**Министерство образования Республики Беларусь**

**"Гомельский государственный университет имени Франциска Скорины"**

**Математический** факультет

Кафедра **вычислительной математики и программирования**

**Дипломная работа**

**"Использование платформы j2me для мобильных телефонов при организации видеонаблюдения"**

**Гомель 2009**

**Реферат**

**Ключевые слова:** видеонаблюдение, обнаружение движения, мидлет, платформа J2ME, Bluetooth, SMS-сообщение, MMS-сообщение, файловая система мобильного телефона, камера, библиотека LWUIT, FileConnection API JSR-75, MobileMedia API JSR-135, Bluetooth API JSR-82, Wireless Messaging API JSR-205.

**Объект исследования:** программирование мобильного телефона.

**Методы исследования:** платформа J2ME.

**Цель дипломной работы:** изучение принципов и возможностей использования ресурсов мобильного телефона при создании приложений на платформе J2ME.

**Выводы:** изучены возможности работы со следующими ресурсами телефона: файловая система, камера, Bluetooth, отправка SMS и MMS-сообщений. Разработано приложение для мобильного телефона на платформе J2ME, которое позволяет осуществлять видеонаблюдение.

# Содержание

Введение

1. Обзор литературы

2. Платформа J2ME

2.1 Основные понятия

2.2 Библиотека LWUIT

2.3 Bluetooth

3. Проектирование приложения для мобильного телефона на основе платформы J2ME

3.1 Постановка задачи

3.2 Построение интерфейса

3.3 Передача данных по Bluetooth

3.4 Работа с файловой системой телефона

3.5 Отправка SMS и MMS-сообщений

3.6 Алгоритм обнаружения движения

Заключение

Список источников

# Введение

Видеонаблюдение – процесс, осуществляемый с применением технических решений, предназначенных для визуального контроля за охраняемыми или наблюдаемыми территориями, объектами, субъектами. Главная задача видеонаблюдения – обеспечение безопасности объекта путем визуального наблюдения без передачи информации во внешние общедоступные сети.

Видеонаблюдение стало вполне обычным явлением современной повседневной жизни. Развлекательные и торговые объекты, автостоянки и гаражные комплексы, офисы, транспорт, общественные места, промышленные объекты, а также объекты банковского сектора снабжаются видеокамерами. Видеоконтроль обеспечивает предупреждение преступлений и их более быстрое раскрытие, выполняет контроль действий персонала.

В зависимости от типа используемого оборудования системы видеонаблюдения делятся на цифровые и аналоговые (традиционные). В любом случае оборудование состоит из видеокамер, устройств обработки, записывающего оборудования и устройств вывода изображения (монитор). Также в систему часто включаются дополнительные устройства типа кожухов, устройств управления, передачи на дальние расстояния и т.п.

Цифровая система видеонаблюдения обеспечивает:

* высокое качество воспроизводимой видеозаписи;
* высокую скорость доступа к видеоархиву;
* цифровое увеличение и масштабирование любого кадра;
* быстрый поиск и просмотр видеозаписи по камере, дате и времени;
* возможность интеграции с другими компьютерными системами безопасности;
* легкая и недорогая трансляция видеоархивов по каналам связи;
* возможность отправки тревожных сообщений по электронной почте и SMS;
* возможность экспорта видеоинформации на совместимые внешние носители.

Аналоговая система видеонаблюдения проста в настройке и работе и позволяет нанимать для обслуживания персонал меньшей квалификации. Минусы аналоговых систем видеонаблюдения следующие:

* ограниченность функций – обратная сторона высокой надежности;
* постоянное обслуживание – смена и архивация кассет, периодическая чистка и замена видеоголовок видеомагнитофона.

Аналоговые системы используют там, где необходимо организовать видеонаблюдение в небольшом числе помещений и информацию с видеокамер записывать на видеомагнитофон.

Учитывая широкое распространение сотовой связи, мобильные телефоны теперь используются во многих современных системах видеонаблюдения и позволяют просматривать данные от камер, а также получать SMS с уведомлением об обнаруженном движении.

Например, программа GOALmobile – это видеотелефон, осуществляющий одновременную трансляцию с одного телефона на другой. При этом нужно только, чтобы сотовый телефон поддерживал функцию GPRS. Количество одновременно транслируемых камер не ограничено. Переключение между ними производится на телефонной трубке. В составе системы безопасности данная технология позволяет получать тревожные сообщения с выводом на дисплей видеоизображения происходящего события или его записи, а также управлять исполнительными механизмами, подключенными к системе охраны. Например, при нажатии на кнопку звонка перед дверью квартиры на телефон поступит сигнал, и на дисплее будет выведена живая трансляция камеры-глазка. При этом есть возможность нажать кнопку 1 для открытия замка, кнопку 2 для включения сирены, кнопку 3 для выдачи голосового сообщения и т.д.

Кроме того, разработаны приложения для мобильных телефонов, которые используют камеру телефона для обнаружения движения. Например, Цюрихский Институт всепроникающей компьютеризации создал программу, позволяющую использовать встроенные камеры телефонов, находящихся в общей сети, в качестве системы видеонаблюдения.

Программа с названием Facet позволяет мобильным телефонам обмениваться данными с помощью Bluetooth, а благодаря специальным алгоритмам телефоны могут анализировать события, попадающие в объектив камеры. Смысл действия программы таков, что если закрепить пару телефонов, например, Nokia N6630, на потолке и включить программу Facet, то как только в поле обзора одной из камер будет попадать движущийся объект, телефон сразу же будет отсылать сигнал находящемуся рядом телефону, а также отсылать данные на компьютер через GPRS. Подобная система может использоваться в качестве охранной в тех случаях, когда полноценную систему наблюдения развернуть не представляется возможным.

Однако данные приложения дорогие и созданы в основном только для смартфонов и коммуникаторов. Поэтому актуальной задачей является создание аналогичного приложения на платформе J2ME, которая поддерживается большинством телефонов.

# 1. Обзор литературы

В настоящее время сотовая связь – самая широко используемая из всех видов мобильной связи. Наиболее распространенными видами телефонов являются обычный телефон, смартфон, коммуникатор и КПК.

Смартфоны и коммуникаторы отличаются от обычных мобильных телефонов наличием достаточно развитой операционной системы, открытой для создания программного обеспечения сторонними разработчиками. Установка дополнительных приложений позволяет значительно улучшить функциональность смартфонов и коммуникаторов по сравнению с обычными мобильными телефонами.

В настоящее время не существует четкого разграничения между смартфонами и коммуникаторами, поскольку функциональность обоих классов устройств примерно одинакова. Часто применяется так называемый "исторический подход", который заключается в следующем: если устройство ведет свою родословную от КПК – это коммуникатор, а если от мобильных телефонов – это смартфон. В рамках этого подхода под коммуникаторами обычно подразумеваются устройства с сенсорным экраном (может быть дополнен клавиатурой), работающие под управлением операционной системы Windows Mobile. Устройства, использующие для ввода информации исключительно цифровую клавиатуру (аналог телефонной), называются смартфонами. Большинство устройств под управлением операционной системы Symbian традиционно относят к смартфонам (за исключением Nokia серии 9XXX, Nokia E90 и некоторых других). В остальных случаях позиционирование устройства зависит от производителя (обычно устройства с сенсорным экранам относят к коммуникаторам, а к смартфонам относят устройства без него), хотя большинство смартфонов работает на двух конкурирующих между собой операционных системах: Symbian и Windows Mobile Smartphone Edition. В мире Symbian наиболее известными платформами являются Nokia Series 60, 80, 90 и Sony Ericsson UIQ. В лагере Windows наиболее популярна платформа HTC.

Также часть специалистов разделяет коммуникаторы и смартфоны по наличию или отсутствию полноразмерной (QWERTY) клавиатуры (виртуальной или физической).

Современные телефоны (модели средней ценовой категории и выше) прекрасно справляются со многими задачами, выходящими за рамки телефонных: работа с электронной почтой, просмотр текстовых документов и электронных таблиц, работа с планировщиком задач и другими. Расширение функциональности телефонов возможно за счет J2ME-программ, которые поддерживаются практически всеми мобильными телефонами, смартфонами и коммуникаторами. Экран целого ряда мобильных телефонов не уступает большинству смартфонов (в последнее время телефоны оснащаются и сенсорными экранами), многие модели оснащены разъемом для карты памяти. Смартфоны в глазах данной категории пользователей выглядят более привлекательными за счет других факторов (частично обусловленных продвижением смартфонов и коммуникаторов производителями), таких как: расширенные мультимедийные функции (более качественная камера, расширенные возможности воспроизведения видеофайлов, улучшенные музыкальные способности), Wi-Fi, GPS и др.

