




Tehnologii avansate de programare

Curs 13

Cristian Frăsinaru

`acf@infoiasi.ro`

Facultatea de Informatică
Universitatea "Al. I. Cuza" Iași



Platforma Java 2 Micro Edition



Cuprins

- Programare de rețea
- Comunicarea prin mesaje (WMA)
- Activarea MIDlet-urilor (Push Registry)
- Aplicații multimedia
- XML în J2ME



Programare de rețea folosind MIDP



Modele

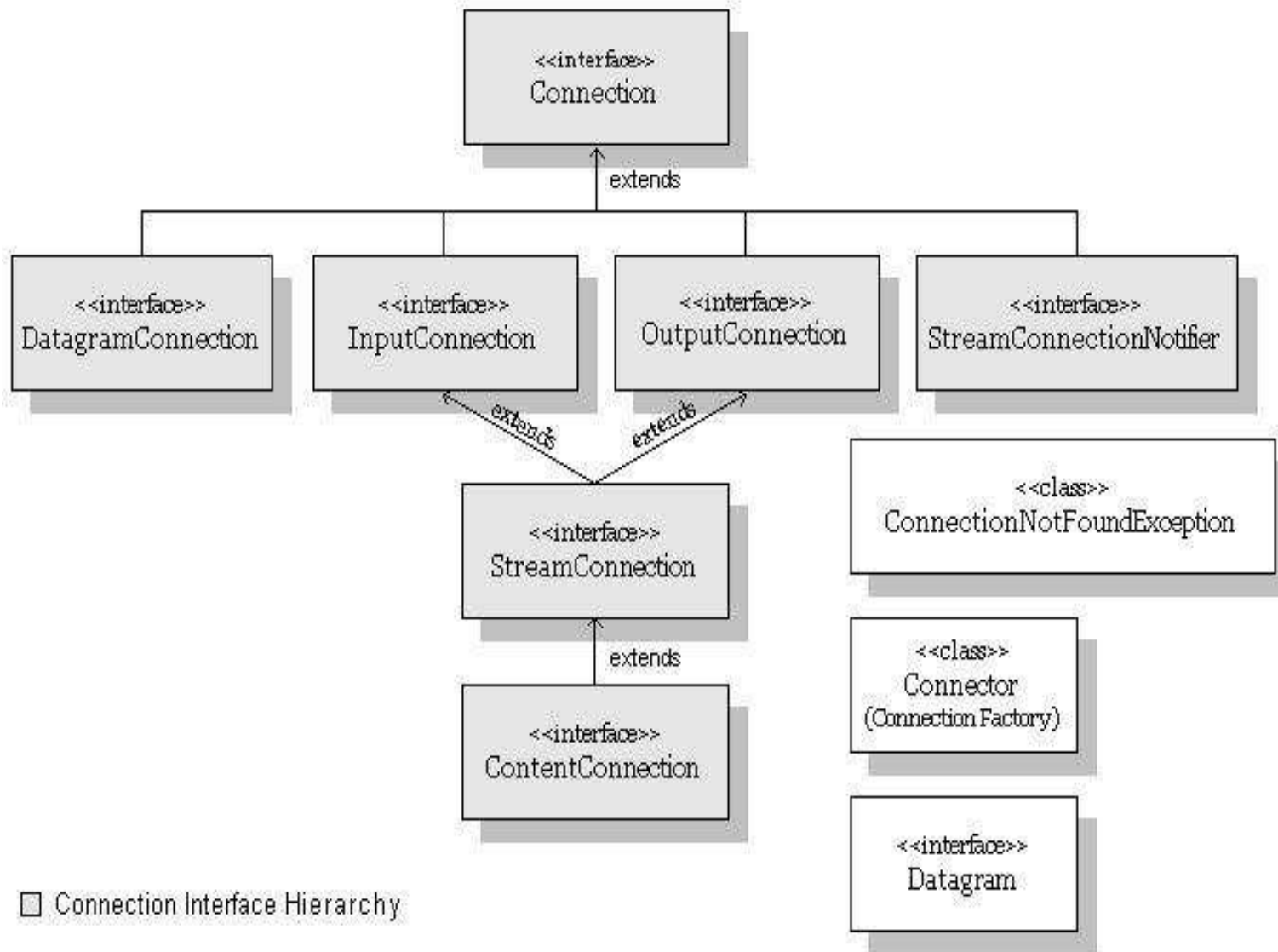
- **WAP** (Wireless Application Protocol) - protocol ce permite comunicarea cu un server Web (prin intermediul unui gateway) si transferul de informații în format WML (Wireless Markup Language).
- **i-mode (DoCoMo)** - tehnologie proprietară ce permite acces permanent la Internet de pe un telefon mobil.
- ...
- **Java** - comunicare **directă** cu orice server Web, folosind diverse protocoale de rețea (TCP, UDP, etc.)

