Mobile Systeme
Grundlagen und Anwendungen standortbezogener Dienste

Location Based Services in the Context of Web 2.0

Department of Informatics - MIN Faculty - University of Hamburg
Lecture Summer Term 2007

Dr. Thilo Horstmann

CLDC

NMEA

MIDP

Google Earth

OpenGIS

SQL

KML

Bluetooth

Mash-Ups

Web 2.0

J2ME

Loxodrome

Euler Spaces

RDMS

GPS

PostGIS

GPX

Maps

JSR 179

Polar Coordinates

API

Threads

Today: J2ME (III)

- Introduction to J2ME Graphical User Interfaces
- The High Level GUI API (MIDP 2.0)
- Demonstration of GUI objects

Observations and implications: Display

- Display size differ significantly on various devices
  - Often it does not make sense to simply scale down a 640x480 true color screen to 101x64 with 4 colors. You need to adopt the application
  - Small display size allows for one window (at a time) only
    - No arrangement of Windows (like on Desktop)
    - Different window management
  - Display not always visible (backlight powers off to save battery power)
- Generally, is the display the right output device?
  - Speech output, event notification using sound, vibration

Observations and implications: User input

- General Limited user input capabilities
- User interaction strongly depends on available hardware
    - keyboard, stylus pen, joystick, wheel, touch screen (apple iPhone)
    - Interfaces built for some input device may not be suitable for others
- Again, does manual user input fit into the user context?
    - Speech input output
- A mobile device is not a miniaturized PC. It will be used differently!

J2ME Graphical User Interfaces

- High level API:
  - similar to AWT classes of JDK
  - high portability across devices
    - but low control of visual appearance (shape, color, or font)
  - pre defined features like scrolling and layout means built-in
  - JSR 118, implementation in package javax.microedition.lcdui
  - subclasses of Class Screen

