec.paint(g);
90              spriteDiode.paint(g);
91          }
92          if(recorded){
93              spriteDiode.paint(g);
94              spriteRec.paint(g);
95              spriteLeft.paint(g);
96              spriteRight.paint(g);
97          }
98          if(playingAudio){
99              spriteMicro.paint(g);
100         }
101     }
102
103     public void initializeRecorder(){
104
105         try {
106             player = Manager.createPlayer("capture://audio?encoding=audio/wav");
107             player.realize();
108             recordControl = (RecordControl)player.getControl("RecordControl");
109             output = new ByteArrayOutputStream();
110             recordControl.setRecordStream(output);
111         }
112         catch (Exception e) { }
113     }
114
115     public void initializePlayback(){
116         ByteArrayInputStream recordedInputStream = new ByteArrayInputStream(↩
                recordedSoundArray);
117         try {
118             playback = Manager.createPlayer(recordedInputStream,"audio/wav");
119             playback.prefetch();
120             playback.addPlayerListener(this);
121         }
122         catch (Exception e) { }
123     }
124
125     public void commandAction(Command command, Displayable displayable) {
126         if(command == backCommand) {
127             shutdown();
128         }
129     }
130
131     public void shutdown(){
132         if(name!=null){
133             HttpController c = new HttpController();
134             RmsController rms = new RmsController(controller.travelName);
135             c.sendWaypoints(rms.getWaypoints());
136             rms.clearWaypoints();
```

```
137             c.sendMedia(name+".wav",recordedSoundArray);
138         }
139         try{
140             player.close();
141             playback.close();
142             recordControl.stopRecord();
143         }catch (Exception e){};
144         active = false;
145         controller.resumeTravel();
146     }
147 
148     public void keyPressed(int keyCode) {
149         switch(keyCode){
150             case -5:
151                 if(recorded){
152                     fileController = new FController(controller.travelName);
153                     name = controller.travelName +
154                             "_" + controller.gpsCanvas.createMediaPoint(2);
155                     fileController.saveFile
156                             (name ,recordedSoundArray, "wav");
157                     shutdown();
158                     break;
159                 }
160                 if(!recording){
161                     removeCommand(backCommand);
162                     spriteRec.setFrame(3);
163                     try{
164                         recordControl.startRecord();
165                         player.start();
166                     } catch (Exception e) { }
167                     recording = true;
168                     break;
169                 }
170                 else{
171                     try{
172                         recordControl.commit();
173                         recordedSoundArray = output.toByteArray();
174                         player.close();
175                     } catch (Exception e) { }
176                     spriteDiode.setFrame(2);
177                     spriteRec.setFrame(2);
178                     spriteLeft.setFrame(2);
179                     spriteRight.setFrame(1);
180                     recording = false;
181                     initializePlayback();
182                     recorded = true;
183                     break;
184                 }
185             case -6:
186                 if(recorded){
187                     if(!playingAudio){
188                         try{
189                             playback.start();
190                         } catch(Exception e) {}
191                         spriteLeft.setFrame(1);
192                         repaint();
193                         playingAudio = true;
194                     }else{
195                         try{
196                             playback.stop();
197                         } catch(Exception e) {}
198                         spriteLeft.setFrame(2);
199                         repaint();
200                         playingAudio = false;
201                     }
202                 }
203                 break;
204             case -7:
205                 active = false;
206                 shutdown();
207                 break;
208         }
209     }
210 
211     public void playerUpdate(Player player, String event, Object eventData) {
212         if (event == PlayerListener.END_OF_MEDIA){
```

```
213             spriteLeft.setFrame(2);
214             repaint();
215             playingAudio=false;
216         }
217     }
218 
219     public void run(){
220         while(active){
221             repaint();
222             if(recording){
223                 if(spriteDiode.getFrame()==0)
224                     spriteDiode.setFrame(1);
225                 else
226                     spriteDiode.setFrame(0);
227             }
228             try{
229                 Thread.sleep(500);
230             }catch(Exception e){}
231         }
232     }
233 }
```

Listing 6.3: *Klasa* ***AudioCanvas***

```
1  package view;
2 
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5  import java.io.*;
6  import control.Controller;
7  import control.FController;
8  import javax.microedition.media.*;
9  import javax.microedition.media.control.*;
10 import javax.microedition.lcdui.game.Sprite;
11 
12 public class MediaCanvas extends Canvas implements Runnable, PlayerListener {
13 
14     private Controller controller;
15     private Font displayFont = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, ↩
           Font.SIZE_LARGE);
16     private Font smallFont = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font↩
           .SIZE_SMALL);
17     private final int WIDTH, HEIGHT;
18     private int possition, textPossition;
19     private boolean active = true, photo, audio, video, playing;
20     private String name, type;
21     private Image screen, picture, leftButton;
22     private FController fileController;
23     private byte[] data;
24     private Player playback;
25     private VideoControl videoControl;
26     private Sprite spriteLeft;
27     private String[] msg = {"Coordinates: ","Size: ","Date: "};
28     private String[] stats;
29 
30     public MediaCanvas(Controller c, String fileName) {
31         controller = c;
32         name = fileName;
33         type = name.substring(name.lastIndexOf('.'),name.length());
34 
35         setFullScreenMode(true);
36 
37         WIDTH = getWidth();
38         HEIGHT = getHeight();
39 
40         fileController = new FController(controller.travelName);
41         data = fileController.openFile(name);
42 
43         try{
44             this.wait(1000);
45         } catch(Exception e){}
46 
47         if(type.equals(".jpg")){
48             try{
49                 picture = makeThumb(Image.createImage(data,0,data.length));
```

```
50              }catch (Exception e){}
51              photo = true;
52          }
53          if(type.equals(".wav")){
54              possition = 190;
55              textPossition = 0;
56              initializeAudioPlayback();
57              audio = true;
58          }
59          if(type.equals(".3gpp")){
60              possition = 55;
61              textPossition = 90;
62              initializeVideoPlayback();
63              video = true;
64          }
65
66          try{
67              screen = Image.createImage("/images/btScreen.png");
68              leftButton = Image.createImage("/images/LeftButton.png");
69          }catch(Exception e){}
70
71          if(audio||video){
72              spriteLeft = new Sprite(leftButton, 50, 60);
73              spriteLeft.setPosition(19,possition);
74              spriteLeft.setFrame(2);
75          }
76
77          stats = fileController.getFileStat(name);
78          Thread t = new Thread(this);
79          t.setPriority(Thread.MAX_PRIORITY);
80          t.start();
81      }
82
83      private Image makeThumb(Image image) {
84          int width = image.getWidth();
85          int height = image.getHeight();
86
87          int tWidth = 176;
88          int tHeight = 144;
89
90          Image thumb = Image.createImage(tWidth, tHeight);
91          Graphics g = thumb.getGraphics();
92
93          for (int y = 0; y < tHeight; y++) {
94              for (int x = 0; x < tWidth; x++) {
95                  g.setClip(x, y, 1, 1);
96                  int dx = x * width / tWidth;
97                  int dy = y * height / tHeight;
98                  g.drawImage(image, x - dx, y - dy, Graphics.LEFT | Graphics.TOP);
99              }
100         }
101         Image done = Image.createImage(thumb);
102         return done;
103     }
104
105     public void initializeAudioPlayback(){
106         ByteArrayInputStream recordedInputStream = new ByteArrayInputStream(data)↩
                ;
107         try {
108             playback = Manager.createPlayer(recordedInputStream,"audio/wav");
109             playback.prefetch();
110             playback.addPlayerListener(this);
111         }
112         catch (Exception e) {controller.showAlert("1","1"); }
113     }
114
115     public void initializeVideoPlayback(){
116         ByteArrayInputStream videoInputStream = new ByteArrayInputStream(data);
117         try {
118             playback = Manager.createPlayer(videoInputStream,"video/3gpp8");
119             playback.prefetch();
120             videoControl = (VideoControl)playback.getControl("VideoControl");
121             videoControl.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, this);
122             videoControl.setDisplayLocation(32,120);
123             videoControl.setDisplaySize(176, 144);
124             videoControl.setVisible(true);
```

```
            playback.addPlayerListener(this);
        }
        catch (Exception e) {controller.showAlert("2","2");  }
    }

    public void paint(Graphics g){
        g.drawImage(screen,0,0,Graphics.LEFT | Graphics.TOP);
        g.setFont(smallFont);
        g.drawString(name, 35, 37 , Graphics.LEFT | Graphics.TOP);
        if(photo)
            try{
            g.drawImage(picture,32,120,Graphics.LEFT | Graphics.TOP);
            }catch (Exception e){controller.showAlert("3","3");  }
        if(audio||video)
            spriteLeft.paint(g);
        for(int i=0; i<stats.length; i++){
            g.drawString(msg[i] + stats[i], 35, 70 + textPossition + i*smallFont.↩
                getHeight() , Graphics.LEFT | Graphics.TOP);
        }
    }

    public void playerUpdate(Player player, String event, Object eventData) {
        if (event == PlayerListener.END_OF_MEDIA){
            spriteLeft.setFrame(2);
            playing = false;
            repaint();
        }
    }

    public void keyPressed(int key){
        switch(key){
            case -5:
            if(audio || video){
                if(!playing){
                    spriteLeft.setFrame(1);
                    try{
                        playback.start();
                    } catch(Exception e) {controller.showAlert("4","4"); }
                    playing = true;
                    repaint();
                }else{
                    spriteLeft.setFrame(2);
                    try{
                        playback.stop();
                    } catch(Exception e) {controller.showAlert("5","5"); }
                    playing = false;
                    repaint();
                }
            }
            break;
            case -6:
                try{
                    playback.close();
                }catch(Exception e){}
                controller.resumeBrowse();
            break;
        }
    }

    public void run(){
        while(active){
            if(!playing)
                repaint();
            try{
                Thread.sleep(1000);
            }catch(Exception e){}
        }
    }
}
```

Listing 6.4: *Klasa* `MediaCanvas`