Generic Connection Framework

GCF definește fundamentele pentru dezvoltarea aplicațiilor de rețea pentru dispozitive mobile, oferind suport pentru toate tipurile de comunicare fie pe bază de *pachete*, fie pe bază de *fluxuri*.

Inițial, modelul GFC a fost definit în configurația CLDC (Connected Limited Device Configuration), în pachetul **javax.microedition.io**.

Ierarhia claselor și interfețelor



□ Connection Interface Hierarchy

Formatul general al unui URL

protocol:adresa;parameteri

- *protocol* - metoda de acces: ftp, http, etc
- *adresa* - numele complet sau IP-ul resursei, portul, utilizator, parola, calea locală, etc.
- *parameteri* - perechi de forma ;nume=valoare.

Exemple

```
http://java.sun.com/developer  
datagram://address:port#  
comm:0;baudrate=9600  
file:/myFile.txt  
...
```

Tipuri de conexiuni

btl2cap	Bluetooth	L2CAPConnection
datagram	Datagram	DatagramConnection
file	File Access	FileConnection, InputConnection
http	HyperText Transport Protocol	HttpConnection
https	Secure HTTP	HttpsConnection
comm	Serial I/O	CommConnection
sms mms cbs	Short Messaging Service Multimedia Messaging Service Cell Broadcast SMS	MessageConnection
apdu jcrmi	Security Element	APDUConnection, JavaCardRMICConnection
socket serversocket	Socket	SocketConnection, ServerSocketConnection
datagram	UDP Datagram	UDPDatagramConnection

Crearea unei conexiuni

Se realizează cu clasa **Connector**. Aceasta este o clasă de tip *Factory*, în sensul că va instanția un obiect funcție de schema (protocolul) specificat în URL.