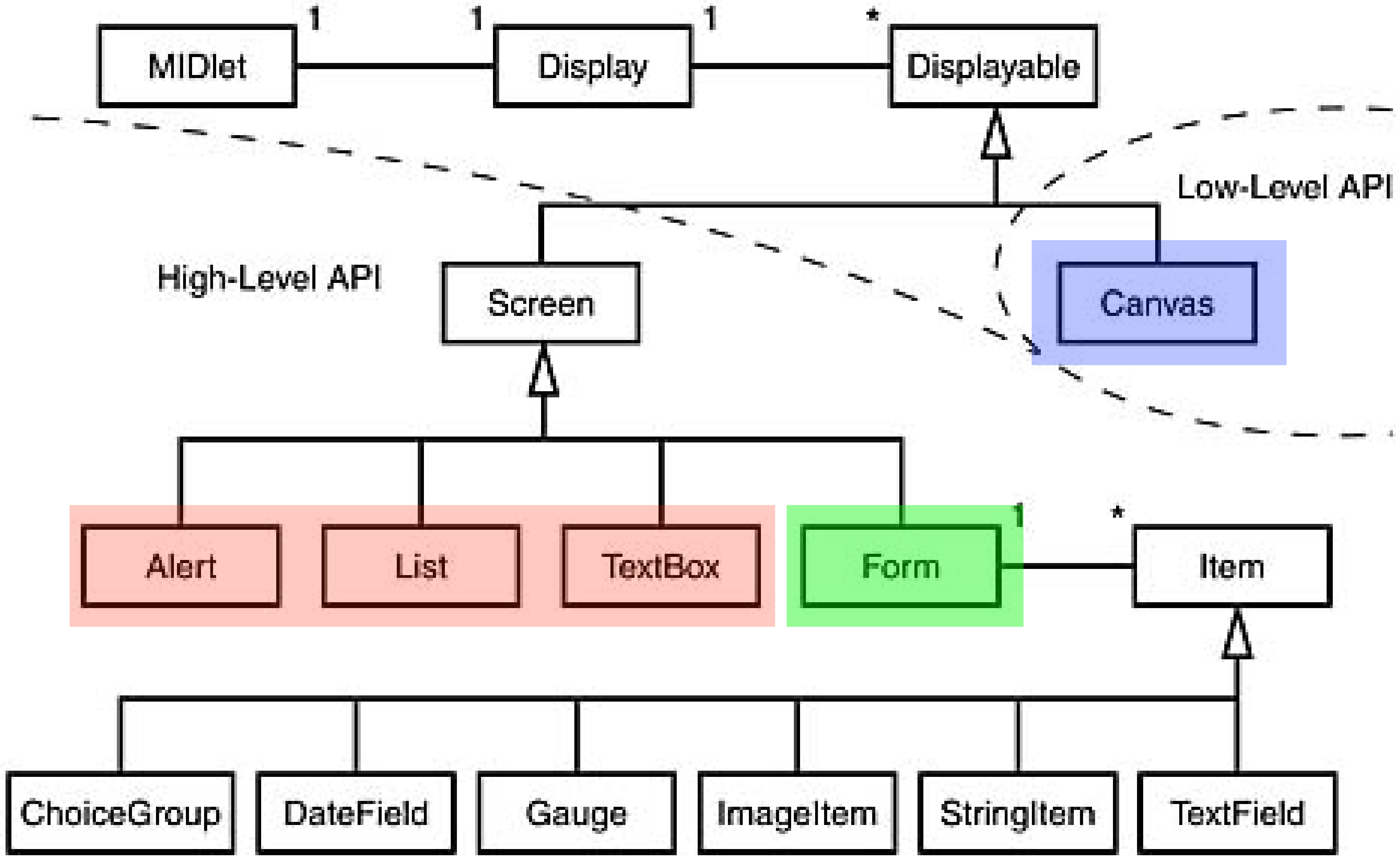
J2ME Graphical User Interfaces

- Low level API
    - Direct control of display and input device
        - e.g. key events, direct drawing of pixels
    - Higher performance but less portability
        - direct hardware access
    - Implemented by classes Canvas and Graphics
    - javax.microedition.lcdui.game
        - supports the development of mobile games and other applications withe high GUI requirements

MIDP UI model compared to AWT

- Conceptually they are very similar
- Each user interface widget is an instance of a class
- Some widgets can contain other widgets, so complex displays can be built
- Some widgets generate events, which can be handled by the application.
- The display manager is responsible for placing widgets, not the application.
- MIDP is much simpler than AWT, and immeasurably simpler than Swing.
- No explicit menu support
- No windowing: although one screen can be raised on top of another, the top screen obscures the bottom one completely.
- The developer can create new user interface widgets (Custom Items).

J2ME GUI components



The MIDP GUI Programming Model

- central abstraction is a Display
  - encapsulates device-specific graphics rendering user input. Only one screen can be visible at a time, and the user can traverse only through the items on that screen.
- There are three types of screens (Displayables):
  - Screens that encapsulate a complex user interface component that involves a List or TextBox component. The structure of these screens is predefined, and the application cannot add other components to these screens.
  - Generic screens that use a Form component. The application can add text, images, and a simple set of related UI components to the form.
  - Screens used within the context of the low-level API, such as a subclass of the Canvas class.
- The Display class is the display manager that is instantiated for each active MIDlet and provides methods to retrieve information about the device's display capabilities. A screen is made visible by calling the Display.setCurrent(...) method.

Display, Displayable, and Screen

- The ***Display*** represents the manager of the display and input devices of the system.
    - It includes methods for retrieving properties of the device and for requesting that objects be displayed on the device.
- A ***Displayable*** (abstract) has the capability of being placed on the display.
    - may have a title, a ticker, zero or more commands and a listener associated with it.
    - subclasses may be using the high-level or low-level api
- The ***Screen*** (abstract) is the common superclass of all high-level user interface classes.
    - there are no methods defined in screen

Screen Management in J2ME

- Every MIDlet belongs to exactly one Display object and vice versa
  - a Display represents a virtual Screen
  - may be visible or hidden
- a MIDlet may get the current screen after it has been started by the AMS
  - in the method startApp() call Display.getDisplay(MIDlet m)
  - not in the constructor as this may lead to undefined results!
  - call setCurrent(Displayable d) of Display object asks the AMS to replace the current Screen with d.
- You need to manage your screens yourself
  - e.g., by using a stack

Example: Simple J2ME GUI

```
public class DateTimeApp extends MIDlet {
  Alert timeAlert;
  public DateTimeApp() {
    timeAlert = new Alert("Alert!");
    timeAlert.setString(new Date().toString());
  }
  public void startApp() {
    Display.getDisplay(this).setCurrent(timeAlert);
  }
  public void pauseApp() {
  }
  public void destroyApp(boolean unconditional) {
  }
}
```

High Level Screen Objects

- Class Screen is abstract and thus cannot be instantiated
- There are 3 high level concrete screen classes that will make up a complete screen.
    - Alert, List, and TextBox
    - There is no way to add other GUI objects to such screens
- The 4th high level screen object, form, serves as a container for Items
- Every Screen can have two additional characteristics: a title and a ticker
    - inherited from Displayable
    - both are optional

High level and Low level Event Handling

- as with the GUI elements there are two kinds of events
  - high-level (such as selecting an item from a list) and
  - low-level (such as pressing a key on the device).
- The application is notified of events through callbacks
  - Implemented in Java with Interfaces (rather than passing a function pointer like in other languages C++)

High level Event Handling

- There are 2 key concepts: ***Command*** and ***CommandListener***
- The ***Command*** class encapsulates the semantic information of an action. The command itself contains only information about a command, but not the actual action that happens when a command is activated.
- A command contains four pieces of information: a short label, an optional long label, a type, and a priority. One of the labels is used for the visual representation of the command, whereas the type and the priority indicate the semantics of the command.

```
Command infoCommand = new Command("Info", Command.SCREEN, 2);
```

- The actual placement depends on the implemention and is based on the command type (“Command.SCREEN”) and priority (2).