Программы, написанные специально для операционной системы смартфона или коммуникатора, являются полноценными скомпилированными в двоичный код последовательностями низкоуровневых микропроцессорных команд. Так как во всех смартфонах установлены более мощные процессоры, чем в мобильных телефонах, возможности программиста по созданию таких приложений практически не ограничены. Специализированные приложения более функциональны и рациональней используют ресурсы процессора, чем "универсальные" J2ME-программы. Поэтому смартфоны пользуются большей популярностью среди разработчиков программного обеспечения.

Для операционной системы Symbian приложения разрабатываются на языке С++ с использованием следующих сред разработок: Visual Studio, IDE Metrowerks CodeWarrior Development Studio, Borland C++ BuilderX Mobile Edition, Carbide.C++.

В настоящее время получили большое распространение коммуникаторы и смартфоны, которые совмещают в себе функции КПК с функциями мобильного телефона. КПК обладают рядом недостатков по сравнению с коммуникаторами, основной из которых – меньшее время автономной работы. К началу 2008 года КПК были практически вытеснены смартфонами и коммуникаторами, объемы поставок КПК постоянно снижаются.

Android – недавно появившаяся платформа, основанная на Linux, разрабатываемая OHA. С использованием Android SDK и плагина для Eclipse ADT для платформы Android можно создавать приложения на языке программирования Java, управляющие устройством через разработанные компанией Google библиотеки. Также есть возможность писать приложения на C и других языках программирования. Под управлением Android работают телефоны HTC G1, Samsung i7500 и др.

Выбор платформы J2ME для разработки обусловлен ее распространенностью и возможностью доступа к необходимым ресурсам телефона.

# 2. Платформа J2ME

2.1 Основные понятия

Платформа J2ME была разработана компанией Sun Microsystems с целью предоставления возможности создания Java-приложений для устройств с ограниченными ресурсами памяти и процессора, таких как сотовые телефоны, пейджеры, смарт-карты, органайзеры и миникомпьютеры.

Платформа J2ME является безопасной для использования, потому что при доступе в интернет, использовании других коммуникаций, отправке SMS, доступе к файловой системе пользователь обязательно будет об этом уведомлен и для продолжения использования функции необходимо его подтверждение. При корректной реализации спецификаций единственным вариантом несанкционированного доступа является программа троян, которая под видом необходимости выполнения полезной пользователю функции на самом деле будет использовать полученные права в своих целях.

Архитектура платформы J2ME представлена на рисунке 2.1.

Профиль

Библиотеки

Конфигурация

JVM

Прошивка (операционная
система) телефона

Рисунок 2.1 – Архитектура платформы J2ME

Устройства, на которых сможет работать J2ME-приложение (мидлет), определяются поддерживаемой конфигурацией и профилем платформы [2]. Конфигурация определяет самые базовые классы, такие как класс System, Runtime, Thread и т.д., то есть является фундаментом платформы. Над каждой конфигурацией надстраиваются свои профили. Профиль определяет требования к аппаратной части устройства, а также может включать минимальный набор API.

Выделяют 2 конфигурации: CLDC и CDC. CDC предназначена для устройств, имеющих постоянное сетевое соединение, таких как двунаправленные пейджеры, телевизионные приставки, автомобильные системы навигации, интеллектуальные коммуникаторы. Такие устройства характеризуются более мощными системными ресурсами, например, как минимум 2 мегабайтами памяти.

Над конфигурацией CDC могут быть надстроены профили Foundation Profile, Personal Basis Profile и Personal Profile.

Foundation Profile не имеет функциональности для работы с GUI. Предназначен для встраиваемых устройств.

Personal Basis Profile содержит основные элементы GUI. Является надстройкой над Foundation Profile.

Personal Profile содержит графический интерфейс пользователя, основанный на AWT. Является надстройкой над Personal Basis Profile.

Конфигурация CLDC предназначена для устройств с ограниченным объемом памяти и вычислительной мощностью (телефоны, органайзеры, КПК). Для этих мобильных устройств характерны следующие параметры:

* процессор 16 или 32-разрядный;
* память от 160 до 512 килобайт;
* беспроводное сетевое соединение;
* питание от аккумуляторов.

Данная конфигурация содержит только базовые пакеты java.lang.\*, java.io.\*, java.util.\*, javax.microedition.io.\* и добавленный в версии 1.1 пакет java.lang.ref.\*. Пакеты, совпадающие с J2SE, содержат минимальный набор классов, необходимых для создания приложений.

По реализации пересекающихся с J2SE классов версии байт-кода CLDC 1.0 соответствует JDK 1.1, CLDC 1.1 – JDK 1.3. Иногда к названию конфигурации добавляют HI, что означает HotSpot Implementation (виртуальная машина с улучшенными алгоритмами оптимизации выполняемого кода в целом и часто выполняемых кусков кода в частности, для J2SE она стала виртуальной машиной по умолчанию с версии 1.3).

MIDP – единственный профиль для конфигурации CLDC. Он содержит пакеты для работы с графикой, звуком, взаимодействия с консолью (клавиатура и экран), базовый набор классов для отображения стандартных экранов и элементов управления. Существуют две основные версии MIDP API – 1.0 и 2.0 (самих версий MIDP больше, но в них изменения касаются безопасности и не затрагивают само API).

Профиль MIDP задает следующие технические характеристики:

* разрешение экрана минимум 96х54 пикселя;
* 32 килобайта динамической памяти;
* 128 килобайт под компоненты MIDP;
* 8 килобайт для хранения постоянных данных;
* беспроводная сеть;
* питание от аккумулятора.

Дальнейшие возможности устройств с точки зрения разработчика обуславливаются поддержкой дополнительных пакетов. Например, для работы с мультимедиа используется MMAPI (JSR-135).

Необходимой составляющей CDLC является KVM, которая из-за компактности имеет ряд недоступных свойств:

* не поддерживаются операции с дробными числами;
* нельзя создать класс-загрузчик для динамической загрузки классов во время выполнения;
* отсутствует механизм отражения – набор средств для получения полной и исчерпывающей информации о каком-либо классе во время выполнения программы;
* не реализован Java Native интерфейс – возможность обращения из Java-программ к программам, реализованным на других языках программирования, так называемым native-методам;
* не поддерживается финализация;
* отсутствует поддержка групповых потоков.

Основным классом написания приложений для мобильных устройств является MIDlet. Жизненный цикл мидлета основан на методах startApp(), pauseApp(), destroyApp() и представлен на рисунке 2.2.

приостановлен

активный

уничтожен

создание

startApp()

destroyApp()

pauseApp()

destroyApp()

Рисунок 2.2 – Жизненный цикл мидлета

Все экранные формы расширяют абстрактный класс Displayable. Этот класс описывает экранную форму, которая может иметь заголовок, бегущую строку, несколько команд и слушатель событий. Отображаемое содержимое формы определяется подклассами. Существует шесть классов, которые представляют собой различные типы экранных форм [3]:

* List позволяет выбрать пользователю элементы из списка. Каждый элемент в списке представляется строчкой и может иметь Image;
* TextBox позволяет пользователю вводить и редактировать текст;
* Alert можно сравнить с диалоговым окном, оповещающим о какой-то произошедшей ситуации;
* Form может содержать комбинацию элементов, которые могут представлять строчки, изображения, поля ввода, списки;
* Canvas позволяет приложению самому обеспечивать отрисовку, переопределив метод paint (Graphics g);
* TextBox – представляет собой многострочное поле для ввода и редактирования текста.

Подсветка экрана, вибратор, звук в MIDP 1.0 отсутствуют (они появились только в MIDP 2.0). Однако, не смотря на все эти ограничения, можно наблюдать огромное количество мобильных телефонов, в которых есть такие возможности. Происходит это потому, что производители добавляют свой собственный API, реализующий данные возможности. В связи с этим мидлет, сделанный для одного телефона, может не работать на другом.

J2ME-приложение состоит из двух файлов: jad и jar. Jad-файл – это дескриптор приложения, текстовый файл, в котором указаны атрибуты приложения. Jar-файл – это сама программа, это пакет мидлетов (Midlet Suite).

2.2 Библиотека LWUIT

Библиотека LWUIT [4] базируется на профиле MIDP 2.0 и предназначена для быстрой разработки пользовательских интерфейсов, которые будут одинаково выглядеть на телефонах различных производителей. Например, на смартфонах Nokia элемент StringItem (строковый элемент формы) ставит после себя символ перевода строки, а на смартфонах Sony Ericsson – нет. На Nokia элемент Ticker (бегущая строка) закрывает заголовок формы, на других телефонах – нет, а на третьих Ticker может вообще не поддерживаться. Такая портабельность достигается за счет реализации собственного слоя поверх системной канвы и разработанных собственных элементов графического интерфейса.