```
package view;

import javax.microedition.midlet.*;
```

```
4  import javax.microedition.lcdui.*;
5  import javax.microedition.lcdui.game.Sprite;
6  import control.Controller;
7  import control.HttpController;
8  import control.RmsController;
9  import control.FController;
10 import model.Waypoint;
11 import javax.microedition.media.*;
12 import javax.microedition.media.control.*;
13
14 public class PhotoCanvas extends Canvas implements Runnable {
15
16     private Controller controller;
17     private final int WIDTH, HEIGHT;
18     private boolean active = true;
19     private boolean initialized, snapshot;
20     private Image screen, recButton, leftButton, rightButton, snap, snapThumb;
21     private Sprite spriteRec, spriteLeft, spriteRight;
22     private Player player;
23     private VideoControl videoControl;
24     private FController fileController;
25     private byte[] rawSnap;
26     private String name;
27     private Command backCommand;
28     private RmsController rms;
29
30     public PhotoCanvas(Controller c) {
31         controller = c;
32         setFullScreenMode(true);
33
34         try{
35             screen = Image.createImage("/images/PhotoScreen.png");
36             recButton = Image.createImage("/images/RecButton.png");
37             leftButton = Image.createImage("/images/LeftButton.png");
38             rightButton = Image.createImage("/images/RightButton.png");
39         }catch(Exception e){}
40
41         spriteRec = new Sprite(recButton, 60, 80);
42         spriteRec.setPosition(90,190);
43         spriteRec.setFrame(1);
44
45         spriteLeft = new Sprite(leftButton, 50, 60);
46         spriteLeft.setPosition(17,190);
47         spriteLeft.setFrame(0);
48
49         spriteRight = new Sprite(rightButton, 50, 60);
50         spriteRight.setPosition(173,190);
51         spriteRight.setFrame(0);
52
53         WIDTH = getWidth();
54         HEIGHT = getHeight();
55
56         backCommand = new Command("Back", Command.BACK, 4);
57         addCommand(backCommand);
58
59         initializeCamera();
60
61         Thread t = new Thread(this);
62         t.setPriority(Thread.MAX_PRIORITY);
63         t.start();
64     }
65
66     public void paint(Graphics g){
67
68         if(!initialized){
69             //controller.gpsCanvas.getMap().paint(g);
70             g.drawImage(screen,0,0,Graphics.LEFT | Graphics.TOP);
71             spriteRec.paint(g);
72             initialized = true;
73         }
74         if(snapshot){
75             g.drawImage(snapThumb,32,32,Graphics.LEFT | Graphics.TOP);
76             spriteRec.paint(g);
77             spriteLeft.paint(g);
78             spriteRight.paint(g);
79         }
```

```
80      }
81
82      public void initializeCamera(){
83          try {
84              player = Manager.createPlayer("capture://video");
85              player.realize();
86              videoControl = (VideoControl)player.getControl("VideoControl");
87              player.start();
88              videoControl.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, this);
89              videoControl.setDisplayLocation(32, 32);
90              videoControl.setDisplaySize(176,144);
91              videoControl.setVisible(true);
92          }
93          catch (Exception e) { }
94      }
95
96      public void snapshot(){
97          try{
98              rawSnap = videoControl.getSnapshot("encoding=jpeg&width=320&height↩
                    =240");
99              snap = Image.createImage(rawSnap, 0, rawSnap.length);
100             snapThumb = makeThumb(snap);
101             videoControl.setVisible(false);
102             snapshot = true;
103         } catch (Exception e) { }
104     }
105
106     private Image makeThumb(Image image) {
107         int width = image.getWidth();
108         int height = image.getHeight();
109
110         int tWidth = 176;
111         int tHeight = 144;
112
113         Image thumb = Image.createImage(tWidth, tHeight);
114         Graphics g = thumb.getGraphics();
115
116         for (int y = 0; y < tHeight; y++) {
117             for (int x = 0; x < tWidth; x++) {
118                 g.setClip(x, y, 1, 1);
119                 int dx = x * width / tWidth;
120                 int dy = y * height / tHeight;
121                 g.drawImage(image, x - dx, y - dy, Graphics.LEFT | Graphics.TOP);
122             }
123         }
124
125         Image done = Image.createImage(thumb);
126         return done;
127     }
128
129     public void shutdown(){
130
131         if(name!=null){
132             HttpController c = new HttpController();
133             rms = new RmsController(controller.gpsCanvas.trackName);
134             c.sendWaypoints(rms.getWaypoints());
135             rms.clearWaypoints();
136             c.sendMedia(name+".jpg",rawSnap);
137         }
138
139         try{
140             player.close();
141             videoControl.setVisible(false);
142         }catch (Exception e){controller.showAlert("title","message");}
143         active = false;
144         controller.resumeTravel();
145     }
146
147     public void keyPressed(int keyCode) {
148         switch(keyCode){
149             case -5:
150                 if(!snapshot){
151                     spriteRec.setFrame(2);
152                     repaint();
153                     removeCommand(backCommand);
154                     snapshot();
```

```
155                 }
156                 break;
157             case -6:
158                 fileController = new FController(controller.travelName);
159
160                 name = controller.travelName +
161                     "_" + controller.gpsCanvas.createMediaPoint(1);
162                 fileController.saveFile
163                     (name, rawSnap, "jpg");
164
165                 shutdown();
166                 break;
167             case -7:
168                 if(snapshot)
169                     shutdown();
170                 break;
171         }
172     }
173
174     public void run(){
175         while(active){
176             repaint();
177             try{
178                 Thread.sleep(100);
179             }catch(Exception e){}
180         }
181     }
182 }
```

Listing 6.5: *Klasa PhotoCanvas*

```
1  package view;
2
3  import javax.microedition.midlet.*;
4  import javax.microedition.lcdui.*;
5  import javax.microedition.lcdui.game.Sprite;
6  import control.Controller;
7  import control.HttpController;
8  import control.RmsController;
9  import control.FController;
10 import java.io.*;
11 import javax.microedition.media.*;
12 import javax.microedition.media.control.*;
13
14 public class VideoCanvas extends Canvas implements Runnable, CommandListener, ↩
       PlayerListener {
15
16     private Controller controller;
17     private Font displayFont = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, ↩
           Font.SIZE_LARGE);
18     private final int WIDTH, HEIGHT;
19     private boolean active = true;
20     private boolean initialized, recording, recorded, playingVideo;
21     private Image screen, recButton, leftButton, rightButton, diode;
22     private Sprite spriteRec, spriteLeft, spriteRight, spriteDiode;
23     private Command backCommand;
24     private Player player, videoPlayer;
25     private RecordControl recordControl;
26     private ByteArrayOutputStream output;
27     private byte[] recordedVideoArray;
28     private FController fileController;
29     private VideoControl videoControl, videoPControl;
30     private String name;
31
32     public VideoCanvas(Controller c) {
33         controller = c;
34
35         setFullScreenMode(true);
36         setCommandListener(this);
37
38         try{
39             screen = Image.createImage("/images/PhotoScreen.png");
40             recButton = Image.createImage("/images/RecButton.png");
41             leftButton = Image.createImage("/images/LeftButton.png");
42             rightButton = Image.createImage("/images/RightButton.png");
```

```
43                  diode = Image.createImage("/images/RecDiode.png");
44              }catch(Exception e){}
45
46              spriteRec = new Sprite(recButton, 60, 80);
47              spriteRec.setPosition(90,190);
48              spriteRec.setFrame(1);
49
50              spriteLeft = new Sprite(leftButton, 50, 60);
51              spriteLeft.setPosition(17,190);
52              spriteLeft.setFrame(0);
53
54              spriteRight = new Sprite(rightButton, 50, 60);
55              spriteRight.setPosition(173,190);
56              spriteRight.setFrame(0);
57
58              spriteDiode = new Sprite(diode, 20, 20);
59              spriteDiode.setPosition(188,32);
60              spriteDiode.setFrame(0);
61
62              WIDTH = getWidth();
63              HEIGHT = getHeight();
64
65              backCommand = new Command("Back", Command.BACK, 4);
66              addCommand(backCommand);
67
68              initializeRecorder();
69
70              Thread t = new Thread(this);
71              t.setPriority(Thread.MAX_PRIORITY);
72              t.start();
73          }
74
75          public void paint(Graphics g){
76              if(!initialized){
77                  //controller.gpsCanvas.getMap().paint(g);
78                  g.drawImage(screen,0,0,Graphics.LEFT | Graphics.TOP);
79                  spriteRec.paint(g);
80                  initialized = true;
81              }
82              if(recording){
83                  spriteRec.paint(g);
84                  spriteDiode.paint(g);
85              }
86              if(recorded){
87                  spriteRec.paint(g);
88                  spriteLeft.paint(g);
89                  spriteRight.paint(g);
90              }
91          }
92
93          public void initializeRecorder(){
94
95              try {
96                  player = Manager.createPlayer("capture://video");
97                  player.realize();
98                  videoControl = (VideoControl)player.getControl("VideoControl");
99                  player.start();
100                 videoControl.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, this);
101                 videoControl.setDisplayLocation(32,32);
102                 videoControl.setDisplaySize(176,144);
103                 videoControl.setVisible(true);
104                 recordControl = (RecordControl)player.getControl("RecordControl");
105                 output = new ByteArrayOutputStream();
106                 recordControl.setRecordStream(output);
107             }
108             catch (Exception e) { }
109         }
110
111         public void initializePlayback(){
112             ByteArrayInputStream videoInputStream = new ByteArrayInputStream(↩
                    recordedVideoArray);
113             try {
114                 videoPlayer = Manager.createPlayer(videoInputStream,"video/3gpp8");
115                 videoPlayer.prefetch();
116                 videoPControl = (VideoControl)videoPlayer.getControl("VideoControl");
117                 videoPControl.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, this);
```