High level event handling

- The action is defined in a CommandListener object associated with the screen
    - You (the application developer) have to implement the CommandListener Interface (typically by using a nested class or an inner class)
    - `commandAction(Command c, Displayable d)`: Indicates that a command event has occurred on Displayable d.
- if a CommandListener method does not return or the return is not delayed, the system may be blocked.
    - use threads such that the method returns quickly!
- You add commands to a displayable object with the `addCommand` method
- and register a listener with the `setCommandListener` method

Example: Commands and CommandListener

```
tb =  new TextBox("Example Text Box", null,  max, TextField.ANY);
tb.addCommand(new Command("Exit", "Exit application", Command.EXIT, 1));
tb.addCommand(new Command("Delete", "Delete text", Command.SCREEN, 2));
tb.addCommand(new Command("Help", Command.HELP, 2));

tb.setCommandListener(new CommandListener() {
   public void commandAction(Command c, Displayable d) {
      switch(c.getCommandType()) {
         case Command.EXIT:  exitApp(); break;  //notifyDestroyed()
         case Command.HELP:
            // e.g. show some text in an alert box
            break;
         case Command.SCREEN: tb.setString(null); break;
      }
    }
}); //setCommandListener
```

Let’s see same real examples (with code :-) Alert, List, TextBox

Form

- A form is also a subclass of Class Screen but serves as a Container for Items
  - it is not intended to be used on its own
  - As a subclass of a Screen it also inherits the properties and methods of a Screen (like TextBox, Alert, and List)
- It may hold one or more of the following items (i.e. any subclass of Class item)
  - Spacer, StringItem, ImageItem, ChoiceGroup, TextField, DateField, Gauge, CustomItem
- A Form comes with a high level layout manager which simplifies the platform independent arrangement of items
  - in a nutshell, items are laid out left to right in rows that stack from top to bottom, just like ordinary text (cf. API documentation of Form). Item Layout attributes can be used to control the Layout (e.g. LAYOUT_LEFT, LAYOUT_TOP, etc.).
  - However, the exact layout displayed depends on the implementation

Items

- StringItem
  - A string item displays a non-editable string and its associated label. You can change the text programmatically with the StringItem's setText method and retrieve its value via the getText method.
- ImageItem
  - Displays an image and its associated label, or an alternate string if the image cannot be displayed. The image must be immutable and can be set using setImage.
- TextField
  - This displays text in an editable field. You can specify a pre-existing value through the constructor or with the setString method. The text field can be a single or multi-line text field, although how each device handles this varies. Some allow multi-line editing directly on the containing form, while others require a separate screen. You can specify how the field will handle data entered by the user, or constraints, but devices are not required to provide all of these. These constraints provide the user some guidance for what data to enter and potentially ease data entry.

Items (cont.)

- DateField
  - Allows user input of a date and/or time value. Internally, MIDP holds the value as a java.util.Date object. You can specify an initial date or accept the default null date value. How the device handles and renders user input is device dependent and may be anything from a simple text entry to a complicated calendar or clock widget.
- Gauge
  - You use a gauge to display a continuous value over a specified range. The gauge can be interactive, letting users change the value manually, or non-interactive, appropriate for tasks such as a progress bar.
- ChoiceGroup
  - Provides a list of alternative choices to the user and allows one or more choices to be selected. ChoiceGroups can be EXCLUSIVE, like a radio group in standard desktop applications, or MULTIPLE, allowing multiple item selection. ChoiceGroups notify ItemListeners of changes when the state of a choice (selected or not selected) changes, but not when focus changes from one choice to another.

Event Handling of Items

- Every Item on the form can have its own event handlers similar to the event handling of Displayable‘s
  - Interface ItemCommandListener
- In addition, the containing form gets notified if the state of any item changes
  - requires an ItemStateListener (use setItemStateListener()) in the Form
- When returning from a screen that has been activated by an item command, use Display.setCurrentItem(item) rather than (Display.setCurrentForm()).
  - this ensures the item will be selected again

Some more examples: Forms with Items

Homework :-)

- Provide a simple GUI for the students RS of the previous lecture.
  - The first screen shows a simple Form with a TextField and a search button. After the search button has been clicked the matching Student record (lastName) will be retrieved from the record store.
  - The result form should contain a form with 2 StringItems, a DateItem and an ImageItem to display an instance of a Student object. Provide a back button to allow the user to return to the seach form.
  - If nothing was found, an alert box will be displayed.

This Lecture

- Breymann, U., Mosemann, H.: Java ME – Anwendungsentwicklung für Handys, PDA und Co., Hanser, 2006, www.java-me.de
    - Chapter 5.1, 5.2, 5.3
- User Interfaces with MIDP 2.0:
    - http://today.java.net/pub/a/today/2005/05/03/midletUI.html

Thank you!

Dr. Thilo Horstmann

e-mail: thilo.horstmann@gmail.com
blog: http://www.das-zentralorgan.de