LWUIT создана по образу Swing и реализует простую графику и модель пользовательского интерфейса с небольшими начальными требованиями к ресурсам и умеренными требованиями к производительности. В ней используется интеграция графики SVG и механизм MVC.

Данная библиотека входит в состав Sun J2ME SDK 3.0 и предоставляет следующие основные возможности:

* поддержка окон;
* пользовательские стили;
* анимация;
* сенсорная навигация.

Базовым классом для всех элементов графического интерфейса является Component. Все элементы добавляются в контейнеры, прежде чем они будут отображены на экране. Контейнеры также могут быть вложены друг в друга. Самым основным контейнером является Form, который представляет собой экранную форму. В общем случае у него есть строка заголовка сверху, строка меню с командами внизу и по центру располагается ContentPane – контейнер для элементов графического интерфейса. Далее приводится листинг 2.1 создания простого приложения с использованием данной библиотеки, которое выводит приветствие и содержит одну кнопку.

Листинг 2.1 – Создание формы

import com.sun.lwuit. Display;

import com.sun.lwuit. Form;

import com.sun.lwuit. Label;

import com.sun.lwuit.layouts. BorderLayout;

import com.sun.lwuit.plaf.UIManager;

import com.sun.lwuit.util. Resources;

public class HelloMidlet extends javax.microedition.midlet.MIDlet {

public void startApp() {

 // инициализация библиотеки

Display.init(this);

 // Установка пользовательской темы

try {

Resources r = Resources.open ("/myresources.res");

UIManager.getInstance().setThemeProps (r.getTheme (

r.getThemeResourceNames() [0]));

} catch (java.io.IOException e) {

System.out.println ("IOException "+e.getMessage());

}

 // 1. Создание формы

Form mainForm = new Form ("Form Title");

 // 2. Установка LayoutManager

 // для определения способа размещения копонетов

mainForm.setLayout (new BorderLayout());

 // 3. Добавление Label в центр формы

mainForm.addComponent (BorderLayout.CENTER, new Label ("Hello World"));

 // 4. Добавление командной кнопки

mainForm.addCommand (new Command ("Run", 2));

 // 5. Вывод формы на экран

mainForm.show();

}

public void pauseApp() {

}

public void destroyApp (boolean unconditional) {

}

}

Далее рассматриваются основные элементы графического интерфейса [5].

Label может отображать строчку текста и / или картинку.

Button представляет собой кнопку и помещается на ContentPane. Может содержать рисунок.

Чтобы обработать нажатие кнопки, необходимо реализовать интерфейс ActionListener. Далее приводится программный код простого обработчика нажатия кнопки (листинг 2.2).

Листинг 2.2 – Обработка нажатия кнопки

final Button button = new Button ("Old Text");

button.addActionListener (new ActionListener() {

public void actionPerformed (ActionEvent evt) {

button.setText ("New Text");

}

});

RadioButton – радиокнопка, является наследником Button.

ButtonGroup позволяет объединять радиокнопки в группы, чтобы в любой момент была выбрана только одна из них.

CheckBox является наследником RadioButton и представляет собой флажок.

ComboBox – это выпадающий список. К нему можно применить все возможности прорисовки, как и для элементов списка List. В выпадающем списке ComboBox могут содержаться не только строчки текста, но и радиокнопки, флажки, кнопки и другие списки.

TextArea предоставляет возможность редактировать текст в текстовом редакторе телефона – это значит, что возможно использование более сложных методов ввода (например, T9).

TabbedPane – это контейнер, который позволяет переключаться между различными группами компонентов, расположенными на вкладках.

2.3 Bluetooth

Bluetooth – это недорогая технология, позволяющая электронным устройствам обмениваться информацией посредствам радиоканала. Радиус действия Bluetooth-передатчиков, как правило, не превышает 15-ти метров. Для связи используется частота 2.45 GHz. В настоящее время доступна технология Bluetooth версии 1.1, которая включает в себя технологию радиосвязи, набор программ и профилей.

Технология названа в честь короля Дании Harold Bluetooth, жившем в X веке, который объединил и правил Норвегией и Данией.

Первое Bluetooth-устройство появилось на рынке в 1999. Разработчиком Bluetooth является Bluetooth Special Interest Group. В состав этой организации входят такие компании, как Sony Ericsson, Intel, IBM, Toshiba, Nokia, Microsoft, 3COM, Motorola.

Цель спецификации [6] протокола Bluetooth – обеспечить взаимодействие устройств различных производителей, поэтому она определяет целый стек протоколов, а не только радиосистему (рисунок 2.3).

Applications

TCS

OBEX

WAP

SDP

RFCOMM

Logical Link Control and Adaption Protocol (L2CAP)

Host Controller Interface (HCI)

Baseband / Link Controller

Link Manager Protocol

Radio

Рисунок 2.3 – Стек протоколов Bluetooth

HCI – это обычно слой, разделяющий программную и аппаратную реализацию стека протокола. Слои ниже the HCI, как правило, реализованы аппаратно, а слои выше – программно. Однако устройства с ограниченными ресурсами могут полностью реализовать всю функциональность аппаратно.

Applications – профили Bluetooth, которые определяют, как приложения должны использовать стек протоколов.

Telephony Control System (TCS) – обеспечивает телефонный сервис.

Service Discovery Protocol (SDP) – используется для поиска сервисов на удаленном Bluetooth-устройстве.

WAP и OBEX – предоставляют интерфейсы для верхнего слоя других протоколов связи.

RFCOMM – описывает RS-232 как последовательный интерфейс.

L2CAP – собирает данные от верхних слоев и преобразовывает их между пакетами разных размеров.

HCI – управляет связью между хостом и Bluetooth-модулем.

Link Manager Protocol – контролирует и конфигурирует связи с другими устройствами.

Baseband и Link Controller – управляют физическими связями, частотой хопов и сборкой пакетов.

Radio – модулирует и демодулирует данные для передачи и приема по воздуху.

Технология Bluetooth [7] использует технический прием FHSS. Радиодиапазон делится на 79 подканалов. В каждый момент времени используется только один из этих подканалов. Переключение между ними (хоп) происходит каждые 625 микросекунд (1600 хопов за секунду). Такое частое переключение необходимо, чтобы уменьшить интерференцию, вызываемую другими рядом расположенными Bluetooth-устройствами и другими устройствами, использующими тот же самый диапазон частот (например, микроволновая печь или Wi-Fi).

Если устройство взаимодействует через Bluetooth и Wi-Fi одновременно, то применяется механизм AFH.

Каждому Bluetooth-устройству назначается уникальный 48-битный адрес, который используется для идентификации и синхронизации хопов, а также для генерации ключей в процедурах безопасности Bluetooth.

Пикосеть – это форма Bluetooth-сети (рисунок 2.4). Она состоит из одного мастера и нескольких (не более 7) подчиненных устройств (slaves).

A

E

D

C

B

Master

Slave

Slave

Slave

Slave

Рисунок 2.4 – Пикосеть

Устройство, инициирующее соединение, автоматически становится мастером. Подчиненные устройства могут только предавать данные в выделенное им время. Все взаимодействия между подчиненными устройствами осуществляются не напрямую, а через мастера. При подключении нового устройства к пикосети возможен обмен ролями.

Две пикосети могут существовать в пределах радиодиапазона друг друга. Частота хопов не синхронизируется между пикосетями, поэтому всегда будут возникать конфликты на одной и той же частоте.

Две взаимодействующие пикосети образуют рассеянную сеть – scatternet (рисунок 2.5). Общий узел является разделяемым и в каждый момент времени может работать только в одной пикосети, что уменьшает трафик в два раза между ними.

A

B

C

F

D

Master

Slave

Slave

Slave

Slave

G

E

Master

Рисунок 2.5 – Взаимодействие двух пикосетей

В процессе поиска [8] рядом находящихся Bluetooth-устройств (inquiry) запрашивающее устройство будет получать Bluetooth-адреса и время всех обнаруженных устройств. Чтобы устройство было обнаружено, необходимо у него установить соответствующий режим доступности – IAC. IAC может принимать два значения: GIAC и LIAC. GIAC используется, когда необходимо разрешить устройству быть обнаруженным всегда, а LIAC – когда устройство должно быть доступно в течении ограниченного промежутка времени.