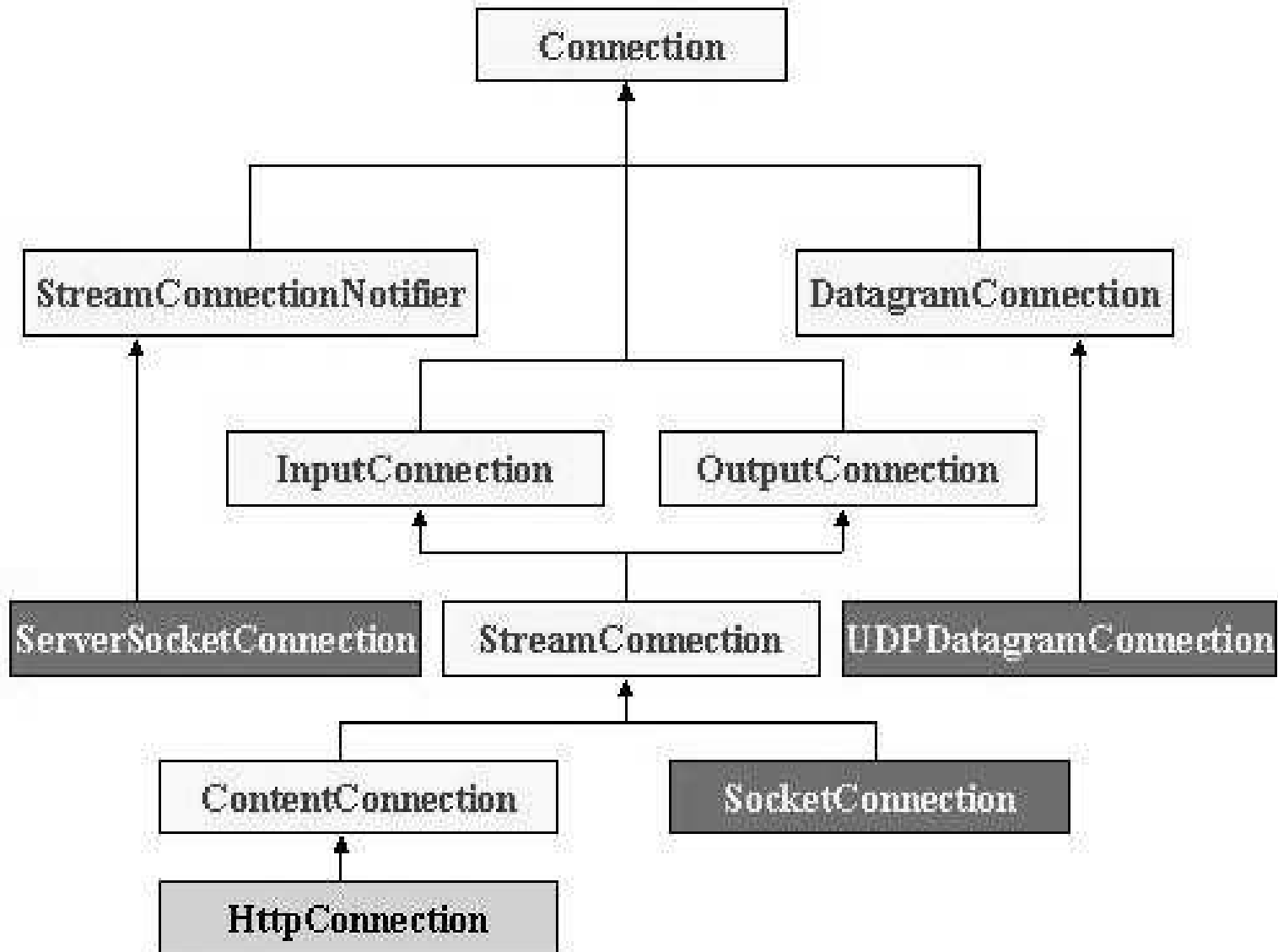
- Metoda **Connector.open**

```
String url = "socket://www.j2medeveloper.com:80";  
c = (SocketConnection)Connector.open(url);  
s = c.openInputStream();
```

- Metode de crearea directă a unor fluxuri specifice

```
String url = "socket://www.j2medeveloper.com:80";  
s = (InputStream)Connector.openInputStream(url);
```

Conexiuni suportate de MIDP



HttpConnection

HTTP este un protocol bazat pe paradigma *cerere-răspuns*, în care parametrii cererii trebuie trimiși înaintea cererii propriu-zise. Conexiunea poate exista în una din stările:

- *Setup* - In această stare pot fi apelate doar metodele `setRequestMethod` și `setRequestProperty`
- *Connected* - Metode cum ar fi **`openInputStream`**, **`openDataInputStream`** determină tranziția din starea `Setup` în starea `Connected`
- *Closed*

Citirea unui fișier

```
String url = "http://www.infoiasi.ro/~acf/hello.txt"
StreamConnection c = null;
InputStream s = null;
StringBuffer b = new StringBuffer();
try {
    c = (StreamConnection)Connector.open(url);
    s = c.openInputStream();
    int ch;
    while((ch = s.read()) != -1) {
        b.append((char) ch);
    }
    System.out.println(b.toString());
} finally {
    if(s != null)
        s.close();
    if(c != null)
        c.close();
}
```

Programare *low-level*

Incepând cu MIDP 2.0 a fost introdus suport pentru programare de rețea low-level, folosind socket-uri și datagrame. Interfețele adăugate în `javax.microedition.io` sunt:

- pentru protocolul **socket**: `SocketConnection`,
`ServerSocketConnection`

```
Connector.open("socket://host:port") -> SocketConnection  
Connector.open("socket://:port") -> ServerSocketConnection
```