```
118             videoPControl.setDisplayLocation(32,32);
119             videoPControl.setDisplaySize(176, 144);
120             videoPControl.setVisible(true);
121             videoPlayer.addPlayerListener(this);
122         }
123         catch (Exception e) { }
124     }
125 
126     public void commandAction(Command command, Displayable displayable) {
127         if(command == backCommand) {
128             shutdown();
129         }
130     }
131 
132     public void shutdown(){
133         if(name!=null){
134             HttpController c = new HttpController();
135             RmsController rms = new RmsController(controller.travelName);
136             c.sendWaypoints(rms.getWaypoints());
137             rms.clearWaypoints();
138             c.sendMedia(name+".3gpp",recordedVideoArray);
139         }
140         try{
141             player.close();
142             videoPlayer.close();
143             recordControl.stopRecord();
144             videoControl.setVisible(false);
145             videoPControl.setVisible(false);
146         }catch (Exception e){}
147         active = false;
148         controller.resumeTravel();
149     }
150 
151     public void keyPressed(int keyCode) {
152         switch(keyCode){
153             case -5:
154                 if(recorded){
155                     fileController = new FController(controller.travelName);
156                     String name = controller.travelName +
157                             "_" + controller.gpsCanvas.createMediaPoint(3);
158                     fileController.saveFile(
159                             name ,recordedVideoArray, "3gpp");
160                     shutdown();
161                     break;
162                 }
163                 if(!recording){
164                     removeCommand(backCommand);
165                     spriteRec.setFrame(3);
166                     spriteDiode.setFrame(0);
167                     try{
168                         recordControl.startRecord();
169                     } catch (Exception e) { }
170                     recording = true;
171                     break;
172                 }
173                 else{
174                     try{
175                         recordControl.commit();
176                         recordedVideoArray = output.toByteArray();
177                         videoControl.setVisible(false);
178                         player.close();
179                     } catch (Exception e) { }
180                     spriteDiode.setFrame(2);
181                     spriteRec.setFrame(2);
182                     spriteLeft.setFrame(2);
183                     spriteRight.setFrame(1);
184                     recording = false;
185                     initializePlayback();
186                     recorded = true;
187                     break;
188                 }
189             case -6:
190                 if(recorded){
191                     if(!playingVideo){
192                         try{
193                             videoPlayer.start();
```

```
194                         } catch(Exception e) {}
195                         spriteLeft.setFrame(1);
196                         repaint();
197                         playingVideo = true;
198                     }else{
199                         try{
200                             videoPlayer.stop();
201                         } catch(Exception e) {}
202                         spriteLeft.setFrame(2);
203                         repaint();
204                         playingVideo = false;
205                     }
206                 }
207                 break;
208             case -7:
209                 active = false;
210                 shutdown();
211                 break;
212         }
213     }
214 
215     public void playerUpdate(Player player, String event, Object eventData) {
216         if (event == PlayerListener.END_OF_MEDIA){
217             spriteLeft.setFrame(2);
218             repaint();
219             playingVideo=false;
220         }
221     }
222 
223     public void run(){
224         while(active){
225             if(!playingVideo) repaint();
226             try{
227                 Thread.sleep(500);
228             }catch(Exception e){}
229         }
230     }
231 }
```

Listing 6.6: *Klasa* `VideoCanvas`

7 Podstawy protokołu SMS

Najczęściej spotykaną grupą urządzeń mobilnych są telefony komórkowe. Zostały pomyślane jako urządzenia do transmisji rozmów, jednak ogromną popularność wśród użytkowników zawdzięczają po części również niewielkiej usłudze *krótkich wiadomości tekstowcyh*. Długość wysyłanych wiadomości SMS jest zależna od wybranego systemu kodowania i wynosi:

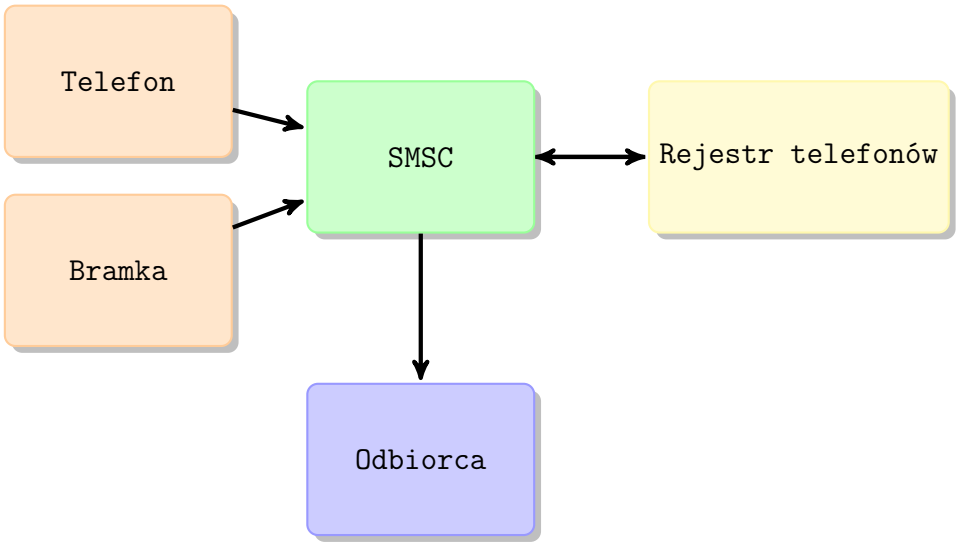
- 160 znaków – gdy na każdy znak przeznaczone jest 7 bitów,
- 140 znaków – w przypadku 8 bitów,
- 70 znaków – przy 16 bitowym kodowaniu.

Oprócz tekstu wprowadzonego przez użytkownika chce przesłać, wiadomość zawiera nagłówek opisujący sposób dostarczenia, numery odbiorcy oraz centrum odpowiedzialnego za odebranie wiadomości.

Przygotowaną wiadomość można wysłać z telefonu lub bramki internetowej. Niezależnie od źródła odbiorcą będzie centrum obsługi (*Shorm Message Service Center*), którego zadaniem będzie:

- połączenie z rejestrem aktywnych urządzeń,
- pozyskanie informacy o odbiorcy,
- a następnie dostarczenie wiadomości.

Proces ten ilustruje rysunek 7.1.



Rysunek 7.1: *Schemat przekazywania wiadomości SMS*

Wszystkie strony tego procesu będą korzystały z danych nagłówka. Centrum będzie musiało poznać numer adresata, a odbiorca będzie musiał zdekodować wiadomość. Jak jednak odczytać informacje ciągu opisanego w 7.1?