После обнаружения устройства можно его просканировать на наличие доступных сервисов. Для этого предназначен протокол Service Discovery Protocol.

Bluetooth-устройство хранит информацию обо всех своих сервисах в специальной базе данных – Service Discovery DataBase. Каждая запись в ней содержит атрибуты, определяющие один сервис. Атрибуты определяются 16-ричным идентификаторами.

Обязательными для каждой записи являются только два атрибута ServiceRecordHandle (ID 0x0000) и ServiceClassIDList (ID 0x0001). Различные атрибуты содержат данные разных типов. Поэтому используются специальные элементы данных для хранения этих значений, которые состоят из дескриптора типа данных, определяющего тип и размер, и самих данных.

Universally Unique IDentifier (UUID) – это тип данных для идентификации сервисов, протоколов, профилей и пр. Он представляет собой 128-битное число, которое является всегда уникальным. Bluetooth-технология использует UUID различной длины. Диапазон коротких UUID зарезервирован для наиболее часто используемых сервисов, протоколов и профилей.

Bluetooth-профиль предоставляет набор высокоуровневых процедур и унифицированных способов использования верхних слоев Bluetooth. Профиль определяет способ реализации функциональности с применением Bluetooth.

Безопасность при использовании Bluetooth играет важную роль, потому что передаваемые беспроводным способом данные могут быть прослушаны рядом находящимися устройствами, а также устройство, которое доступно для обнаружения, может быть подвержено сканированию на наличие сервисов и их последующему использованию.

Для решения указанных выше проблем спецификация Bluetooth определяет модель безопасности, состоящую из трех компонентов: аутентификация, шифрование, авторизация. К тому же определены три режима безопасности, которые входят в профиль GAP, поддерживаемый всеми сертифицированными Bluetooth-устройствами:

* незащищенный;
* безопасность на уровне сервисов – используется большинством Bluetooth-устройств. Необходимость защиты решается на уровне сервиса. Меры защиты инициируются более высокими слоями после того, как Bluetooth-связь уже установлена более низкими слоями;
* безопасность на уровне установления связи – инициируется более низкими слоями в стеке протоколов Bluetooth. Этот режим возможен только для Bluetooth-устройств с заводскими предустановленными настройками, которые не могут быть изменены пользователем.

На более мощных устройствах с Bluetooth пользователь может сам определять уровень безопасности.

Сопряжение, или связывание (bonding), – это процедура аутентификации Bluetooth-устройствами друг друга, зависящая от разделяемого ими аутентификационного ключа. Если такого ключа нет у них, то новый ключ должен быть создан до завершения сопряжения.

Генерация аутентификационного ключа называется парингом (pairing). Паринг включает генерацию ключа инициализации и аутентификационного ключа, после чего выполняется взаимная аутентификация.

Ключ инициализации основан на ключе доступа (введенным пользователем случайном числе) и Bluetooth-адресе одного из устройств. Ключ доступа становится известным только двум данным устройствам. Аутентификационный ключ основан на случайном числе и Bluetooth-адресах обоих устройств. Ключ инициализации используется для шифрования при обмене данными для создания аутентификационного ключа и потом отбрасывается.

По завершении процесса паринга устройства являются аутентифицированными друг другом. Оба устройства используют один и тот же аутентификационный ключ. Устройства далее считаются сопряженными и могут аутентифицировать друг друга через bonding без ключа доступа.

Устройства могут оставаться в паринге до тех пор, пока одно из них не потребует установления нового паринга или пока не будет удален аутентификационный ключ на одном из устройств.

Сохранение ключа аутентификации выгодно при частом установлении соединения между данными устройствами.

После того, как два устройства аутентифицировали друг друга, шифрование может быть необходимо для данного соединения любому из этих устройств.

До применения шифрования устройства должны договориться о режиме шифрования и длине ключа.

Возможны три режима шифрования:

* отсутствие шифрования – если одно из устройств не поддерживает шифрование, то будет соединение без шифрования;
* шифрование пакетов между устройствами и широковещательных пакетов;
* шифрование только пакетов между устройствами.

До применения шифрования устройства должны договориться о размере ключа (8 – 128 бит).

Авторизация – это процесс предоставления удаленному Bluetooth-устройству доступа к некоторому сервису. Чтобы быть авторизированным, удаленное устройство вначале должно быть сначала аутентифицировано через bonding.

Доступ может быть предоставлен на время или постоянно.

При авторизации для некоторого устройства может быть установлен атрибут trust, который ассоциирует права авторизации с некоторым устройством. Устройство с таким атрибутом (доверенное устройство) может подключаться к Bluetooth-сервису и процесс авторизации будет успешен без вмешательства пользователя.

Чтобы хранить информацию о доверенных устройствах и различных уровнях авторизации для различных сервисов, применяются две базы данных – одна для устройств, другая для сервисов. Некоторым слоям необходим доступ к этим базам, который предоставляет менеджер безопасности.

Другие важные задачи, решаемые менеджером безопасности, – это запрос ключа доступа во время процесса паринга и запрос авторизации, когда удаленное устройство пытается соединиться с сервисом, требующим авторизации, а также предоставление интерфейса для настройки параметров безопасности.

# 3. Проектирование приложения для мобильного телефона на основе платформы J2ME

3.1 Постановка задачи

Требуется разработать приложение для мобильного телефона на платформе J2ME, которое будет осуществлять видеонаблюдение и предоставлять следующие основные возможности:

* расписание работы камеры – возможность указывать время начала и окончания видеонаблюдения;
* обнаружение движения и отправка SMS-сообщения или MMS-сообщения с соответствующим изображением;
* оповещение о возникающих ошибочных ситуациях на указанный номер;
* файловый браузер для выбора или создания папки, в которую требуется сохранять видеоданные от камеры как последовательность изображений;
* передача видеоданных, получаемых от камеры, в виде отдельных кадров по Bluetooth.

Для передачи данных по Bluetooth требуется разработать второй мидлет, который будет принимать данные, отображать полученное изображение и при необходимости сохранять его в заранее выбранную папку.

3.2 Построение интерфейса

Графический интерфейс приложения реализован с помощью библиотеки LWUIT. Стиль оформления всех использованных элементов интерфейса был настроен вручную. Основные принципы настройки стиля следующие.

Во-первых, можно изменить стиль одного конкретного компонента. Стиль каждого элемента определяется объектом класса Style. Чтобы получить стиль компонента, необходимо вызвать метод getStyle(). С помощью метода setStyle() можно изменить стиль данного компонента.

Во-вторых, можно изменить стиль всех компонентов одного типа, например, кнопок. Для этого необходимо получить объект UIManager.getInstance(), который определяет стиль оформления всех элементов по умолчанию. Далее следует использовать его метод setComponentStyle с двумя аргументами – имя класса компонента и объект класса Style. В листинге 3.1 приведен программный код настройки стиля текстового поля по умолчанию.

Листинг 3.1 – Применение стиля оформления для текстового поля

/\*\*

\* Configure TextField Style

\*

\* @param manager – User Interface Manager

\*/

private void configureTextField (UIManager manager) {

Style textFieldStyle = new Style();

textFieldStyle.setFgSelectionColor(0x0000);

textFieldStyle.setFgColor(0x000000);

textFieldStyle.setBgTransparency(0x00);

textFieldStyle.setFont(boldFont);

textFieldStyle.setBorder (Border.createLineBorder(1));

manager.setComponentStyle ("TextField", textFieldStyle);

}

На рисунке 3.1 показана экранная форма с основными элементами и уже примененными стилями.

Рисунок 3.1 – Заставка приложения

При создании экрана заставки (рисунок 3.1) было необходимо выравнивание по центру. В библиотеке LWUIT нет такого способа размещения компонентов, поэтому был создан класс CenterLayout (листинг 3.2), расширяющий класс Layout. Layout является абстрактным классом, который наследуют все остальные классы-менеджеры размещения компонентов контейнера. В новом классе были переопределены методы layoutContainer (Container parent) и getPreferredSize (Container parent).