- pentru protocolul **datagram**:
`UDPDatagramConnection`

```
Connector.open("datagram://host:port") -> UDPDatagramConnection
```

Interfața *ServerSocketConnection*



```
...
ServerSocketConnection server =
    (ServerSocketConnection) Connector.open("socket://:2500");
SocketConnection client =
    (SocketConnection) server.acceptAndOpen();

client.setSocketOption(DELAY, 0);
client.setSocketOption(KEEPALIVE, 0);

DataInputStream dis = client.openDataInputStream();
DataOutputStream dos = client.openDataOutputStream();

String result = is.readUTF();
os.writeUTF(...);

is.close();
os.close();
client.close();
server.close();
```

...



Interfața *SocketConnection*



...

```
SocketConnection client =  
    (SocketConnection) Connector.open("socket://" + hostname + ":" + port
```

```
client.setSocketOption(Delay, 0);  
client.setSocketOption(KEEPALIVE, 0);
```

```
InputStream is = client.openInputStream();  
OutputStream os = client.openOutputStream();
```

```
os.write("some string".getBytes());  
int c = 0;  
while((c = is.read()) != -1) {  
    // do something with the response  
}
```

```
is.close();  
os.close();  
client.close();
```

...





Wireless Messaging API (WMA)



Ce este WMA ?

