Informática Móvil

Diseño de aplicaciones con J2ME

WMA: Wireless Messaging API

José Ramón Arias García
Ignacio Marín Prendes

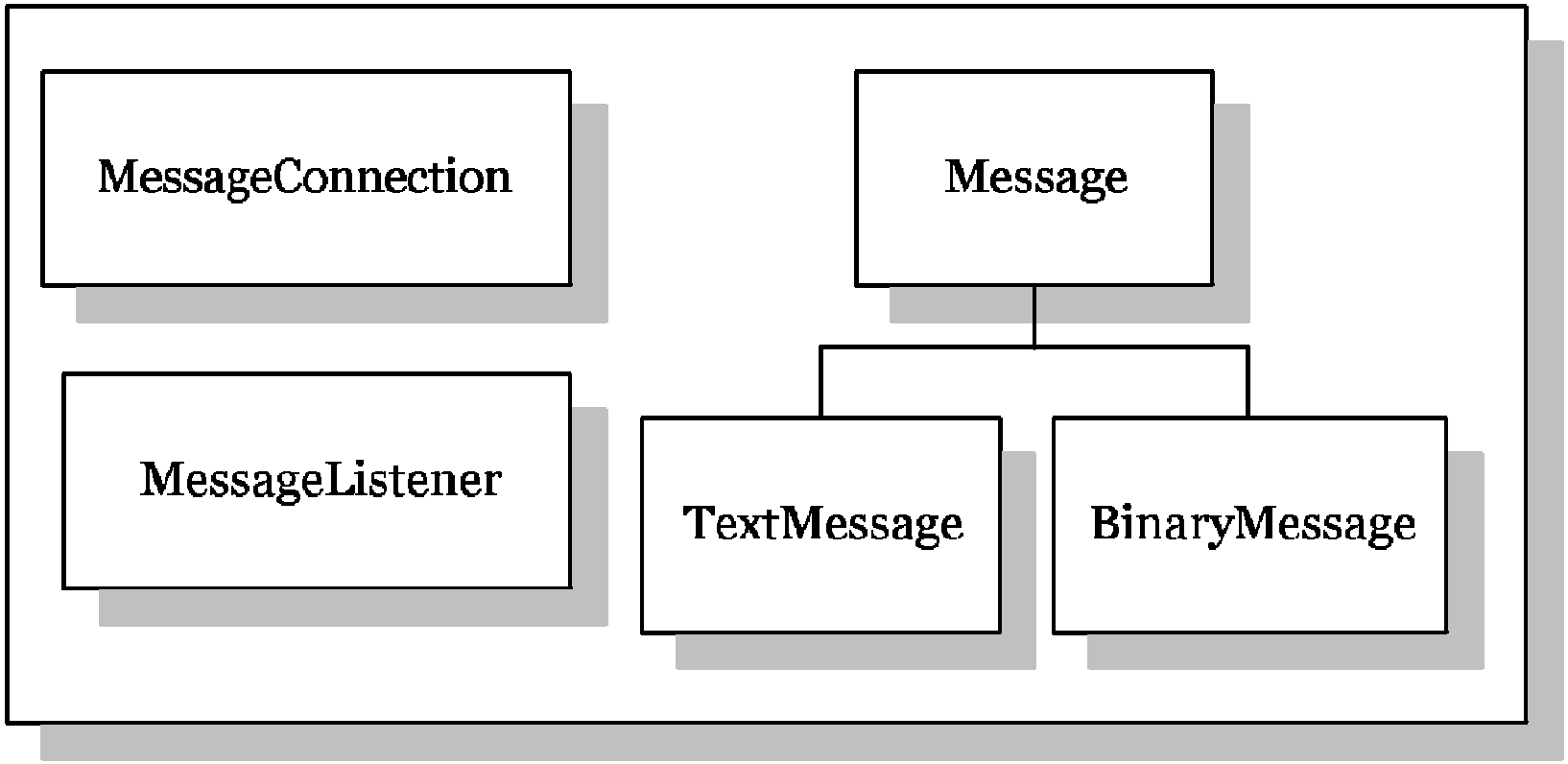
UNIVERSIDAD DE OVIEDO
Área de Arquitectura y Tecnología de Computadores

Curso 2004/2005

Introducción

- WMA (Wireless Messaging API)
  - API que proporciona a las aplicaciones MIDP la capacidad de enviar/recibir mensajes SMS
  - El API de mensajería está basado en el Generic Connection Framework (GCF) definido en CLDC 1.0
    - GCF está definido en *javax.microedition.io*
    - Soporta E/S y red en los perfiles J2ME
  - Las interfaces de la API de mensajería WMA están definidas en el paquete *javax.wireless.messaging*
    - Este paquete define todas las interfaces necesarias para enviar y recibir mensajes wireless, tanto binarios como de texto
    - *javax.wireless.messaging* es un paquete opcional
  - Actualmente, versión WMA 1.1 (2.0 en desarrollo)