Листинг 3.2 – Выравнивание элементов по центру

/\*\*

\* Layout the given parent container children

\* @param parent the given parent container

\*/

public class CenterLayout extends Layout {

public void layoutContainer (Container parent) {

int components = parent.getComponentCount();

Style parentStyle = parent.getStyle();

int centerPos = parent.getLayoutWidth() / 2 +

parentStyle.getMargin (Component.LEFT);

int y = parentStyle.getMargin (Component.TOP);

for (int iter = 0; iter < components; iter++) {

Component current = parent.getComponentAt(iter);

Dimension d = current.getPreferredSize();

current.setSize(d);

current.setX (centerPos – d.getWidth() / 2);

Style currentStyle = current.getStyle();

y += currentStyle.getMargin (Component.TOP);

current.setY(y);

y += d.getHeight() + currentStyle.getMargin (Component.BOTTOM);

}

}

/\*\*

\* Returns the container preferred size

\*

\* @param parent the parent container

\* @return the container preferred size

\*/

public Dimension getPreferredSize (Container parent) {

int components = parent.getComponentCount();

Style parentStyle = parent.getStyle();

int height = parentStyle.getMargin (Component.TOP)

+ parentStyle.getMargin (Component.BOTTOM);

int marginX = parentStyle.getMargin (Component.RIGHT)

+ parentStyle.getMargin (Component.LEFT);

int width = marginX;

for (int iter = 0; iter < components; iter++) {

Component current = parent.getComponentAt(iter);

Dimension d = current.getPreferredSize();

Style currentStyle = current.getStyle();

width = Math.max (d.getWidth() + marginX

+ currentStyle.getMargin (Component.RIGHT) +

currentStyle.getMargin (Component.LEFT), width);

height += currentStyle.getMargin (Component.TOP) + d.getHeight() +

currentStyle.getMargin (Component.BOTTOM);

}

Dimension size = new Dimension (width, height);

return size;

}

}

В следующем листинге 3.3 содержится программный код экранной формы-заставки с применением разработанного класса-менеджера размещения элементов CenterLayout.

Листинг 3.3 – Экранная форма заставки

public class SplashScreen extends Form {

public SplashScreen() {

setLayout (new CenterLayout());

setTitle ("Welcome");

addComponent (new Label(("Video observer")));

addComponent (new Label(("by M. Luskanova")));

}

}

Рисунок 3.2 – Главное меню

Для создания главного меню (рисунок 3.2) был реализован специальный класс, который представляет собой один пункт меню. Каждый элемент такого меню является контейнером с несколькими строчками и представляет их как единое целое. Такая функциональность достигается за счет наследования от класса Container и переопределения методов getComponentAt() и pointerPressed() (листинг 3.4).

Метод getComponentAt (int x, int y) вызывается, когда необходимо вернуть компонент, находящийся в заданной точке экрана. Если этот метод не переопределить, то будет возвращен элемент, входящий в данный контейнер. Переопределение второго метода гарантирует, что не произойдет никакой обработки (например, выделения) в случае, если пользователь выберет какой-либо элемент данного пункта.

Метод pointerPressed (int x, int y) выполняется в случае нажатия сенсорного указателя. Если этот метод не переопределить, то будет выделен компонент внутри данного контейнера.

Листинг 3.4 – Пункт главного меню

public class MylistItem extends Container implements FocusListener {

private Style selectedStyle =

UIManager.getInstance().getComponentStyle ("\_selected");

private Style unselectedStyle =

UIManager.getInstance().getComponentStyle ("\_unselected");

/\*\*

\* This method provides performance of

\* included components as one

\*/

public Component getComponentAt (int x, int y) {

return this;

}

/\*\*

\* Do nothing when user clicks on any element

\*/

public void pointerPressed (int x, int y) {

}

/\*\*

\* Returns "\_unselected" style by default

\*/

protected String getUIID() {

return "\_unselected";

}

/\*\*

\* Change style for selected menu item

\*/

public void focusGained (Component cmp) {

cmp.setStyle(selectedStyle);

}

/\*\*

\* Change style for unselected menu item

\*/

public void focusLost (Component cmp) {

cmp.setStyle(unselectedStyle);

}

}

Метод getUIID() переопределен для того, чтобы возвращать стиль невыделенного элемента. Метод focusGained() вызывается при получении данным пунктом меню фокуса, и поэтому требуется этот пункт отрисовать другим стилем.

При перемещении фокуса на следующий пункт меню нужно поменять стиль этого пункта на обычный, что и выполняется в переопределенном методе focusLost().

Само меню создается следующим образом с использованием вышеуказанного класса (листинг 3.5). Preferences.getInstanse() – это объект, который хранит все сделанные пользователем настройки.

Листинг 3.5 – Создание главного меню

Preferences pr = Preferences.getInstanse();

setLayout (new BoxLayout (BoxLayout.Y\_AXIS));

getContentPane().setIsScrollVisible(false);

MylistItem mli = new MylistItem (MainMenuConstatns.SCHEDULE, "Schedule");

if (! pr.isStartRecordFromUnbounded()) {

mli.addString ("from" +

pr.getStartRecordFromCalendar().get (java.util. Calendar.YEAR) +"." +

pr.getStartRecordFromCalendar().get (java.util. Calendar.MONTH) +"." +

pr.getStartRecordFromCalendar().get (java.util. Calendar.DAY\_OF\_MONTH)

+" "+pr.getStartRecordFromCalendar().get (java.util. Calendar.HOUR\_OF\_DAY)

+":"+pr.getStartRecordFromCalendar().get (java.util. Calendar.MINUTE));

}

if (! pr.isEndRecordAtUnbounded()) {

mli.addString ("till" +

pr.getEndsRecordAtCalendar().get (java.util. Calendar.YEAR) +"." +

pr.getEndsRecordAtCalendar().get (java.util. Calendar.MONTH) +"." +

pr.getEndsRecordAtCalendar().get (java.util. Calendar.DAY\_OF\_MONTH) + " " +

pr.getEndsRecordAtCalendar().get (java.util. Calendar.HOUR\_OF\_DAY) +":" +

pr.getEndsRecordAtCalendar().get (java.util. Calendar.MINUTE));

}

if (! pr.isStartRecordFromUnbounded() ||! pr.isEndRecordAtUnbounded()) {

if (pr.isScheduleIsActive()) {

mli.addString ("Active");

} else {

mli.addString ("Disabled");

}

}

addComponent(mli);

mli = new MylistItem (MainMenuConstatns.TRANSLATE\_VIDEO, "Translate video");

addComponent(mli);

if (pr.getBTConnectionURL()!= "") {

mli.addString ("via bluetooth to "+pr.getBTFriendlyName());

}

if (pr.getSaveVideoToFile()!= "") {

mli.addString (pr.getSaveVideoToFile());

}

if (pr.getMovNotification()!= "") {

String msgType = (pr.isSendSMS()? "sms": "mms") +" to";

addComponent (new MylistItem (MainMenuConstatns.MOVEMENT\_NOTIFICATION,

new String[] {"Movement notification", msgType

+ pr.getMovNotification()}));

} else {

addComponent (new MylistItem (MainMenuConstatns.MOVEMENT\_NOTIFICATION,

"Movement notification"));

}

if (pr.getErrNotificationNumber()!= "") {

addComponent (new MylistItem (MainMenuConstatns.ERROR\_NOTIFICATION,

new String[] {"Error Notification", pr.getErrNotificationNumber()}));

} else {

addComponent (new MylistItem (MainMenuConstatns.ERROR\_NOTIFICATION,

"Error Notification"));

}

addComponent (new MylistItem (MainMenuConstatns.CAMERA, "Camera"));

addComponent (new MylistItem (MainMenuConstatns.EXIT, "Exit"));

addComponent (new MylistItem (MainMenuConstatns.HELP, "Help"));

Обработка нажатия пользователем кнопок и определение, какая экранная форма должна быть показана далее, выполняется так, как показано в следующем листинге 3.6.

Листинг 3.6 – Обработка нажатий кнопок для главного меню

class MainMenuActionListener implements ActionListener {

public void actionPerformed (ActionEvent act) {

Component cmp = getFocused();

if (cmp instanceof MylistItem) {

int id = ((MylistItem) cmp).getId();

switch (id) {

case MainMenuConstatns.SCHEDULE:

midlet.getScheduleDisplay().show();

break;

case MainMenuConstatns.TRANSLATE\_VIDEO:

midlet.getVideoTransmitionDisplay().show();

break;

case MainMenuConstatns.MOVEMENT\_NOTIFICATION:

midlet.getMovementNotificationDisplay().show();

break;

case MainMenuConstatns.CAMERA:

midlet.getCamera().show();

midlet.getCamera().startCamera();

midlet.getCamera().revalidate();

break;

case MainMenuConstatns.EXIT:

midlet.exitMIDlet();

break;

case MainMenuConstatns.HELP:

midlet.getHelpScreen().show();

break;

case MainMenuConstatns.ERROR\_NOTIFICATION:

midlet.getErrorNotification().show();

break;

}

}

}

На форме настройки расписания (рисунок 3.3) необходимо было реализовать динамическое скрытие / отображение некоторых элементов, а именно: когда выбирается радиокнопка enter time, то должны сразу под ней отобразиться поля для ввода часов, минут, дня, месяца, года. А при выборе радиокнопки unbounded эти поля должны быть скрыты.