```
07917283010010F5040BC87238880900F10000993092516195800AE8329BFD4697D9EC37
```

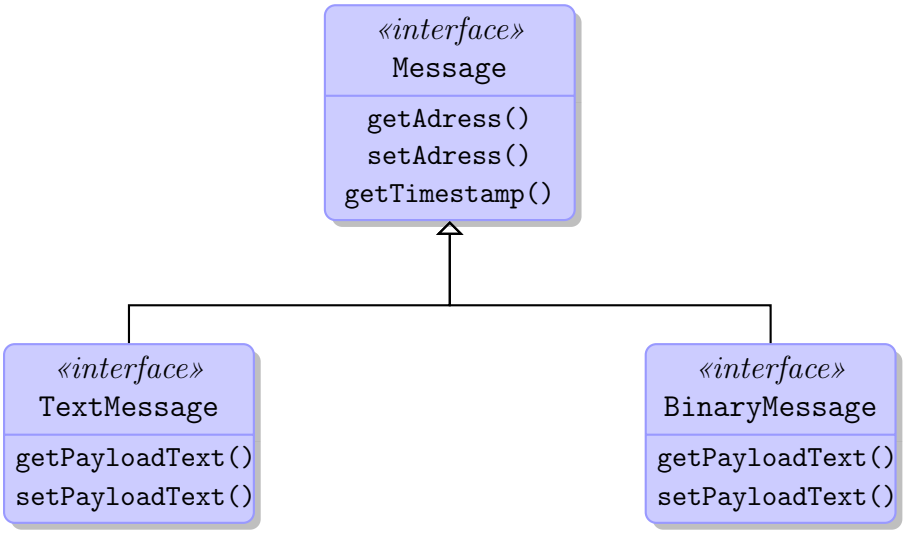
Listing 7.1: *Wiadomość SMS [13]*

Na szczęście większość telefonów komórkowych implementuje mechanizmy dekodowania powyższej informacji, a programistom pozostaje jedynie wykorzystanie dedykowanego *API*.

7.1 Wykorzystanie interfejsu programistycznego dla bezprzewodowego wys

W przypadku urządzeń z *J2ME* specyfikacja *JSR-120* określa pakiety klas dedykowanych do wysyłania i odbierania wiadomości. *Wireless Messaging API* (*WMA*) został przewidziany dla urządzeń konfiguracji *CLDC* i *CDC* obsługujących protokoły *SMS* i *CBS* [34].

Twórcy *WMA* dokonali naturalnego podziału wiadomości *SMS* na dwie części: nagłówka i danych. Widoczne jest to w pakiecie `javax.wireless.messaging`. Definiując interfejs `Message` założono implementacje takich metod jak `getAdress` `setAddress` czy `getTimestemp`, które stanowią części nagłówka. Analogicznie postąpiono z częścią opisującą dane, jednak ponieważ treść wiadomości może być przesyłana w formie tekstowej lub binarnej utworzono dwa interfejsy: `TextMessage` oraz `BinaryMessage`. Relacje te przedstawiono na rysunku 7.2.



Rysunek 7.2: *interfejsy* `javax.wireless.messaging`

7.1.1 Wysyłanie *SMS*a

Przedstawiony opis charakteryzuje jednak tylko wewnętrzną budowę wiadomości, nie informując w jaki sposób należy ją utworzyć. Zaproponowany mechanizm wysyłania/odbierania wiadomości jest relatywnie prosty i składa się z trzech kroków:

- uzyskać 'połączenie',
- przygotować wiadomość,

- wysłać wiadomość.

Aby zrealizować pierwszy krok niezbędne jest skorzystanie z *GCF* i klasy `javax.microedition.io.Connector`, która wywołując metodę `open` oczekuje parametru reprezentującego *URL*. *WMA* przewiduje trzy rodzaje zapisu *UML*a:

- `sms://+123456789` – interpretowany jako numer odbiorcy 123-45-67-89,
- `sms://+12345678910:1234` – dodatkowo określono port na który należy dostarczyć *SMS*,
- `sms://:1234` – określenie numeru portu na którym serwer będzie oczekiwał lub wysyłał informacje.

Dysponując połączeniem dla *URL*a rozpoczętego od `sms://`, można dokonać konwersji do `MessageConnection`, która pozwoli na wywołanie metod `newMessage` i `send`. Można zatem zrealizować krok drugi, wykorzystując metodę `newMessage` utworzyć wiadomość i zapisać w niej dane.

Trzeci i ostatni krok zakłada wyslanie przygotowanej informacji. Operacja ta jest realizowana przez metodę `send`, która jako parametr przyjmie obiekt wysyłanej wiadomości.

Realizację wszystkich trzech kroków przedstawiono w listingu 7.2.

```
String addr = "sms://+123456789";
MessageConnection conn = (MessageConnection) Connector.open(addr);
TextMessage msg = (TextMessage) conn.newMessage(MessageConnection.TEXT_MESSAGE);
msg.setPayloadText("Hello world");
conn.send(msg)
```

Listing 7.2: *Przykładowy program wysyłający wiadomość SMS*

Operację wysyłania można również zrealizować w oparciu o połączenie serwerowe. Jako że ustanowienie takiego połączenia ma postać:

```
String addr = "sms://:1701";
MessageConnection conn = (MessageConnection) Connector.open(addr);
```

widać że nie wprowadzono numeru pod który należy wysłać wiadomość. Dlatego ciężar tej operacji został przeniesiony na `Message`, która przez metodę `setAddress` będzie musiała ustawić odbiorcę.

```
msg.setAddress("sms://+123456789:1701");
```

Wykorzystanie połączenia serwerowego stosowane jest w przypadkach gdy użytkownik wielokrotnie wysyła informacje. Podyktowane jest to względami oszczędności, w rozwiązaniu 7.2 dla wysłania każdej wiadomości niezbędne jest nawiązanie połączenia którego utworzenie zajmuje cenny czas. Problem ten nie występuje w przypadku połączenia serwerowego.

7.1.2 Odbieranie *SMS*a

Wiadomości wysyłane w rozdziale 7.1.1 trzeba odebrać. W tym celu utworzony zostanie jeszcze jeden program 7.3. Z racji swojej funkcjonalności będzie on zbliżony do programów wysyłających informacje, bowiem aby ropocząć nasłuch przychodzących wiadomości konieczne jest ustanowienie połączenia serwerowego.

Dysponując połączeniem wykorzystamy metodę `receive` odczytującą przychodzące wiadomości.

```
String addr = "sms://:1701";
MessageConnection conn = (MessageConnection) Connector.open(addr);
Message msg = conn.receive();
```

Listing 7.3: *Przykładowy program odczytujący wiadomość SMS*

Program przedstawiony w listingu 7.3 pozwala na odczytanie wyłącznie pojedynczej wiadomości. Naturalnym rozwiązaniem będzie więc umieszczenie metody wczytującej wiadomość w pętli co pozwoli przyjąć dowolną ich liczbę. Należy jednak zwrócić uwagę na fakt iż metoda `receive` będzie oczekiwała na wiadomości, co może doprowadzić do zablokowania programu. Dlatego zaleca się umieszczenie tej operacji w odrębnym wątku.

Istnieje również alternatywne rozwiązanie, dokumentacja *WMA* twierdzi że interfejs `MessageConnection` zawiera metodę `setMessageListener` pozwającą zarejestrować obiekt nasłuchujący zgodny z interfejsem `MessageListener`. Podążając dalej za opisem, jeżeli w kolejce przychodzących wiadomości pojawi się nowy element, natychniast zostanie wywołana metoda `notifyIncomingMessage`. Pozwala to na sprowadzenie rozwiązania problemu do kodu przedstawionego w listingu 7.4.

```
String addr = "sms://:1701";
MessageConnection conn = (MessageConnection) Connector.open(addr);
conn.setMessageListener(new MyMessageListener());

class MyMessageListener
    implements MessageListener{

    public void notifyIncomingMessage(MessageConnection conn) {
        Message msg = null;
        try{
            msg = conn.receive();
        }catch(Exception ex){
            // Error msg
        }

    }
}
```

Listing 7.4: *Przykładowy program odczytujący wiadomość SMS*

7.2 Uruchamianie aplikacji MIDlet-owej po otrzymaniu SMS-a

Przedstawiona w rozdziale 1 technika uruchamiania *MIDlet*ów była niewystarczająca. Dlatego wraz z wprowadzeniem *MIDP 2.0* dodano mechanizm nazwany *Push Registry*, który pozwala aplikacjom *MIDlet*owym na samodzielne uruchamianie się. Przewidziano dwie metody aktywowania *MIDlet*u w oparciu o mechanizm:

- połączenia sieciowego,
- ustawienia czasu.

Oba mechanizmy uzupełniły możliwość ręcznego wyboru uruchamianej aplikacji i z punktu widzenia cykłu życia *MIDlet*'u nie wprowadziły żadnych zmian.

Zasadę działania tego mechanizmu można przedstawić jako mapowanie nazwy *MIDlet*u i identyfikatora zasobu sieciowego. W momencie nadejścia połączenia następuje dopasowanie do zarejestrowanych zasobów, a znalezienie zasobu skutkuje uruchomieniem *MIDlet*u [26].

7.2.1 Rejestracja

Aby zarejestrować *MIDlet* niezbędne jest wykonane trzech operacji:

- Konieczne jest utworzenie ciągu znaków reprezentującego zasób sieciowy. W przypadku wiadomości tekstowych przychodzących na port 1701 ciąg znaków będzie miał postać: `sms://1701`.
- Określenie nazwa klasy *MIDlet*'u który należy uruchomić.
- Zdefiniowanie którzy z nadawców mogą skorzystać z tej usługi. W przypadku gdy będzie to nieograniczona grupa można użyć symbolu '*'.

Powyższe operacje można wykonać z poziomu uruchomionego programu lub umieścić je w zestawie instalacyjnym *MIDlet*'u.

W przypadku programowej rejestracji należy skorzystać z klasy `javax.microedition.io.PushRegistry` i jej statycznej metody `registerConnection`. Operacje tę można zatem przedstawić w następujący sposób:

```
PushRegistry.registerConnection(
    "sms://:50000",
    "PushyMIDlet",
    "*"
);
```

Przeciwną do rejestracji jest operacja wyrejestrowania, która jest realizowana jak następuje:

```
PushRegistry.unregisterConnection(
    "sms://:50000",
);
```

8 Komunikacja bezprzewodowa

8.1 Technologia Bluetooth [30]

8.1.1 Historia

W 1994 r. firma Ericsson Mobile Communications rozpoczęła pracę nad konstrukcją taniego łącza radiowego niskiej mocy, które miało służyć połączeniu telefonu komórkowego z różnymi akcesoriami. Łącza kablowe były bardzo skuteczne i efektywne, ale niewygodne. Alternatywą mogło być połączenie z wykorzystaniem podczerwieni, jednak komunikacja taka wymaga bezpośredniego 'widzenia' się urządzeń, dlatego też wybór padł na łącze radiowe. Tanie połączenie radiowe otwierało zupełnie nowe perspektywy zastosowań, jakich nie oferowały poprzednie rozwiązania. Zapoczątkowana przez Ericssona idea zakładająca opracowanie standardu komunikacji radiowej niewielkiej mocy, o ograniczonym zasięgu, zainteresowała wielu producentów urządzeń bateryjnych oraz komputerów przenośnych. Po rocznych konsultacjach, w lutym 1998 r. najbardziej zainteresowane konsorcja: Ericsson, Nokia, IBM, Intel i Toshiba, utworzyły specjalny zespół Bluetooth SIG (Special Interest Group), który 26 czerwca 1999 roku ogłosił specyfikację standardu Bluetooth 1.0.

Nazwa 'Bluetooth' pochodzi od przydomka średniowiecznego króla wikingów Haralda Blaatanda II, zwanego Sinozębym (żył w latach 940 – 985 n.e.), który to podczas swojego panowania zjednoczył Danię oraz Norwegię, a powstałe w ten sposób państwo miało wielki wpływ na losy średniowiecznej Europy oraz świata. Twórcy technologii Bluetooth postawili sobie podobny cel do osiągnięcia, pragnąc aby technologia Bluetooth służyła zunifikowanej komunikacji wszelkich urządzeń elektronicznych.

W przeciągu roku po dacie upublicznienia specyfikacji standardu Bluetooth, do konsorcjum przystąpiło ponad 650 producentów różnorodnych urządzeń cyfrowych oraz elektronicznych (za [27]), co oznaczało przyjęcie w najbliższym czasie standardu Bluetooth jako standardu ogólnoświatowego. Następstwem tego były ówczesne oszacowania, iż do roku 2002 ponad 100 mln. urządzeń takich jak telefony komórkowe, palmtopy i inne zostanie wyposażonych w łącze Bluetooth (za [27]), na rok obecny szacunki te wynoszą około miliard sztuk rocznie. Tym bardziej, że ciągle narasta potrzeba wymiany danych bezprzewodowo przez coraz tańsze i popularniejsze telefony komórkowe oraz niewielkie urządzenia przenośne.

Obecnie szeroko stosowany jest standard Bluetooth 1.1, w ubiegłym roku zaś swoją premierę miał standard Bluetooth 2.0. Wbrew oczekiwaniom nie powiększono w nim zasięgu w stosunku do standardu 1.1, jednak wprowadził wiele usprawnień. Najważniejsze z nich to:

- trzykrotnie szybszy transfer danych;
- niższe zużycie energii;
- uproszczenie połączeń typu “multi-link”;

a wszystko to przy zachowaniu wstecznej kompatybilności ze starszymi wersjami standardu.

8.1.2 Podstawowe cechy technologii Bluetooth

Technologia Bluetooth służy do transmisji danych z przepustowością od 1Mb/s do 12 Mb/s na standardową odległość do 10 metrów, przy czym zasięg może być wydłużony do 100 metrów przy użyciu opcjonalnego wzmacniacza oraz anteny. Połączenie takie jest w stanie zapewnić przesyłanie danych, w tym dźwięku oraz obrazu. Każde urządzenie wyposażone w łącze Bluetooth jest w stanie wymieniać dane z innymi urządzeniami o ile znajduje się w ich zasięgu, w przeciwieństwie do technologii wykorzystujących promienie podczerwone unikamy tutaj problemu 'widzenia się' urządzeń. Możliwe jest na przykład użycie telefonu komórkowego zamiast popularnych kart magnetycznych służących do identyfikacji użytkowników, stosowanych obecnie w wielu firmach. Zgodnie z założeniami urządzenia powinny być w stanie zidentyfikować oraz zsynchronizować się nawzajem. Bez wątpienia z punktu widzenia użytkownika jest to jedna z najważniejszych cech nowego standardu, która przemawia na jego korzyść. Ma ona jednak także niebagatelny wpływ na kwestię bezpieczeństwa takich połączeń, jako że użytkownik z założenia nie powinien ingerować w proces synchronizacji urządzeń, jednak problem ten został dobrze rozwiązany. Łącze Bluetooth zostało wzbogacone o mechanizmy odpowiedzialne za potwierdzanie autentyczności oraz szyfrowanie, dzieki czemu istnieje możliwość odrzucania niechcianych połączeń. Celem zapewnienia kompatybilności połączeń z telefonami komórkowymi, notebookami i palmtopami różnych producentów, Bluetooth Special Interest Group (SIG) ciągle prowadzi programy szkoleń dla projektantów urządzeń podczas których zespoły konstrukcyjne mogą testować swoje produkty wyposażone w łącza Bluetooth, badając ich kompatybilność ze specyfikacją standardu oraz produktami innych wytwórców.

Pasmo częstotliwości wykorzystywane przez standard Bluetooth ma szerokość około 80MHz i wykorzystuje częstotliwość około 2,4GHz, a dokładnie leżącą w zakresie 2,402-2,480GHz (z niewielkimi odstępstwami w niektórych krajach), który to zakres jest wolny we wszystkich niemal krajach i nie wymaga licencji na korzystanie z niego. Pasmo zostało podzielone na 79 kanałów odległych od siebie o 1 MHz. W celu uniknięcia przypadkowych przechwyceń zastosowana została technika rozpraszania widma z przeskokiem częstotliwości (ang. Spread Spectrum with Frequency Hopping), która polega na nieustannej zmianie częstotliwości pracy nadajnika. W celu zapewnienia komunikacji pomiędzy nadajnikiem oraz odbiornikiem również częstotliwość pracy odbiornika musi podlegać nieustannej zmianie w takt pracy nadajnika. Dane wysyłane są jako pakiety z których każdy jest nadawany na innej częstotliwości. Zmiana częstotliwości sygnału następuje 1600 razy na sekundę, więc dane transmitowane są na jednej częstotliwości przez 625 mikrosekund.

Częstotliwości zmieniane są zgodnie z generatorem liczb pseudolosowych, czego następstwem jest fakt że oba urządzenia zarówno nadawca jaki i odbiorca muszą być wyposażone w taki generator oraz muszą one być ze sobą zsynchronizowane. Standard Bluetooth przewiduje również posiadanie przez każde z urządzeń unikalnego 48-bitowego znacznika, który stanowi jednocześnie jego adres BDA (ang. Bluetooth Device Address), co znacznie upraszcza identyfikację urządzenia w sieci.

Istotnym zagadnieniem w standardzie Bluetooth są profile, od których zależy dalszy rozwój tej technologii. Twórcy standardu poświęcili bardzo wiele miejsca w dokumentach standaryzacyjnych opisowi szeregu profili aplikacji, czyli ogólnych wy-

magań stawianych oprogramowaniu, umożliwiającemu realizację różnego typu usług telekomunikacyjnych. Zatem profile służą zapewnieniu kompatybilności między aplikacjami oraz urządzeniami Bluetooth pochodzącymi od różnych producentów. W standardzie Bluetooth zdefiniowano 13 różnych profili aplikacji oznaczonych symbolami od K1 do K13.

K1 ogólny profil dostępu GAP (Generic Access Profile) Podstawowy profil dostępu wprowadza definicje zalecenia i wspólne wymagania dotyczących podstawowych trybów pracy i procedur dostępu. Określa on zachowanie urządzenia w stan oczekiwania i połączenia, które umożliwia zestawienie połączenia pomiędzy urządzeniami Bluetooth, analizę stanu otoczenia i zapewnia odpowiednią poufność.

K2 profil aplikacji wykrywania usług SDAP (Service Discovery Application Profile) Profil ten umożliwia identyfikację usług realizowanych w innych urządzeniach ściągnięcie dostępnych informacji dotyczących tych usług

K3 profil dla telefonii bezprzewodowej CTP (Cordless Telephony Profile) Profil CTP, który definiuje właściwości i procedury wymagane do współpracy pomiędzy różnymi elementami telefonu „trzy w jednym". Telefon „trzy w jednym"to rozwiązanie wprowadzające dodatkowy tryb pracy telefonu komórkowego jako radiotelefonu bliskiego zasięgu do połączenia z siecią stacjonarną poprzez stację bazową

K4 profil dla bezprzewodowej komunikacji wewnętrznej IntP (Interkom Profile) Profil interkomu definiuje wymagania dla urządzeń Bluetooth dotyczące połączeń bezpośrednich pomiędzy telefonami typu „trzy w jednym"tzw. usługa interkomu

K5 profil wirtualnego portu szeregowego SPP (Serial Port Profile) Profil portu szeregowego opisuje wymagania związane z realizacją emulowanego radiowego łącza szeregowego np. pomiędzy dwoma komputerami

K6 profil dla bezprzewodowego zestawu słuchawkowego HP (Headset Profile)-umożliwia jej bezprzewodowe połączenie i pełnienie roli urządzenia wejściowego i wyjściowego dla sygnałów dźwiękowych (audio)

K7 profil usług modemowych DUN (Dial-up Networking Profile)-profil dostępu do sieci stosowany jest przez komputer do uzyskania komputerowego dostępu do Internetu poprzez telefon komórkowy lub modem

K8 profil usług telefaksowych FP (Fax Profile)

K9 profil dostępu do sieci lokalnej LA (LAN Access Profile) Definiuje zestaw procedur zapewniający bezprzewodowy dostęp do sieci LAN. Zdefiniowano trzy typy zastosowań tego profilu:

1. Udostępnianie połączenia z siecią lokalną pojedynczej stacji Bluetooth

2. Klasyczny punkt dostępowy, pozwalający na jednoczesne przyłączenie do sieci lokalnej większej liczby urządzeń

3. Bezpośrednie połączenie między urządzeniami

K10 ogólny profil wymiany danych w postaci obiektów GOEP (Generic Object Exchange Profile) W ramach usługi transmisji szeregowej wyodrębniono specjalna grupę profili które precyzują wymagania odnośnie wymiany danych w podstacji obiektów .Przykładem wykorzystania tego profilu jest mogą być aplikacje służące do synchronizacji danych ,przesyłania danych oraz wymiany informacji. Urządzeniami które najczęściej korzystają z tego typu profili są laptopy, notatniki elektroniczne czy telefony komórkowe. Profil GOEP określa reguły komunikacji typu „wyślij i pobierz" bazując na architekturze klient – serwer

K11 profil przesyłania obiektów OPP (Object Push Profile) Profil przesyłania obiektów OPP definiuje trzy podstawowe rodzaje operacje:

1. przesyłanie jednego lub więcej obiektów
2. pobranie tzw. wizytówki biznesowej
3. wymianę wizytówek, rozumianą jako następujące po sobie operacje 1 i 2 Dane przesyłane przez aplikacje oparte o ten profil są zapisywane w formatach: vCard2.1, vCalendar, vMessage, vNote.

K12 profil przesyłania plików FTP (File Transfer Profile) Profil aplikacji transferu plików FTP umożliwia przesyłanie danych w łączu bezprzewodowym. W ramach profilu FTP zdefiniowano następujące rodzaje operacji: a) Wybór serwera FTP z listy dostępnych serwerów tzn. pozostających w zasięgu radiowym urządzenia. b) Przeglądanie zasobów serwera c) Kopiowanie obiektów (pliku lub folderu) z serwera i na serwer. d) Kasowanie plików lub folderów oraz zakładanie nowego folderu na serwerze. Stroną inicjującą połączenie jest klient. Aplikacje działające zgodnie z protokołem FTP maja możliwość przesyłania oraz poprawnego interpretowania poleceń protokołu OBEX służących do wyświetlania zawartości folderów

K13 profil synchronizacji danych SP (Synchronization Profile) Wirtualny port szeregowy stanowi rozszerzenie ogólnego profilu dostępu GAP o elementy konieczne do zapewnienia transmisji w trybie szeregowym. Należą do nich procedury protokołu RFCOMM, zapewniając transport danych oraz, nie podlegający ścisłej standaryzacji moduł emulatora portu szeregowego. Zadaniem emulatora jest ukrycie przed oprogramowaniem użytkownika, bezprzewodowego charakteru łącza

Jak widać nowa technologia pokonuje przeszkody których nie były w stanie pokonać istniejące dotychczas połączenia kablowe czy wykorzystujące podczerwień. Te pierwsze o ile są skutecznie i niezawodne to sprawiają wiele problemów samym użytkownikom. Podczerwień jest technologią, która swoje lata świetności ma już dawno za sobą i o ile rozwiązuje problem uporczywych przewodów o tyle o jej niezawodności i zasiegu nie można już mówić tak dobrze. Bluetooth nie wymaga bezpośredniej widoczności urządzeń, rozwiązuje również problem uciążliwych kabli oferując jednocześnie lepszy niż IrDA zasieg, jednak narażony jest na wszelkiego rodzaju ataki polegające na podsłuchu pasma.

8.2 Praktyczna realizacja połączeń klient/serwer [17]

8.2.1 Połączenie od strony serwera

Komunikacja odbywa się metodą klient/serwer. Nawiązanie połączenia po stronie serwera przedstawia listing 8.1.