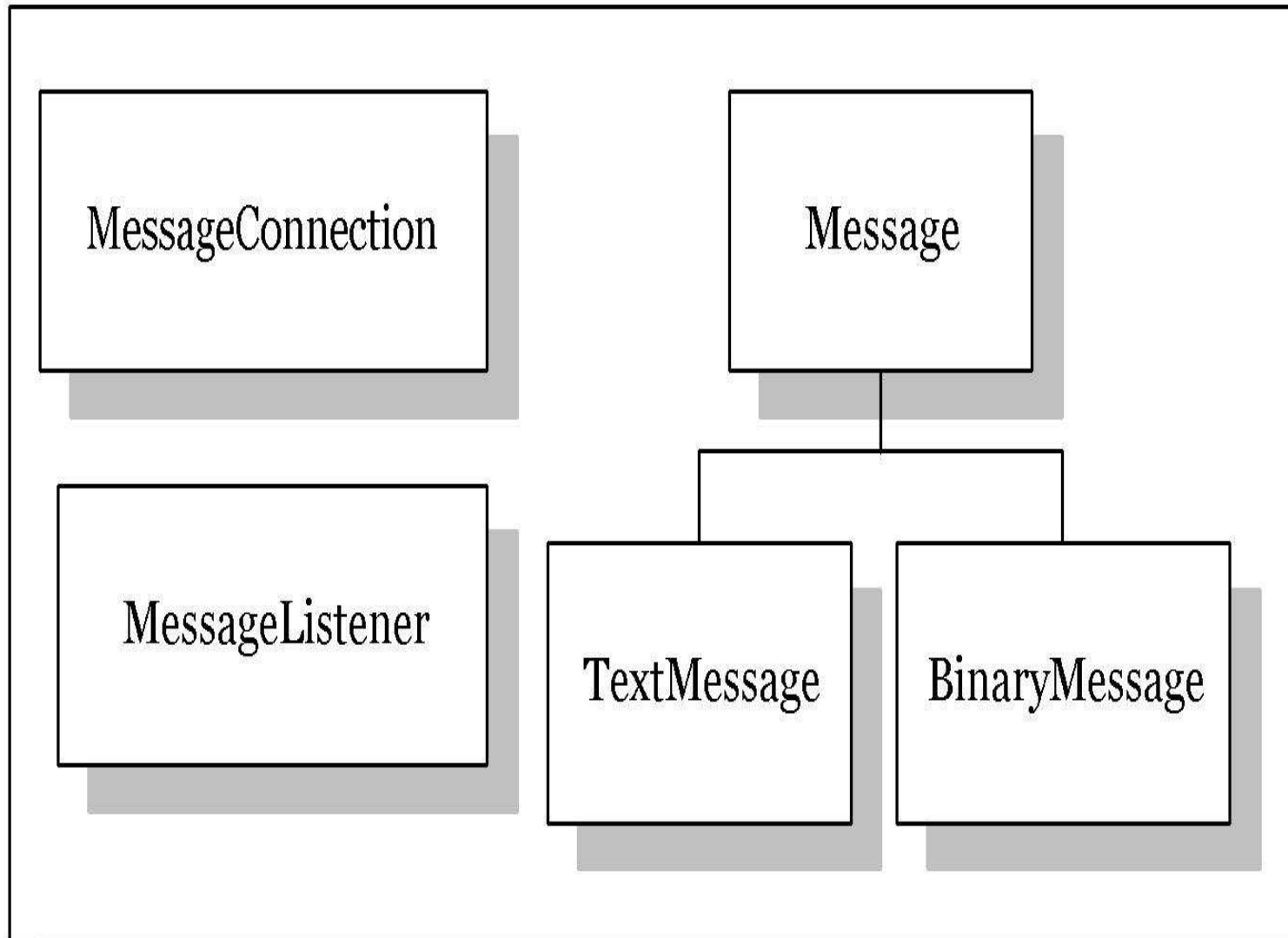


WMA oferă suport pentru **trimiterea / recepționarea de mesaje text sau binare**, cum ar fi SMS-uri (Short Message Service).

WMA este un pachet opțional (**javax.wireless.messaging**), bazat pe Generic Connection Framework (GCF) ce extinde profile create pe configurația CLDC.



Componentele WMA



WMA API

Message	<code>getAddress(), getTimestamp(), setAddress()</code>
BinaryMessage	<code>getPayloadData(), setPayloadData()</code>
TextMessage	<code>getPayloadText(), setPayloadText()</code>
MessageConnection	<code>newMessage(), receive(), send(), setMessageListener(), numberOfSegments()</code>
MessageListener	<code>notifyIncomingMessage()</code>

Crearea unei conexiuni

- Modul **client** - doar pentru a trimite mesaje

```
(MessageConnection)Connector.open("sms://+5121234567:5000");
```

- Modul **server** - pentru a trimite / recepționa mesaje

```
(MessageConnection)Connector.open("sms://:5000");
```

Tipuri de protocoale

- sms
- sms
- cbs

Trimiterea unui mesaj text

```
public void sendTextMessage(MessageConnection mc,
                            String msg,
                            String url) {
    try {
        TextMessage tmsg =
            (TextMessage)mc.newMessage(MessageConnection.TEXT_MESSAGE);
        tmsg.setAddress(url);
        tmsg.setPayloadText(msg);
        mc.send(tmsg);
    }
    catch(Exception e) {