Рисунок 3.3 – Заполнение расписания работы камеры

Такая функциональность реализуется путем указания у каждой радиокнопки ActionListener, который отслеживает выбор радиокнопок (листинг 3.7). dateFrom и dateTo являются контейнерами типа Container и содержат все необходимые поля для указания точной даты и времени.

Листинг 3.7 – Обработка выбора радиокнопок

public void actionPerformed (ActionEvent evt) {

Component cmp = (Component) evt.getSource();

if (cmp == rb1DateFromEnter) {

if (rb1DateFromEnter.isSelected()) {

if (! getContentPane().contains(dateFrom)) {

addComponent (3, dateFrom);

revalidate();

return;

}

}

}

if (cmp == rbDateFromUnbounded) {

if (getContentPane().contains(dateFrom)) {

removeComponent(dateFrom);

revalidate();

return;

}

}

if (cmp == rbDateToEnter) {

if (rbDateToEnter.isSelected()) {

if (! getContentPane().contains(dateTo)) {

addComponent (getContentPane().getComponentCount() – 2, dateTo);

dateTo.setFocus(true);

revalidate();

return;

}

}

}

if (cmp == rbDateToUnbounded) {

if (getContentPane().contains(dateTo)) {

removeComponent(dateTo);

revalidate();

return;

}

}

}

Рисунок 3.4 – Экранная форма со списком

Для всех списков в данном приложении разработан специальный способ их отображения – класс MyListCellRenderer. По умолчанию для отображения списков применяется DefaultListCellRenderer, который просто преобразовывает все элементы к объектам класса Label. Для создания своего способа отображения элементов списка необходимо реализовать интерфейс ListCellRenderer.

Метод getListCellRendererComponent этого интерфейса должен возвращать отображаемый элемент списка из объекта, передаваемого ему в качестве параметра. Переопределение методов getListFocusComponent обеспечивает возможность настройки стилей отображения выделенного и невыделенного элемента списка.

Пример списка, реализованного с помощью MyListCellRenderer, приведен на рисунке 3.4. В листинге 3.8 приведен программный код класса, который прорисовывает список.

Листинг 3.8 – Прорисовка списка

public class MyListCellRenderer extends Container implements ListCellRenderer {

public MyListCellRenderer() {

setLayout (new BoxLayout (BoxLayout.Y\_AXIS));

}

/\*\*

\* Returns displayable list element from received object

\*/

public Component getListCellRendererComponent (List list, Object value,

int index, boolean isSelected) {

if (value instanceof String) {

setText((String) value);

} else {

setText (value.toString());

}

setFocus(isSelected);

applyStyle (this, isSelected);

return this;

}

/\*\*

\* Applies style for selected and unselected element

\*/

private void applyStyle (Component component, boolean isSelected) {

Style style = null;

if (isSelected) {

style = UIManager.getInstance().getComponentStyle ("\_selected");

} else {

style = UIManager.getInstance().getComponentStyle ("\_unselected");

}

component.setStyle(style);

}

public Component getListFocusComponent (List list) {

return null;

}

}

3.3 Передача данных по Bluetooth

API для работы с Bluetooth содержится в пакете JSR-82 [9]. Эта библиотека состоит из двух пакетов: базовое Bluetooth API и OBEX.

Для работы с Bluetooth у телефона должны быть минимум 512 килобайт памяти, а также поддерживаться CLDC.

Чтобы приложение могло использовать возможности Bluetooth, необходимо реализовать интерфейс DiscoveryListener и его методы deviceDiscovered(), inquiryCompleted(), servicesDiscovered(), serviceSearchCompleted(), используемые для обнаружения рядом находящихся Bluetooth-устройств.

Далее приводится программный код (листинг 3.9), который инициирует поиск. Вначале инициализируется переменная LocalDevice local, которая представляет собой данный телефон; потом инициализируется переменная DiscoveryAgent agent. Именно этот объект запускает поиск методом startInquiry() с параметрами DiscoveryAgent.GIAC и ссылкой на объект, который реализовал интерфейс DiscoveryListener и будет обрабатывать все события, связанные с обнаружением устройств и их сервисов. Этот метод возвращает true, если поиск устройств был успешно начат, или false в противном случае.

Переменная Vector devicesFound используется для сохранения всех обнаруженных устройств.

Листинг 3.9 – Начало поиска устройств

\* This method starts device discovering

\*/

public void doDeviceDiscovery() {

try {

local = LocalDevice.getLocalDevice();

} catch (BluetoothStateException bse) {

Log.out.println ("EXCEPTION!" + bse.toString());

}

agent = local.getDiscoveryAgent();

devicesFound = new Vector();

try {

if (! agent.startInquiry (DiscoveryAgent.GIAC, this)) {

Log.out.println (" DISCOVERING IS NOT STARTED");

}

} catch (BluetoothStateException bse) {

Log.out.println (" BluetoothStateException "+ bse.toString());

}

}

При обнаружении Bluetooth-устройства вызывается метод deviceDiscovered(), который получает в качестве параметров найденное устройство remoteDevice и его тип deviceClass.

Далее приводится программный код (листинг 3.10) этого метода. В методе выполняется добавление в список, отображаемый на экране, доступных рядом находящихся Bluetooth-устройств. Найденное устройство также добавляется в массив devicesFound.

Листинг 3.10 – Обработка события, связанного с обнаружением нового устройства

/\*\*

\* Called when a device is discovered during device discovery

\* @param remoteDevice founded device

\* @param deviceClass

public void deviceDiscovered (RemoteDevice remoteDevice,

DeviceClass deviceClass) {

try {

midlet.getSelectDeviceDisplay().getDevicesList().addItem (

remoteDevice.getFriendlyName(false));

devicesFound.addElement(remoteDevice);

} catch (IOException ex) {

Log.out.println ("ex when device has been discovered "+ex.toString());

}

}

После окончания поиска всех устройств вызывается метод inquiryCompleted(), в параметре которого содержится код результата поиска. Далее приводится этот метод (листинг 3.11).

Листинг 3.11 – Завершение поиска устройств

/\*\*

\* Called when device discovering is complete

\* @param – param discovering status

public void inquiryCompleted (int param) {

midlet.getSelectDeviceDisplay().removeCommand (midlet.getSelectDeviceDisplay().

getStopSearchDevices());

midlet.getSelectDeviceDisplay().addCommand (midlet.getSelectDeviceDisplay().

getBackSelectDevice());

switch (param) {

case DiscoveryListener.INQUIRY\_COMPLETED:

if (devicesFound.size() > 0) {

midlet.getSelectDeviceDisplay().removeCommand (

midlet.getSelectDeviceDisplay().getStopSearchDevices());

midlet.getSelectDeviceDisplay().addCommand (

midlet.getSelectDeviceDisplay().getBackSelectDevice());

doServiceSearch((RemoteDevice) devicesFound.elementAt(0));

} else

break;

case DiscoveryListener.INQUIRY\_ERROR:

 //Error during inquiry

break;

case DiscoveryListener.INQUIRY\_TERMINATED:

 // Inquiry terminated by agent.cancelInquiry()

break;

}

}

Рисунок 3.5 – Список найденных устройств

После поиска устройств (на рисунке 3.5 показан список обнаруженных устройств) начинается поиск сервисов на одном из них. Найденные сервисы запоминаются в массиве ServiceRecord[] servicesFound. В следующем листинге 3.12 приводится метод, который начинает поиск сервисов. Для начала поиска вызывается метод searchServices() ранее созданного объекта DiscoveryAgent agent, которому передаются следующие параметры: атрибуты искомого сервиса или его UUID, удаленное устройство, на котором производить поиск, а также ссылка на объект, который будет обрабатывать все события, связанные с обнаружением сервисов.

Листинг 3.12 – Инициация поиска сервисов

/\*\*

\* This method starts service search on the divice

\* @param device to be scanned for services

public void doServiceSearch (RemoteDevice device) {

\* Service search will always give the default attributes:

\* ServiceRecordHandle (0x0000), ServiceClassIDList (0x0001),

\* ServiceRecordState (0x0002), ServiceID (0x0003) and

\* ProtocolDescriptorList (0x004).

\* These hex-values must be supplied through an int array

\*/

int[] attributes = null; // {0x100};

/\*

\* Supplying UUIDs in an UUID array enables searching for

\* specific services.