Clases en *javax.wireless.messaging*



Clases en *javax.wireless.messaging*

INTERFAZ	DESCRIPCIÓN	MÉTODOS
Message	Interfaz base, de la que derivan `TextMessage` y `BinaryMessage`	`getAddress()`, `getTimestamp()`, `setAddress()`
BinaryMessage	Subinterfaz de `Message` que proporciona métodos para fijar y detectar el payload binario	`getPayloadData()`, `setPayloadData()`
TextMessage	Subinterfaz de `Message` que proporciona métodos para fijar y detectar el payload de texto	`getPayloadText()`, `setPayloadText()`
MessageConnection	Subinterfaz de la interfaz Connection de GCF, que proporciona una factoría de `Messages`, y métodos para enviar y recibir `Messages`	`newMessage()`, `receive()`, `send()`, `setMessageListener()`, `numberOfSegments()`
MessageListener	Define la interfaz listener para implementer la notificación asíncrona de objetos `Message`	`notifyIncomingMessage()`

Interfaz *Message*

- javax.wireless.messaging.Message es la interfaz base para todo tipo de mensajes comunicados mediante WMA
  - Un tipo Message es lo que se envía, recibe, produce y consume
  - En ciertos aspectos, similar a un datagrama (Datagram)
    - Tiene direcciones origen y destino
    - Tiene payload
    - Tiene formas para enviar un mensaje y bloquearse a la espera de un mensaje
  - Pero WMA proporciona funcionalidad adicional
    - Soporte para mensajes de texto y binarios
    - Interfaz listener para recepción asíncrona de mensajes
- WMA define 2 subinterfaces
  - BinaryMessage y TextMessage
  - Especificación extensible para futuros nuevos tipos de mensajes

Interfaz *Message* (II)

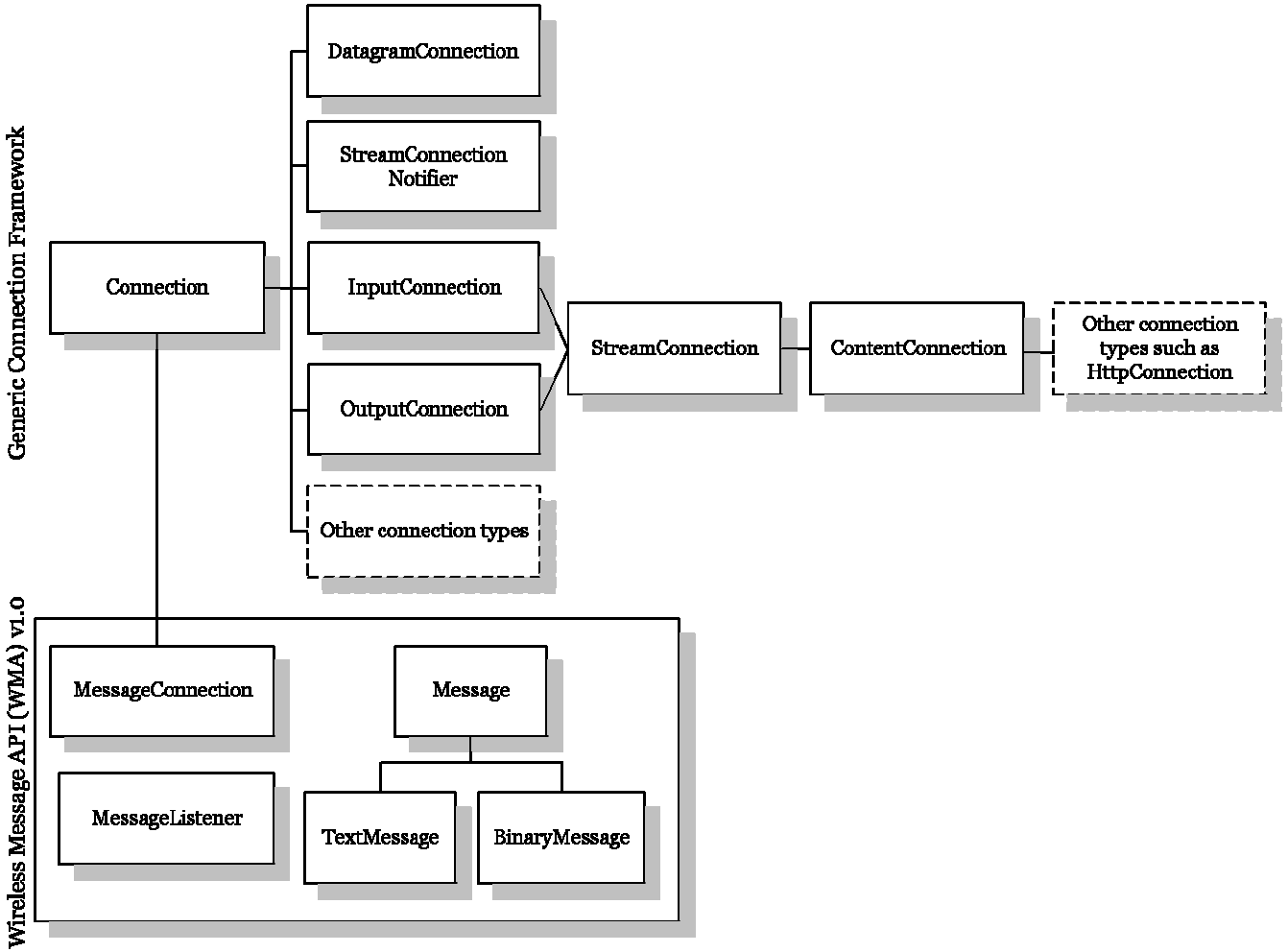
- IMPORTANTE:
  - La forma en que se codifican tanto los mensajes como la información de control asociada a ellos) es ESPECÍFICA DE PROTOCOLO y TRANSPARENTE PARA WMA.
  - Para los detalles de bajo nivel, están los anexos de la especificación WMA.