        System.out.println("sendTextMessage " + e);
    }
}
```

Trimiterea unui mesaj binar

```
public void sendBinaryMessage(MessageConnection mc,
                              byte[] msg,
                              String url) {
    try {
        BinaryMessage bmsg =
            (BinaryMessage)mc.newMessage(MessageConnection.BINARY_MESSAGE);
        bmsg.setAddress(url);
        bmsg.setPayloadData(msg);
        mc.send(bmsg);
    }
    catch(Exception e) {
        System.out.println("sendBinaryMessage " + e);
    }
}
```

Recepționarea mesajelor

● Implementarea interfeței **MessageListener**

```
public class WMAMIDlet extends MIDlet implements MessageListener {
    ...
    public void notifyIncomingMessage(MessageConnection conn) {
        // Procesare mesaj
    }
}
```

● Inregistrarea ascultătorului

```
public void startApp() {
    ...
    MessageConnection mc = (MessageConnection)Connector.open("sms://");
    mc.setMessageListener(this);
    ...
}
```

Procesarea mesajelor

Este realizată uzual într-un fir de execuție.

```
public void run() {  
    ...  
    Message msg = mc.receive();  
  
    if (msg instanceof TextMessage) {  
        TextMessage tmsg = (TextMessage)msg;  
        String data = tmsg.getPayloadText();  
    }  
    if (msg instanceof BinaryMessage) {  
        BinaryMessage bmsg = (BinaryMessage)msg;  
        byte[] data = bmsg.getPayloadData();  
    }  
}
```