UUID[] uuids = new UUID[1];

uuids[0] = new UUID(0x0003);

try {

agent.searchServices (attributes, uuids, device, this);

} catch (BluetoothStateException e) {

Log.out.println ("BluetoothStateException error" + e.getMessage());

}

}

Когда поиск сервисов завершен, вызывается метод serviceSearchCompleted(), код которого приведен ниже (листинг 3.13). В параметре данного метода содержится код результата поиска сервисов.

Листинг 3.13 – Завершение поиска сервисов

/\*\*

\* Called when service search completes

\* @param transID identifies a particular service search

\* @param respCode indicates why the service search is ended

\*/

public void serviceSearchCompleted (int transID, int respCode) {

switch (respCode) {

case DiscoveryListener.SERVICE\_SEARCH\_COMPLETED:

Log.out.println ("serviceSearchCompleted: SERVICE\_SEARCH\_COMPLETED");

if (servicesFound.length > 0) {

midlet.getSelectDeviceDisplay().addCommand (

midlet.getSelectDeviceDisplay().getSelectDeviceCommand());

}

/\*

\* The service to connect to has been found earlier

\* (by service discovery) and the service record is

\* referencedthrough the object named: service (of type

\* ServiceRecord)

\*/

break;

case DiscoveryListener.SERVICE\_SEARCH\_DEVICE\_NOT\_REACHABLE:

Log.out.println ("serviceSearchCompleted: SERVICE\_SEARCH\_DEVICE\_NOT\_REACHABLE");

break;

case DiscoveryListener.SERVICE\_SEARCH\_ERROR:

Log.out.println ("serviceSearchCompleted: SERVICE\_SEARCH\_ERROR");

break;

case DiscoveryListener.SERVICE\_SEARCH\_NO\_RECORDS:

Log.out.println ("serviceSearchCompleted: SERVICE\_SEARCH\_NO\_RECORDS");

break;

case DiscoveryListener.SERVICE\_SEARCH\_TERMINATED:

Log.out.println ("serviceSearchCompleted: SERVICE\_SEARCH\_TERMINATED");

break;

}

}

Когда среди обнаруженных устройств выбрано одно и выполнен на нем поиск сервисов, то можно осуществить передачу данных по Bluetooth. Передача осуществляется через StreamConnection, URL которого получается с помощью следующего метода (листинг 3.14).

Листинг 3.14 – Получение URL для доступа к сервису

public String getServiceConnectionURL (int nom) {

if (nom < servicesFound.length) {

return servicesFound[nom].getConnectionURL (

ServiceRecord.NOAUTHENTICATE\_NOENCRYPT,

false);

} else {

return null;

}

}

Следующий листинг 3.15 содержит код, который выполняет передачу данных. Метод sendImageViaBluetooth получает массив байт, который представляет собой изображение.

Листинг 3.15 – Отправка изображения по Bluetooth

public static void sendImageViaBluetooth (byte[] data) {

OutputStream btOutStr = null;

try {

StreamConnection conn = null;

String connectionURL = Preferences.getInstanse().getBTConnectionURL();

if (connectionURL == "") {

Log.out.println ("connectionURL is empty");

return;

}

try {

conn = (StreamConnection) Connector.open(connectionURL);

} catch (IOException e) {

Log.out.println (" sendImageViaBluetooth" + e.toString());

}

btOutStr = conn.openOutputStream();

btOutStr.write(data);

btOutStr.close();

} catch (IOException ex) {

Log.out.println ("sending image via BT failed (IOException)" + ex.toString());

} catch (Throwable th) {

Log.out.println ("sending image via BT failed" + th.toString());

} finally {

try {

btOutStr.close();

} catch (IOException ex) {

Log.out.println ("IOException while closing stream" + ex.toString());

}

}

}

Прием и отображение полученного кадра выполняется аналогичным способом на другом телефоне. Соответствующий код представлен в приложении А.

На следующем рисунке 3.6 приведен пример получения клиентом изображения по Bluetooth и отображения его на форме.

Рисунок 3.6 – Передача данных по Bluetooth

3.4 Работа с файловой системой телефона

В конфигурации CLDC отсутствует API для работы с файловой системой и внешними картами памяти телефона, т. к. не у каждого устройства с профилем MIDP есть файловая система.

Платформа J2SE включает в себя java.io. File и сопутствующие классы, но эти пакеты слишком тяжеловесны для мобильных устройств. Для работы с файлами в мобильных устройствах разработан пакет JSR-75 [10], который предоставляет простой и легковесный FileConnection API для работы с файловой системой. FileConnection API позволяет приложениям создавать, выполнять чтение и запись в файлы и директории, размещенные на телефоне или карте памяти. Основные классы и интерфейсы пакета JSR-75 следующие:

* интерфейс FileConnection – для доступа к файлам и директориям;
* интерфейс FileSystemListener – для прослушивания извещения о добавлении или удалении корневых директорий;
* класс FileSystemRegistry – центральный реестр для слушателей о добавлении или удалении корневых файловых систем;
* класс ConnectionClosedException – исключение, которое выбрасывается в случае невозможности выполнения методов FileConnection по причине закрытия соединения;
* класс IllegalModeException – исключение, которое выбрасывается в случае выполнения методов, требующих особых режимов безопасности, таких как READ или WRITE, но FileConnection открыт в другом режиме.

Приложение может открыть соединение, используя метод Connector.open(). Его единственный параметр – URL, содержащий полный путь к файлу специального формата: file:// <host>/<root>/<directory>/<directory>/…/<fileName>. Элемент host может быть пустым.

Список доступных корневых директорий устройства можно получить с помощью вызова метода FileSystemRegistry.listRoots().

Установив соединение FileConnection с файлом или каталогом, можно выполнять следующие действия:

* определять, куда указывает соединение FileConnection – на файл или папку – с помощью метода isDirectory();
* получать фильтрованный список файлов и директорий методом list (String filter, boolean includeHidden);
* проверять, существует ли директория или файл, применяя метод exists();
* определять, является ли скрытым файл, используя метод isHidden();
* создавать, удалять файл или директорию методами create(), mkdir() и delete();
* открывать входной / выходной поток для чтения / записи данных в файл, применяя методы openInputStream(), openOutputStream().

Листинг разработанного навигатора по файловой системе мобильного телефона представлен в приложении Б.

На рисунке 3.7 приведен пример отображения одного их каталогов файловой системы телефона.

Рисунок 3.7 – Файловый броузер

## 3.5 Отправка SMS и MMS-сообщений

Для отправки сообщений с мобильного телефона было использовано Wireless Messaging API 2.0 (пакет javax.wireless.messaging) [11]. Основные классы этого пакета следующие:

* MessageConnection – интерфейс, который определяет операции для посылки и получения сообщений;
* TextMessage – интерфейс, представляющий текстовые сообщения;
* MultipartMessage – интерфейс для представления составных сообщений;
* MessagePart – экземпляры данного класса могут быть добавлены в MultipartMessage. Каждый MessagePart состоит из данных.

Для отправки сообщений приложение должно сначала получить экземпляр MessageConnection, передав методу javax.microedition.io. Connector.open() URL, определяющий адресат. В этом URL содержится информация об используемом протоколе (SMS или MMS), номере телефона и порте адресата.

После создания MessageConnection формируется SMS или MMS-сообщение, заполняется данными и отправляется на указанный номер (листинги 3.16 и 3.17).

Листинг 3.16 – Отправка SMS-сообщений

\* Send SMS with specified text to phoneNumber

public static void sendSMS (String phoneNumber, String text) {

try {

String addr = "sms:// " + phoneNumber+":5678";

MessageConnection conn = (MessageConnection) Connector.open(addr);

TextMessage msg = (TextMessage) conn.newMessage (MessageConnection.TEXT\_MESSAGE);

msg.setPayloadText(text);

conn.send(msg);

conn.close();

} catch (IOException ex) {

Log.out.println ("IOException at Utilities.sendSMS()" + ex.toString());

}

}

Листинг 3.17 – Отправка MMS-сообщений

\* Send MMS with specified image in second parameter to phoneNumber

\*/

public static void sendMMS (String phoneNumber, byte[] data) {

try {

private static MessageConnection conn;

private static MultipartMessage msg;

String addr = "mms:// " + phoneNumber +":5432";

conn = (MessageConnection) Connector.open(addr);

msg = (MultipartMessage) conn.newMessage (MessageConnection.MULTIPART\_MESSAGE);

 // size of the part and the number of bytes loaded to mms

int partSize = 10000, counter = 0;

byte buf[] = new byte[partSize];

for (int i = 0; i < data.length / partSize && counter<=290000; i++) {

int curPartLength = partSize;

 // if the number of left bytes in data[] is lenss then part size

if (counter + partSize > data.length) {

buf = new byte [data.length – counter];

curPartLength = data.length – counter;

}

 // load next part of mms

for (int j = 0; j < curPartLength; j++) {

buf[j] = data [counter++];

}

MessagePart p = new MessagePart (

buf,

Preferences.getInstanse().getEncodingType(),

String.valueOf(i), null, null);

msg.addMessagePart(p);

}

} catch (SizeExceededException ex) {

Log.out.print ("SizeExceededException "+ex.toString());

} catch (Exception e) {

Log.out.print ("Exception "+e.toString());

} finally {

try {

conn.send(msg);

conn.close();

} catch (IOException ex) {

Log.out.print ("IOException "+ex.toString());

}

}

}

Для указания номера, на который необходимо отправлять сообщения с уведомлениями об обнаруженном движении или о возможных ошибках в работе, разработаны две формы (рисунок 3.8).