```
1  public void serverConnection() {
2          LocalDevice localDevice = null;
3          try {
4              localDevice = LocalDevice.getLocalDevice();
5
6              // Serwer będzie mógł być wykrywany przez potencjalnych klientów.
7              localDevice.setDiscoverable(DiscoveryAgent.GIAC);
8          } catch (BluetoothStateException e) {
9              return;
10         }
11
12         // Unikalny identyfikator usługi. Nie powinno być dwóch usług, które
13         // mają taki sam identyfikator. Sposób generowania identyfikatora
14         // opisuje dokument RFC4122. UUID można zdobyć używając np. tej strony:
15         // http://www.famkruithof.net/uuid/uuidgen.
16         String uuid = "ff058120f61111dc95ff0800200c9a66";
17         UUID serviceUUID = new UUID(uuid, false);
18
19         // Nadajemy nazwę usłudze.
20         String serviceName = "MojaNazwa";
21
22         // Tworzymy url usługi.
23         String url = "btspp://localhost:" + serviceUUID.toString() + ";name="
24                 + serviceName;
25
26         DataInputStream dis = null;
27         DataOutputStream dos = null;
28         StreamConnection streamConnection = null;
29         StreamConnectionNotifier streamConnectionNotifier = null;
30         try {
31             // Dodajemy usługę do bazy wykrywalnych
32             // usług (Service Discovery Database - SDDB).
33             streamConnectionNotifier = (StreamConnectionNotifier) Connector
34                     .open(url);
35
36             // Istnieje możliwość dodawania atrybutów usługi. Przykładowo,
37             // chcemy umieścić informację
38             // o maksymalnej liczbie połączeń. Klient pobiera atrybuty przy
39             // wyszukiwaniu usługi, czyli jeszcze zanim się połączy.          ↩
                   // Atrybuty możemy zmieniać także w trakcie działania serwera.  ↩
                       // Jeśli nie chcemy dodawać żadnych własnych atrybutów do   ↩
                           // usługi, pomijamy część kodu,odpowiedzialną za ich ↩
                   dodawanie.
40
41             ServiceRecord serviceRecord = localDevice
42                     .getRecord(streamConnectionNotifier);
43
44             // Identyfikator atrybutu, który będzie musiał podać klient        ↩
                   // przy wyszukiwaniu usługi,jeśli chce pobrać wartość tego      ↩
                   // atrybutu. Ponieważ są atrybuty, automatycznie dodawane
45             // do usługi (np. nazwa usługi to atrybut od identyfikatorze
46             // 0x100), bezpiecznie będzie,
47             // jeśli własne atrybuty zaczniemy numerować od 1000, by nie
48             // kolidowały
49             // ze standardowymi atrybutami.
50             int attrID = 1000;
51
52             // Tworzymy obiekt, odpowiadający wartości atrybutu. W          // ↩
                   parametrze konstruktora podajemy typ wartości oraz wartość       ↩
                   // atrybutu.
53             DataElement dataElement = new DataElement(attrID, 10);
54
55             // Ustawiamy atrybut usługi.
56             serviceRecord.setAttributeValue(attrID, dataElement);
57
58             // Poniższa pomijamy, jeśli ustawiamy atrybuty przed wywołaniem
```

```
59              // metody streamConnectionNotifier.acceptAndOpen().
60              localDevice.updateRecord(serviceRecord);
61
62              // Metoda ta jest blokuje dalsze wykonanie programu dopóki      // ↩
                    jakiś klient nie będzie chciał się połączyć.
63
64              streamConnection = streamConnectionNotifier.acceptAndOpen();
65
66              // Wysyłamy i odbieramy dane.
67              String message = "Witamy klienta!";
68              dis = streamConnection.openDataInputStream();
69              dos = streamConnection.openDataOutputStream();
70              dos.writeUTF(message);
71              System.out.println("Wiadomosc od klienta: " + dis.readUTF());
72
73          } catch (IOException e) {
74          } finally {
75              // Zamykamy połączenie.
76              try {
77                  dis.close();
78              } catch (Exception e) {
79              }
80              try {
81                  dos.close();
82              } catch (Exception e) {
83              }
84              try {
85                  streamConnection.close();
86              } catch (Exception e) {
87              }
88              try {
89                  streamConnectionNotifier.close();
90              } catch (Exception e) {
91              }
92          }
93
94          try {
95              localDevice.setDiscoverable(DiscoveryAgent.NOT_DISCOVERABLE);
96          } catch (BluetoothStateException e) {
97          }
98  }
```

Listing 8.1: *Metoda serverConnection()*

8.2.2 Wyszukiwanie usług w zasięgu

Mamy już działający serwer, oczekujący na połączenie. Teraz potrzebujemy klienta, który się do niego podłączy. Pierwszą fazą połączenia od strony klienta jest wyszukanie dostępnych w zasięgu usług, do których można się połączyć. Faza ta składa się z wykrycia urządzeń Bluetooth w zasięgu, a następnie przeszukania wykrytych urządzeń pod kątem dostępności usług. Listing 8.2 obrazuje przebieg tej procedury.