Interfaces *BinaryMessage* y *TextMessage*

- BinaryMessage
  - Representa un mensaje con payload binario
  - Declara métodos para enviar y recibir mensajes
- TextMessage
  - Representa un mensaje con payload de texto
  - Proporciona métodos para leer y escribir payloads de texto (instancias de String)
  - Antes de que el mensaje sea enviado o recibido, la implementación subyacente tiene la responsabilidad de codificar o descodificar el String a o hacia el formato apropiado (p.e., GSM de 7 bits ó UCS-2)

Interfaz *MessageConnection* (I)

- Es una subinterfaz de *javax.microedition.io.Connection*
  - Connection pertenece al GCF (Generic Connection Framework)



Interfaz *MessageConnection* (II)

- Proporciona los métodos newMessage(), send(), y receive() para crear, enviar, y recibir objetos Message
- MessageConnection.numberofSegments()
  - Determina información de segmento acerca del Message *antes* de ser enviado
    - Interesante para mensajes que han de ser partidos
- Define dos constantes String
  - BINARY_MESSAGE y TEXT_MESSAGE
  - Son pasadas al método factoría newMessage() para crear el tipo de objeto Message apropiado

Interfaz *MessageConnection* (III)

- Crear una *Connection* (en este caso, una *MessageConnection*) hay que llamar a ***javax.microedition.io.Connector.open()***
  - Puede haber múltiples *MessageConnection*s abiertos a la vez
- Para cerrarla, ***javax.microedition.Connection.close()***
- Un *MessageConnection* puede crearse como:
  - Cliente (sólo puede enviar mensajes)
  - Servidor (puede enviar y recibir mensajes)
- El tipo de *MessageConnection* se especifica mediante URL
  - Cliente: (MessageConnection)Connector.open("sms://**+5121234567:5000**");
  - Servidor: (MessageConnection)Connector.open("sms://**:5000**");

Interfaz *MessageListener*

- Implementa el patrón de diseño Listener para recibir objetos *Message* de forma asíncrona
  - Sin bloquearse mientras espera mensajes
- Define un solo método: *notifyIncomingMessage()*
  - Es invocado por la plataforma cada vez que se recibe un mensaje
  - Para registrarse, *MessageConnection.setListener()*
  - Puesto que muchas implementaciones de la plataforma son monohilo, la cantidad de procesamiento dentro de *notifyIncomingMessage()* debe ser minimizada
    - Idealmente, debería ser despachado un hilo aparte para consumir y procesar el mensaje
    - Así, la plataforma invierte el menor tiempo posible en *notifyIncomingMessage()*

Envío de mensaje de texto SMS

```
import javax.microedition.io.*;
import javax.wireless.messaging.*;
.....
try {
        String addr = "sms://616223456";
        // Se abre una connexion con la URL
        MessageConnection conn = (MessageConnection) Connector.open(addr);
        // Se crea el mensaje de texto a enviar
        TextMessage msg = (TextMessage)
                                        conn.newMessage(MessageConnection.TEXT_MESSAGE);
        msg.setPayloadText("Hola Mundo");

         // Envío del mensaje
        conn.send(msg);

        // Cierre de la conexión
        conn.close();
} catch (Exception e) {
            System.out.println("Error enviando mensaje: "+e.toString());
}
.....
```

Informática Móvil

Diseño de aplicaciones con J2ME

WMA: Wireless Messaging API

José Ramón Arias García
Ignacio Marín Prendes

UNIVERSIDAD DE OVIEDO
Área de Arquitectura y Tecnología de Computadores

Curso 2004/2005