Рисунок 3.8 – Настройка SMS и MMS-уведомлений

## 3.6 Алгоритм обнаружения движения

Алгоритм обнаружения движения основан на сравнении последовательных кадров, которые получаются от камеры с использованием Mobile Media API [12] следующим образом (листинг 3.18).

Листинг 3.18 – Доступ к камере

 // get access to camera

player = Manager.createPlayer ("capture://video");

player.realize();

videoControl = (VideoControl) getPlayer().getControl ("VideoControl");

videoControl.setDisplayFullScreen(true);

byte[] b = new byte[5000];

 // get snapshot

b = videoControl.getSnapshot(encodings);

im = Image.createImage (b, 0, b.length);

rgbIm = new RGBImage(im);

 // call method for motion detection

int r = process (rgbIm.getRGB());

На рисунке 3.9 приведена диаграмма деятельности, которая описывает процесс видеонаблюдения.

Функция обнаружения движения получает в качестве параметра очередной кадр камеры, который представляет собой одномерный массив. Каждый элемент массива – это одна точка изображения в формате AARRGGBB, который обозначает следующее:

* AA – прозрачность (0xFF означает полную непрозрачность, а 0 – полную прозрачность);
* RR, GG, BB – соответственно красная, зеленая, голубая rgb-составляющие цвета данной точки.

Обнаружение движения выполняется в несколько этапов. На первом сравниваются два последовательных кадра и формируется маска. Для этого подсчитывается средняя прозрачность всех точек двух сравниваемых кадров и определяется их разность (коррекция). Далее происходит сравнение соответствующих точек этих кадров и вычисляются разности rgb-составляющих цветов этих точек (листинг 3.19).

Эти разности затем корректируются, а именно: присваивается 0, если данная разность меньше коррекции, в противном случае – из данной разности вычитают коррекцию. Далее получают оттенок серого по скорректированным rgb-разностям, как показано в листинге 3.19 (вычисление переменной result). В результирующую маску записывается этот оттенок серого с учетом прозрачности alpha, взятой из соответствующей точки предыдущего кадра. Если в данных точках есть значительное изменение, то в маску записывается белая точка.

Листинг 3.19 – Формирование маски

 // ip is a element number in two arrays –

 // current image (inDataInt) and the previous one (refDataInt)

int alpha = refData[ip] & 0xFF000000;

refDataInt = (refData[ip] & 0xFF0000) / 256 / 256;

inDataInt = (inData[ip] & 0xFF0000) / 256 / 256;

r = (refDataInt > inDataInt)? refDataInt – inDataInt: inDataInt – refDataInt;

refDataInt = (refData[ip] & 0x00FF00) / 256;

inDataInt = (inData[ip] & 0x00FF00) / 256;

g = (refDataInt > inDataInt)? refDataInt – inDataInt: inDataInt – refDataInt;

refDataInt = (refData[ip] & 0x0000FF);

inDataInt = (inData [ip++] & 0x0000FF);

b = (refDataInt > inDataInt)? refDataInt – inDataInt: inDataInt – refDataInt;

 // intensity normalization

r -= (r < correction)? r: correction;

g -= (g < correction)? g: correction;

b -= (b < correction)? b: correction;

result = (byte) (java.lang. Math.sqrt((double) ((r \* r) + (g \* g) + (b \* b)) / 3.0));

 // bwData is a mask

if (result > (byte) threshold) {

bwData [op++] = alpha + 0xFFFFFF;

} else {

bwData [op++] = alpha + result;

}

Далее анализируется полученная маска и подсчитывается количество областей, где рядом находящиеся точки красного цвета. На рисунке 3.10 показаны рядом находящиеся точки (отмечены символом "v") около текущей (обведена толстой линией). Если количество таких областей больше некоторого порога, то движение было обнаружено.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | v | v | v |  |  |  |  |
|  |  |  | v | v | v |  |  |  |  |
|  |  |  | v | v | v |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Рисунок 3.10 – Анализ маски

Полный листинг метода обнаружения движения приведен в приложении В.

Рисунок 3.9 – Диаграмма деятельности для процесса видеонаблюдения

# Заключение

В дипломной работе освещены теоретические основы платформы J2ME, архитектуры Bluetooth и обработка видеоданных для обнаружения движения. Разработано приложение на платформе J2ME, которое позволяет осуществлять видеонаблюдение. В нем используются такие ресурсы мобильного телефона как файловая система, камера, Bluetooth, отправка SMS и MMS-сообщений. Изучена библиотека LWUIT, применяемая для построения графического интерфейса.

Приложение состоит из двух мидлетов, устанавливаемых на разных телефонах, один из которых выполняет видеонаблюдение и передает данные по Bluetooth, а второй эти данные принимает.

Для первого мидлета предусмотрена возможность отправки на указанный номер SMS и MMS-сообщений с уведомлением о движении, а также сообщений о возможных ошибках. Приложение имеет встроенный броузер файловой системы и позволяет выбирать папку для сохранения данных камеры в виде отдельных изображений. Включение камеры для обнаружения движения и ее отключение может быть выполнено по заранее определенному расписанию.

Второй мидлет принимает изображения, показывает их и сохраняет на телефоне.

Интерфейс данного приложения реализован с помощью библиотеки LWUIT. Для отправки SMS и MMS-сообщений с мобильного телефона использовалось Wireless Messaging API 2.0 (JSR-205). Работа с файловой системой построена на основе FileConnection API (JSR-75). Передача данных по Bluetooth реализована с помощью пакета JSR-82. Доступ к камере выполнен с помощью MobileMedia API (JSR-135).

Полученные результаты докладывались на XII Республиканской научной конференции студентов и аспирантов "Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях". Опубликованы тезисы доклада [13].

# Список использованных источников

1. Официальный сайт компании Sun Microsystems. Платформа Java 2 Micro Edition [Электронный ресурс] / Режим доступа: http://java.sun.com/javame/reference/apis.jsp. – Дата доступа: 25.05.2009.
2. Горнаков С.Г. Программирование мобильных телефонов на Java 2 Micro Edition / С.Г. Горнаков. – М.: ДМК Пресс, 2004. – 336 с.
3. Пирумян B.В. Платформа программирования J2ME для портативных устройств / B.В. Пирумян Пер. с англ. – М.: КУДИЦ-ОБРАЗ, 2002. – 352 с.
4. Статьи разработчиков LWUIT [Электронный ресурс] / Режим доступа: http://lwuit.blogspot.com/ – Дата доступа: 25.05.2009.
5. Официальный сайт библиотеки LWUIT Edition [Электронный ресурс] / Режим доступа: https://lwuit.dev.java.net/ – Дата доступа: 25.05.2009;
6. Архипкин В.Я. Bluetooth. Технические требования. Практическая реализация / В.Я. Архипкин, А.В. Архипкин. – М.: Мобильные коммуникации, 2004. – 203 с.
7. Вишневский В.М Широкополосные беспроводные сети передачи информации / В.М Вишневский, А.И. Ляхов, С.Л. Портной. – М.: Эко-Трендз, 2005. – 592 с.
8. Andre N. Klingsheim, J2ME Bluetooth Programming, 2004.
9. Спецификация Bluetooth API and OBEX API (JSR-82) for Java Platform, Micro Edition.
10. Спецификация FileConnection Optional Package (JSR-75) for Java 2 Micro Edition Version 1.0.
11. Спецификация Wireless Messaging API (JSR-205) for Java 2 Micro Edition Version 1.1.
12. Спецификация Mobile Media API (JSR-135) for Java 2 Micro Edition.
13. Лусканова М.В. Организация видеонаблюдения с помощью мобильного телефона **/** М.В. Лусканова, Е.А. Ружицкая // "Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях", XII Республиканская научная конференция студентов и аспирантов (2009, Гомель)