```
1   public class BTServiceSearch implements DiscoveryListener {
2
3       private Vector discoveredDevices, discoveredServices;
4
5       private int serviceSearchCount;
6
7       public BTServiceSearch(UUID[] serviceUUIDs, int[] attrids) {
8
9           discoveredDevices = new Vector();
10          discoveredServices = new Vector();
11          DiscoveryAgent discoveryAgent = null;
12
13          try {
14              discoveryAgent =                                            ↩
                    LocalDevice.getLocalDevice().getDiscoveryAgent();
15
16              // Zaczynamy wyszukiwanie urządzeń w zasięgu. W drugim          // ↩
                    parametrze podajemy DiscoveryListener'a.
```

```
17              discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
18
19              // Czekamy, aż wszystkie urządzenia w zasięgu zostaną wykryte.      ↩
                    // Jeśli jakieś urządzenie zostanie znalezione, wtedy           ↩
                    // wywoływana jest metoda DiscoveryListener.deviceDiscovered(). ↩
                            // Po zakończeniu wyszukiwania urządzeń wywoływana jest ↩
                    metoda
20              // DiscoveryListener.inquiryCompleted().
21
22              synchronized (this) {
23                  try {
24                      this.wait();
25                  } catch (Exception e) {
26                  }
27              }
28          } catch (BluetoothStateException e) {
29              // Wyszukiwanie urządzeń nie powiodło się.
30              return;
31          }
32          if (discoveredDevices.size() == 0)
33              return;
34
35              // Jeśli znaleźliśmy inne urządzenia Bluetooth w zasięgu,           ↩
                    //przystępujemy
36              // do wyszukiwania na nich usług.
37
38          int maxServiceSearches = 1;
39          try {
40              // Odczytujemy maksymalną ilość wyszukiwań, które mogą być
41              // prowadzone jednocześnie.
42
43              maxServiceSearches = Integer.parseInt(LocalDevice
44                      .getProperty("bluetooth.sd.trans.max"));
45
46          } catch (NumberFormatException e) {
47          }
48
49          for (int i = 0; i < discoveredDevices.size(); i++) {
50
51              RemoteDevice remoteDevice = (RemoteDevice) discoveredDevices
52                      .elementAt(i);
53              try {
54                  try {
55                          // Zaczynamy wyszukiwanie usług. Jeśli nie chcemy       ↩
                                    // pobierać atrybutów, wtedy jako attrids
56                          // wpisujemy wartość null.
57
58                      discoveryAgent.searchServices(attrids, serviceUUIDs,
59                              remoteDevice, this);
60
61                  } catch (NullPointerException e) {
62                          // Czasem na emulatorze Sun'a, gdy dwa uruchomione      ↩
                                    // emulatory wyszukują się wzajemnie, to ↩
                                pojawia
63                          // się ten wyjątek, chociaż nie powinien
64                          //. Na wszelki wypadek przechwycamy.
65
66                      return;
67
68                  } catch (IllegalArgumentException iae) {
69                  // Ten wyjątek oznacza, że daliśmy za dużo atrybutów. w
70                      // sposób pokazany poniżej. Niestety w praktyce zdarza      ↩
                                // się, że pojawia się ten wyjątek przy prawidłowej
71                      // liczbie podanych atrybutów. Problem ten zauważyłem
72                      // na telefonach Motorola.
73
74                      int maxAttrs = Integer.parseInt(LocalDevice                ↩
                                .getProperty("bluetooth.sd.attr.retrievable.max"));
75                      System.out.println("Za duzo atrybutow. Max moze byc "
76                              + maxAttrs + ".");
77                      return;
78                  }
79              } catch (BluetoothStateException e) {
80                      // Nie można zacząć wyszukiwania na tym urządzeniu.         ↩
                                    // Próbujemy na następnym
81
```

```
82                  continue;
83              }
84
85                  // Sprawdzamy, czy następne wyszukiwanie może zostać                ↩
                        //rozpoczęte.
86                  // Jeśli nie, to czekamy aż poprzednie się zakończy.                ↩
                        // Jeśli zostanie znalezina usługa, wtedy wywołana              ↩
                            //zostanie metoda DiscoveryListener.↩
                        servicesDiscovered().
87              // Po zakończeniu wykrywania usług, wywoływana jest                     ↩
                    // metoda DiscoveryListener.serviceSearchCompleted().
88
89              synchronized (this) {
90                  serviceSearchCount++;
91                  if (serviceSearchCount == maxServiceSearches) {
92                      try {
93                          this.wait();
94                      } catch (Exception e) {
95                      }
96                  }
97              }
98          }
99          // Czekamy na zakończenie wszystkich wyszukiwań.
100         while (serviceSearchCount > 0) {
101             synchronized (this) {
102                 try {
103                     this.wait();
104                 } catch (Exception e) {
105                 }
106             }
107         }
108
109             // Zakończenie wyszukiwania urządzeń w zasięgu z uruchomioną            ↩
                    // usługą, która nas interesuje, zaostało zakończone pomyślnie. ↩
                            // Teraz, mając tablicę discoveredServices, która ↩
                    przechowuje        // znalezione usługi, możemy spróbować połączyć↩
                     się, z którąś        // ze znalezionych usług.
110     }
111
112     public void deviceDiscovered(RemoteDevice remoteDevice,
113             DeviceClass deviceClass) {
114
115         // Dodajemy wykryte urządzenie do tablicy.
116         discoveredDevices.addElement(remoteDevice);
117     }
118
119     public void inquiryCompleted(int param) {
120
121         synchronized (this) {
122             try {
123                 this.notifyAll();
124             } catch (Exception e) {
125             }
126         }
127     }
128
129     public void servicesDiscovered(int transID, ServiceRecord[] sr) {
130
131         for (int i = 0; i < sr.length; i++) {
132             // Jeśli usługi nie ma jeszcze w tablicy wykrytych usług,wtedy
133             // dodajemy ją.
134
135             if (discoveredServices.indexOf(sr[i]) == -1)
136                 discoveredServices.addElement(sr[i]);
137         }
138     }
139
140     public void serviceSearchCompleted(int transID, int respCode) {
141
142         serviceSearchCount--;
143         synchronized (this) {
144             this.notifyAll();
145         }
146     }
147 }
```

Listing 8.2: *Klasa BTServiceSearch*

Należy pamiętać o tym, że wyszukiwanie usług powinno przebiegać według podanego schematu. Nie należy rozpoczynać wyszukiwania usługi w metodzie inquiryCompleted() lub serviceSearchCompleted(). Skutkuje to wystąpieniem błędów: wyszukiwanie kończy się a metoda serviceSearchCompleted() nie jest wywoływana. Przykładem telefonu podatnego na taki błąd jest model Nokia N73.

8.2.3 Połączenie od strony klienta

Gdy usługi dostepne w zasięgu zostaną wyszukane, należy się do nich połączyć. Przedstawia to listing 8.3.

```
1  public class BTServiceSearch implements DiscoveryListener {
2
3      private Vector discoveredDevices, discoveredServices;
4
5      private int serviceSearchCount;
6
7      public BTServiceSearch(UUID[] serviceUUIDs, int[] attrids) {
8
9          discoveredDevices = new Vector();
10         discoveredServices = new Vector();
11         DiscoveryAgent discoveryAgent = null;
12
13         try {
14             discoveryAgent =                                          ↩
                   LocalDevice.getLocalDevice().getDiscoveryAgent();
15
16             // Zaczynamy wyszukiwanie urządzeń w zasięgu. W drugim          // ↩
                   parametrze podajemy DiscoveryListener'a.
17             discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
18
19             // Czekamy, aż wszystkie urządzenia w zasięgu zostaną wykryte.      ↩
                   // Jeśli jakieś urządzenie zostanie znalezione, wtedy           ↩
                   // wywoływana jest metoda DiscoveryListener.deviceDiscovered(). ↩
                           // Po zakończeniu wyszukiwania urządzeń wywoływana jest ↩
                   metoda
20             // DiscoveryListener.inquiryCompleted().
21
22             synchronized (this) {
23                 try {
24                     this.wait();
25                 } catch (Exception e) {
26                 }
27             }
28         } catch (BluetoothStateException e) {
29             // Wyszukiwanie urządzeń nie powiodło się.
30             return;
31         }
32         if (discoveredDevices.size() == 0)
33             return;
34
35             // Jeśli znaleźliśmy inne urządzenia Bluetooth w zasięgu,           ↩
                   //przystępujemy
36             // do wyszukiwania na nich usług.
37
38         int maxServiceSearches = 1;
39         try {
40             // Odczytujemy maksymalną ilość wyszukiwań, które mogą być
41             // prowadzone jednocześnie.
42
43             maxServiceSearches = Integer.parseInt(LocalDevice
44                     .getProperty("bluetooth.sd.trans.max"));
45
46         } catch (NumberFormatException e) {
47         }
48
49         for (int i = 0; i < discoveredDevices.size(); i++) {
50
51             RemoteDevice remoteDevice = (RemoteDevice) discoveredDevices
52                     .elementAt(i);
```

