CSE 7393
J2ME

Session 1

Overview

• The 3 flavors of java
• J2ME architecture
  – Configurations
    • CDC
    • CLDC
  – Profiles
    • Foundation
    • MIDP
Java Platforms

- Driving forces
  - Penetration in a wide variety of market segments: wrist watch, washing machine, IP router, web server, video games etc.
  - Different devices have different requirements and different expectations of Java.
  - One platform (solution) cannot address all the market segments.
  - Users/developers want flexibility. They want to choose what they want to use and what they do not.

Java Platforms

- JVM
- Java programming language
- Core & optional packages

Java2 Standard Edition (J2SE™)
Java2 Enterprise Edition (J2EE™)
Java2 Micro Edition (J2ME™)

Standard desktop & Workstation Applications
Heavy duty server systems
Small & memory Constrained devices
**J2SE**

- Feature complete Java foundation
- Client side enterprise development: stand alone applications & web applets
- 2 binary deliverables
  - JDK: Application development
  - JRE: Runtime environment
- Example: desktop or workstation

**J2EE**

- Distributed applications
  - Server side enterprise application
- Multi-tier application model
  - Middle-tier contains business logic and system services
  - Scalability, manageability, accessibility
- Examples: JDBC, Components, CGI etc.
J2ME: Multi-tier Model

First tier → EIS tier

First tier
- Client
  - Business logic

EIS tier
- Server
  - data

Middle tier
- Business logic
- Services

J2ME: Devices

- Characteristics
  - Limited memory: 128K to 2 Meg
  - Limited computational power
  - Either mobile or plug-in devices
  - Mobile → battery powered

- Requirements
  - Built-in consistency across products
  - Power of OO programming language
  - Code portability
  - Safe network delivery
  - Upward scalability (to J2EE & J2SE)
What is a J2ME Configuration?

The J2ME technology has two design centers:

1. Devices that you hold in your hand (CLDC) devices with 128-512 K of memory available for the Java technology environment and applications

2. Devices you plug into a wall. (CDC) devices with 512 K + of memory

Reference: J2ME documentation from Sun web site
J2ME Configurations

- J2ME Profile1
- J2ME Profile2
- J2ME Profile3
- J2ME Profile4
- J2ME Profile5
- J2ME Profile6

J2ME CLDC Libraries
- Java Language
- K Virtual Machine

J2ME CDC Libraries
- Java Language
- Compact Virtual Machine

128 → 512 Kbytes memory
16 → 32 bit processor

Up to 2Meg memory
32 bit processor

CLDC Vs. CDC

- CLDC
  - 160 Kbytes to 512 Kbytes of total memory available.
  - 16-bit or 32-bit processor.
  - Low power consumption and often operating with battery power.
  - Connectivity with limited bandwidth.

- CDC
  - 2Mbytes or more memory for Java platform
  - 32-bit processor
  - High bandwidth network connection, most often using TCP/IP
J2SE, CDC & CLDC

- Connection framework:
  - java.io.*
  - java.net.*
- Javax.microedition.*

CLDC Vs. CDC: JVM

- **KVM**
  - Lots of features eliminated
    - Floating point, finalize(), error handling, JNI
  - Change in classfile verification
    - Preverifier, verifier
  - Classfile format: JAR

- **CVM**
  - Slightly slimmer than J2SE JVM
  - Has the option of including all JVM features
  - AWT/Swing: unnecessary
Kilobyte Virtual Machine (KVM)

• KVM is a new smaller runtime environment for resource-constrained devices: 40–80 Kbytes – hence the “K” in KVM.

• Developed from the project called “Spotless System” to create an execution engine for the Palm PDA.

• Team discovered that the runtime environment’s size is derived mainly from its runtime libraries.

• The team removed classes that were too big or not critical to the system.

Features Eliminated:

• Java Native Interface, because native functionality is implementation dependent.

• The User define class loader, because CLDC will have a class loader that user can’t override, replace or reconfigure.

• Reflection, to eliminate RMI, serialization

• Thread groups and daemon threads but still supporting multi-threading.