Segmentarea și reasamblarea

Segmentarea și reasamblarea (SAR) este o caracteristică a unor protocoale de transport care permite divizarea unui mesaj mare într-o secvență ordonată de mesaje de dimensiuni mai mici (unități de transmisie). Exemplu: 1 segment SMS - 160 caractere. Pot exista limitări asupra dimensiunii unui mesaj sau a numărului de segmente. Standard: 3 segmente.

```
MessageConnection mc = ... ;
TextMessage tmsg = ... ;
int segcount = mc.numberOfSegments(tmsg);
if (segcount == 0) {
    // alert the user
}
```

Bluetooth

Bluetooth reprezintă o tehnologie wireless pentru distanțe scurte, cu costuri reduse, pentru crearea de rețele personale (PAN - Personal Area Network). O astfel de rețea (*piconet*) este creată dinamic într-un spațiu restrâns în care devine automat posibilă comunicarea între diferite dispozitive mobile (telefoane, PDA-uri) - unul din ele va fi *master*, celelalte (cel mult 7), fiind de tip *slave*. De asemenea, Bluetooth permite conectarea mai multor piconet-uri.

Java APIs for Bluetooth Wireless Technology (JABWT) permite dezvoltarea de aplicații Java bazate pe tehnologia Bluetooth.

JXTA



JXTA reprezintă un set de protocoale de rețea open-source peer-to-peer (P2P) care permit comunicarea între orice tipuri de dispozitive conectate în rețea: stații PC, servere, telefoane, PDA-uri, etc.

JXTA poate fi utilizat din orice limbaj, sistem de operare, dispozitiv fizic sau protocol de transport.

Comunicarea este realizată prin mesaje XML.

JXME (JXTA - J2ME) reprezintă setul de protocoale

JXTA destinat J2ME pentru configurația CLDC și profilul MIDP.



Push Registry

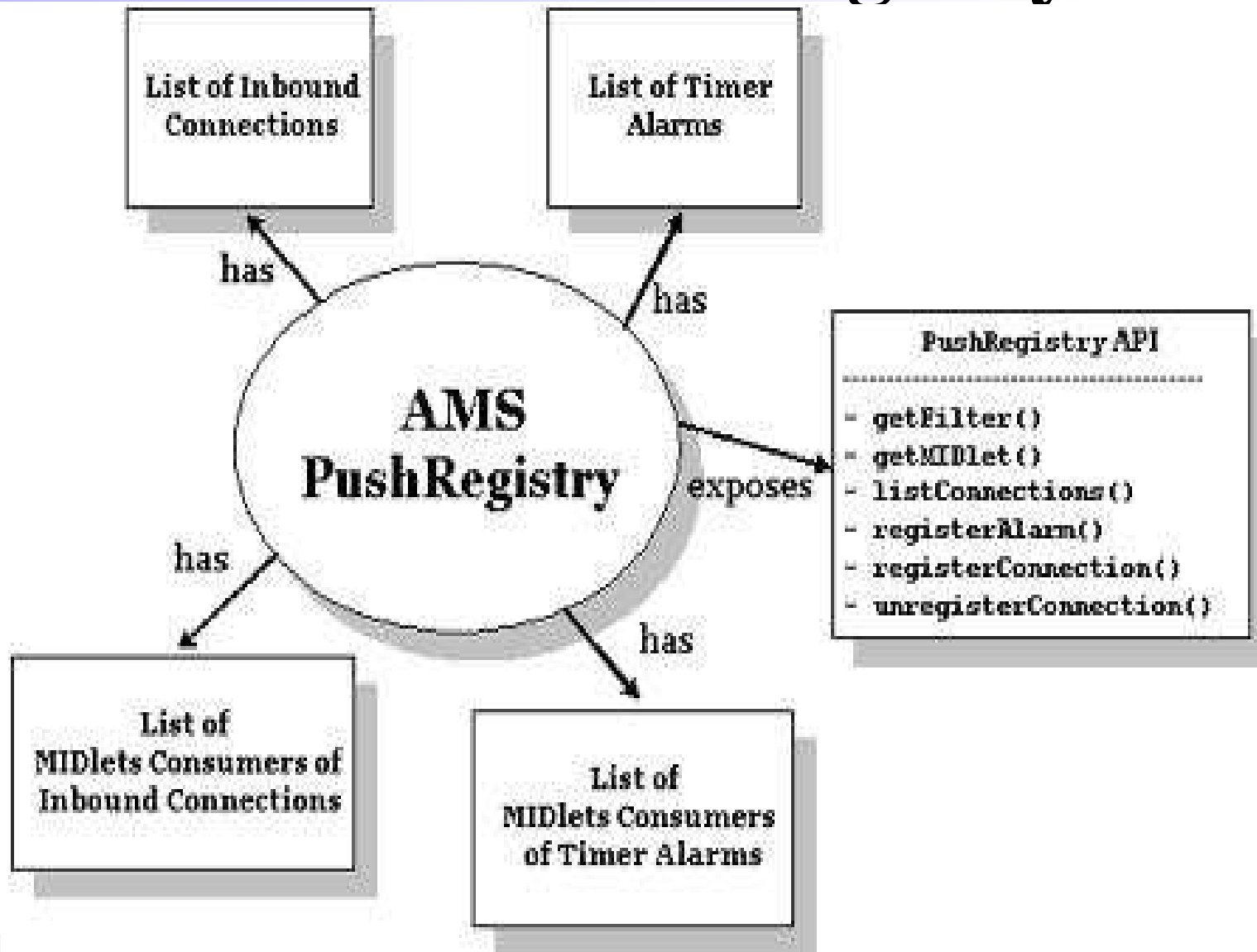
Conceptul de "Push"