```
53              try {
54                  try {
55                          // Zaczynamy wyszukiwanie usług. Jeśli nie chcemy        ↩
                                    // pobierać atrybutów, wtedy jako attrids
56                          // wpisujemy wartość null.
57
58                      discoveryAgent.searchServices(attrids, serviceUUIDs,
59                              remoteDevice, this);
60
61                  } catch (NullPointerException e) {
62                          // Czasem na emulatorze Sun'a, gdy dwa uruchomione       ↩
                                    // emulatory wyszukują się wzajemnie, to ↩
                                pojawia
63                          // się ten wyjątek, chociaż nie powinien
64                          //. Na wszelki wypadek przechwycamy.
65
66                      return;
67
68                  } catch (IllegalArgumentException iae) {
69                  // Ten wyjątek oznacza, że daliśmy za dużo atrybutów. w
70                      // sposób pokazany poniżej. Niestety w praktyce zdarza       ↩
                                // się, że pojawia się ten wyjątek przy prawidłowej
71                      // liczbie podanych atrybutów. Problem ten zauważyłem
72                      // na telefonach Motorola.
73
74                      int maxAttrs = Integer.parseInt(LocalDevice                   ↩
                                .getProperty("bluetooth.sd.attr.retrievable.max"));
75                      System.out.println("Za duzo atrybutow. Max moze byc "
76                              + maxAttrs + ".");
77                      return;
78                  }
79              } catch (BluetoothStateException e) {
80                      // Nie można zacząć wyszukiwania na tym urządzeniu.          ↩
                                    // Próbujemy na następnym
81
82                  continue;
83              }
84
85                  // Sprawdzamy, czy następne wyszukiwanie może zostać             ↩
                            //rozpoczęte.
86                  // Jeśli nie, to czekamy aż poprzednie się zakończy.             ↩
                            // Jeśli zostanie znalezina usługa, wtedy wywołana       ↩
                                //zostanie metoda DiscoveryListener.↩
                        servicesDiscovered().
87              // Po zakończeniu wykrywania usług, wywoływana jest                  ↩
                    // metoda DiscoveryListener.serviceSearchCompleted().
88
89              synchronized (this) {
90                  serviceSearchCount++;
91                  if (serviceSearchCount == maxServiceSearches) {
92                      try {
93                          this.wait();
94                      } catch (Exception e) {
95                      }
96                  }
97              }
98          }
99          // Czekamy na zakończenie wszystkich wyszukiwań.
100         while (serviceSearchCount > 0) {
101             synchronized (this) {
102                 try {
103                     this.wait();
104                 } catch (Exception e) {
105                 }
106             }
107         }
108
109             // Zakończenie wyszukiwania urządzeń w zasięgu z uruchomioną        ↩
                    // usługą, która nas interesuje, zaostało zakończone pomyślnie. ↩
                            // Teraz, mając tablicę discoveredServices, która ↩
                    przechowuje       // znalezione usługi, możemy spróbować połączyć↩
                     się, z którąś       // ze znalezionych usług.
110     }
111
112     public void deviceDiscovered(RemoteDevice remoteDevice,
113             DeviceClass deviceClass) {
```

```
114
115         // Dodajemy wykryte urządzenie do tablicy.
116         discoveredDevices.addElement(remoteDevice);
117     }
118
119     public void inquiryCompleted(int param) {
120
121         synchronized (this) {
122             try {
123                 this.notifyAll();
124             } catch (Exception e) {
125             }
126         }
127     }
128
129     public void servicesDiscovered(int transID, ServiceRecord[] sr) {
130
131         for (int i = 0; i < sr.length; i++) {
132             // Jeśli usługi nie ma jeszcze w tablicy wykrytych usług,wtedy
133             // dodajemy ją.
134
135             if (discoveredServices.indexOf(sr[i]) == -1)
136                 discoveredServices.addElement(sr[i]);
137         }
138     }
139
140     public void serviceSearchCompleted(int transID, int respCode) {
141
142         serviceSearchCount--;
143         synchronized (this) {
144             this.notifyAll();
145         }
146     }
147 }
```

Listing 8.3: *Metoda* `clientConnectio`

8.2.4 Problem przy połączeniach poprzez Bluetooth

Wcześniej opisano ogólny schemat implementacji połączeń poprzez Bluetooth. Niestety często teoria jest teorią, a w praktyce okazuje się, że nie zawsze mozna polegać jedynie na teorii. W dalszej części opisane zostaną problemy napotkane podczas testowania połączeń poprzez Bluetooth na telefonach.

Zdarza się, że pobrany URL usługi ma wartość null. Problem ten dotyczy telefonów Nokia z Symbianem S60v3.

```
1  public static String getConnectionURL(ServiceRecord sr) {
2          String connectionURL = sr.getConnectionURL(
3                  ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
4          if (connectionURL != null)
5              return connectionURL;
6
7          DataElement descriptorList = sr.getAttributeValue(0x0004);
8          Enumeration enumeration = (Enumeration) descriptorList.getValue();
9          enumeration.nextElement();
10         DataElement RFCOMMdescriptor = (DataElement) enumeration.nextElement();
11         enumeration = (Enumeration) RFCOMMdescriptor.getValue();
12         enumeration.nextElement();
13         DataElement RFCOMMchannelNum = (DataElement) enumeration.nextElement();
14
15         try {
16
17             long channelNum;
18
19             if (DataElement.U_INT_8 == RFCOMMchannelNum.getDataType()
20                     || DataElement.U_INT_16 == RFCOMMchannelNum.getDataType()
21                     || DataElement.INT_16 == RFCOMMchannelNum.getDataType()) {
22                 byte[] allBytes = (byte[]) RFCOMMchannelNum.getValue();
23                 channelNum = allBytes[0];
24             } else {
```

```
25                  channelNum = RFCOMMchannelNum.getLong();
26              }
27
28              StringBuffer nameBuffer = new StringBuffer(5 + 3 + 12 + 1 + 2 + 19
29                      + 14 + 13);
30              nameBuffer.append("btspp");
31              nameBuffer.append("://");
32              nameBuffer.append(sr.getHostDevice().getBluetoothAddress());
33              nameBuffer.append(":");
34              nameBuffer.append(channelNum);
35              nameBuffer.append(";authenticate=false");
36              nameBuffer.append(";encrypt=false");
37              nameBuffer.append(";master=false");
38              connectionURL = nameBuffer.toString();
39          } catch (Exception e) {
40              System.err.println(e);
41          }
42
43          return connectionURL;
44      }
```

Listing 8.4: *Usługa o wartości* **null** *– rozwiązanie*

Listings

Bibliografia

[1] A Brif History of the Green Project. https://duke.dev.java.net/green/.

[2] A brief look at java 2 micro edition.

[3] Developing flashy mobile applications using svg and jsr 226.

[4] Eclipse. http://www.eclipse.org/.

[5] Getting started with mobile 2d graphics for j2me.

[6] J2me programming/the j2me platform.

[7] Java me graphics –the next generation.

[8] Java Platform Micro Edition Software Development Kit 3.0. http://java.sun.com/javame/downloads/index.jsp.

[9] Java SE Development Kit. http://java.sun.com/javase/downloads/index.jsp.

[10] MIDP Emulators. http://developers.sun.com/mobility/midp/articles/emulators.

[11] Mobile Tools for Java. http://www.eclipse.org/dsdp/mtj/.

[12] Personal basis profile vs. personal profile: What's the difference?

[13] Sms and the pdu format.

[14] Understanding j2me application models.

[15] Wikipedia.

[16] Michał Górlicki Bartosz Gancarz. Mobilny rozkład jazdy komunikacji miejskiej, 2008.

[17] Michał Wolski Bartosz Wachowski. Inspiro – komunikator na telefony komórkowe poprzez bluetooth, 2008.

[18] Jan Bielecki. *Java 4 Swing*. Helion, 2000.

[19] Eric Giguere. An overview of the pim optional package, 2003.

[20] Vikram Goyal. J2me tutorial, part 4: Multimedia and midp 2.0, 2005.

[21] Vikram Goyal. *Pro Java ME MMAPI, Mobile Media API for Java Micro Edition*. Apress, 2006.

[22] JSR-184 Expert Group. Mobile 3d graphics api for java2 micro edition.

[23] David Hemphill James White. *Java in Small Things*. Manning Publications Co., 2002.

[24] Ville Miettinen Kimmo Roimela Jani Vaarala Kari Pulli, Tomi Aarnio. *Mobile 3D Graphics with OpenGL ES and M3G*. Elsevier, 2008.

[25] James Keogh. *J2ME: The Complete Reference*. Osbourne, 2003.

[26] Jonathan Knudsen. *Kicking Butt with MIDP and MSA*. Addison-Weley, 2008.

[27] Qusay Mahmoud. *Learning Wireless Java*. O'Reilly, 2001.

[28] Stefan Haustein Michael Kroll. *Java 2 Micro Edition Application Development*. Sams Publishing, 2002.

[29] Tomasz Parfjanowicz Paweł Mielniczuk. Multimedialna mobilna aplikacja nawigacyjna, 2007.

[30] Piotr Bucior Piotr Niżnik. Sterowanie urządzeniami elektrycznymi przy użyciu technologii bluetooth, 2007.

[31] Vartan Piroumian. *Wireless J2ME Platform Programming*. Prentice Hall PTR, 2003.

[32] Jonathan Knudsen Sing Li. *Beginning J2ME: From Novice to Professional, Third Edition*. Apress, 2005.

[33] Michał Tomaszewski. Java w komputerach naręcznych. *Software 2.0*, 65(5/2000), May 2000.

[34] Michael Juntao Yuan. *Enterprise J2ME: Developing Mobile Java Applications*. Pearson Education, 2003.