• Finalization of class instances

• Weak references

• AWT Java Abstract Window Tool kit will be replaced with limited user interface classes from javax.microedition.lcdui package

• The floating-point type
CLDC Compatibility With the JVM Specification

- No Floating Point Support
- No Finalization
- Limited Error Handling
- No Java Native Interface
- No User-defined Class Loaders
- No Reflection
- Threading supported but Thread groups are not
- No Weak References
- Off-device preverification and runtime verification of classes
- Programmer can not override system classes or change the class file lookup order

CLDC Target Devices

- Connected Limited Device Configuration
- Mobile phones and Pagers which have (160 – 512 Kbytes memory). These use the Kilobyte Virtual Machine (KVM).
- Limited Bandwidth (9600 bps or less) and intermittent network connectivity
- CLDC device uses 16 or 32 bit RISC/CISC processor
- CLDC uses the profile called MIDP
What is a Profile?

1. A profile is a specification that details the built on top of APIs and utilizing the underlying configuration, necessary to provide a complete runtime environment for a specific kind of device.

2. Profiles can be thought of as selecting classes from APIs to form a complete environment.

3. Profiles are designed and integrated to meet the needs of specific industry segments.

Reference: J2ME documentation from Sun web site
Challenges for Wireless Developers

• 1. Transmission Errors
Messages sent over wireless links are exposed to interference (and varying delays) that can alter the content received by the user.

• 2. Message Latency
the time it takes to deliver a message -- is primarily affected by the nature of each system –Keep user inform of processing delay

• 3. Security
Any information transmitted over wireless links is subject to interception.

HelloWorld.java

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

public class HelloWorld extends MIDlet {
    private Form mainscreen;
    private Display myDisplay;

    HelloWorld1() {
        myDisplay = Display.getDisplay(this);
        mainscreen = new Form("Hello World");
        StringItem strItem = new StringItem("Hello", "Hello world");
        mainscreen.append(strItem);
    }

    public void startApp() throws MIDletsStateChangeException {
        myDisplay.setCurrent(mainscreen);
    }

    protected void pauseApp() { }

    protected void destroyApp(boole unconditiona) throws MIDletsStateChangeException{ }
}
J2ME UI

- MIDP provides some limited UI elements
  - Form
  - Alert
  - Choice and ChoiceGroup
  - List
  - StringItem
  - TextBox
  - TextField
  - DateField
  - Gauge
  - Ticker
J2ME UI vs J2SE UI

- MIDP UI elements ARE NOT subsets of AWT/Swing
- Interaction with the user is based around a succession of screens
- MIDP only has a single command listener

J2ME Canvas

- J2ME does allows lower level access to the Canvas
  - Can create custom graphics and user interfaces by extending Canvas
Lightweight Window Toolkit

- Motorola introduced the LWT to address the limitation of MIDP
  - Works on all java-enable mobile phone
  - Similar to J2SE Swing
    - Layout management with absolute or relative widget placement
    - Notion of containers
    - Component Listeners

Java 2 Platform
Mobile Information Device Profile

MIDP Hardware

- Memory (added to CLDC memory)
  - 128K non-volatile for MIDP components
  - 8K non-volatile for application persistent data
  - 32K volatile for KVM

- Display
  - Screen 96x54
  - Display depth 1-bit
  - Pixel shape (aspect ratio) 1:1
MIDP Hardware

• Input (one or more)
  – One-handed keypad
  – Two-handed keypad
  – Touch screen
• Networking
  – Two-way
  – Wireless
  – Possibly intermittent
  – Limited bandwidth

Software Development Process
J2ME Development Process

- Java files compiled to class files
- Class files “preverified” by the preverifier tool
  - Classes have additional attributes added for runtime verification
- Class files are placed in jars and downloaded
- Runtime verifier does a linear check on classes using preverification attributes
  - Attributes are ignored by the J2SE JVM

MIDP Application Model and MIDlets
MIDP Application Model

- MIDlet is the basic application
  - Similar to the J2SE applet
  - GUI based
- MIDlet Suites – security for applications that share resources or data

Midlets - States

- New HelloWorld()
- Paused
  - pauseApp
  - startApp
  - destroyApp
- Active
  - startApp
  - destroyApp
- Destroyed
Midlets - Suites

• Collection of MIDlets placed in a single jar file
• Share a Common
  – Name space for storage
  – Object heap
  – Static data in classes
• Installation, upgrade, removal as a single suite unit, no individual item may be changed