Conceptul de "Push" se referă la mecanismul sau abilitatea unei aplicații de a primi și procesa asincron informații, având avantajul de a reduce substanțial utilizarea resurselor dispozitivului pe care rulează programul respectiv. Folosind regiștri de tip "Push" este posibilă **activarea midlet-urilor**:

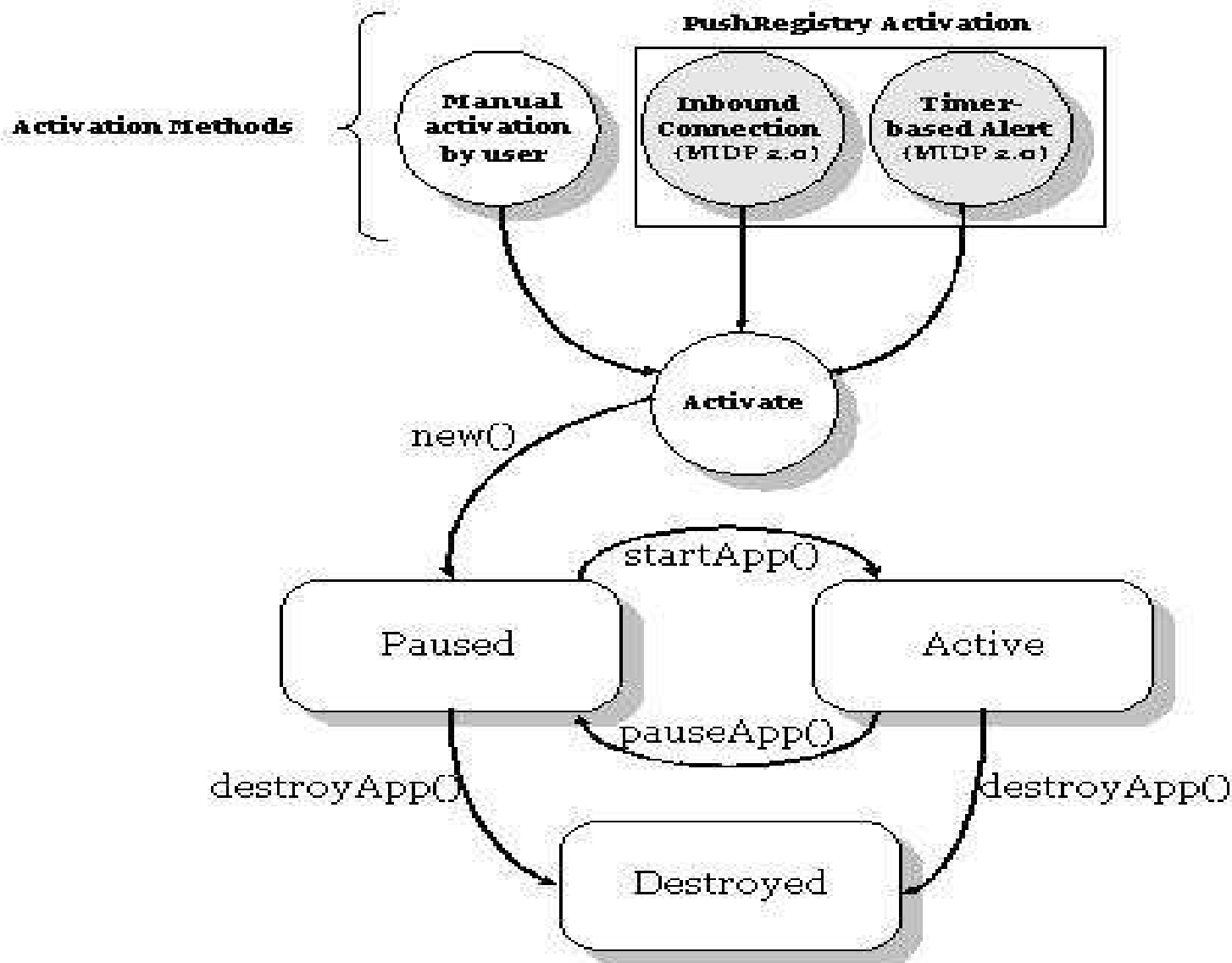
- pe baza unui timer
- prin rețea

Utilitatea: aplicații pentru e-mail, agenda, etc.

Elementele Push Registry



Activarea midlet-urilor



Inregistrarea

- **Statică** - în fișierul descriptor (.jad) al suitei de midleturi pot fi înregistrate doar activări prin rețea.

```
MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>,  
                <AllowedSender>
```

```
MIDlet-Push-1: socket://:79, com.sun.example.SampleChat, *
```

```
MIDlet-Push-2: datagram://:50000, com.sun.example.SampleChat, *
```

- **Dinamică** - folosind Push Registry API pot fi înregistrate la momentul execuției (runtime) atât activări prin rețea cât și pe bază de timer.

- `PushRegistry.registerAlarm`

- `PushRegistry.registerConnection`

Activarea pe bază de timer

```
private void scheduleMIDlet(long deltatime)
    throws ClassNotFoundException,
        ConnectionNotFoundException,
        SecurityException {

    String midlet = this.getClass().getName();
    Date alarm = new Date();
    long t = PushRegistry.registerAlarm(midlet, alarm.getTime() + deltatime);
}

...

public void destroyApp(boolean uc) throws
    MIDletStateException {
    // Eliberare resurse
    ...
    // Inregistrarea activarii
    scheduleMIDlet(defaultDeltaTime);
    display = null;
}
```

Activarea prin rețea

```
...
String midletClassName = this.getClass().getName();
String url = "socket://:5000";
String filter = "*";
try {
    ServerSocketConnection ssc =
        (ServerSocketConnection)Connector.open(url);
    PushRegistry.registerConnection(url, midletClassName, filter);
    SocketConnection sc =
        (SocketConnection)ssc.acceptAndOpen();
    InputStream is = sc.openInputStream();
    ...
}
catch(SecurityException e) { ... }
catch(ClassNotFoundException e) { ... }
catch(IOException e) { ... }
...
```



Crearea de aplicații multimedia folosind J2ME



Mobile Media API

Mobile Media API (MMAPI) este un pachet opțional pentru dezvoltarea de aplicații multimedia folosind J2ME. Implementarea standard pentru CLDC/MIDP este inclusă în J2ME Wireless Toolkit.

- `javax.microedition.media`
- `javax.microedition.control`
- `javax.microedition.protocol`

Caracteristici



- Suport pentru generarea de tonuri, redarea și înregistrarea semnalelor audio și video
- Clase specializate pentru dispozitive cu capacități limitate
- Independență de protocol sau conținut
- Extensibilitate - pot fi adăugate noi facilități, formate, tipuri de control



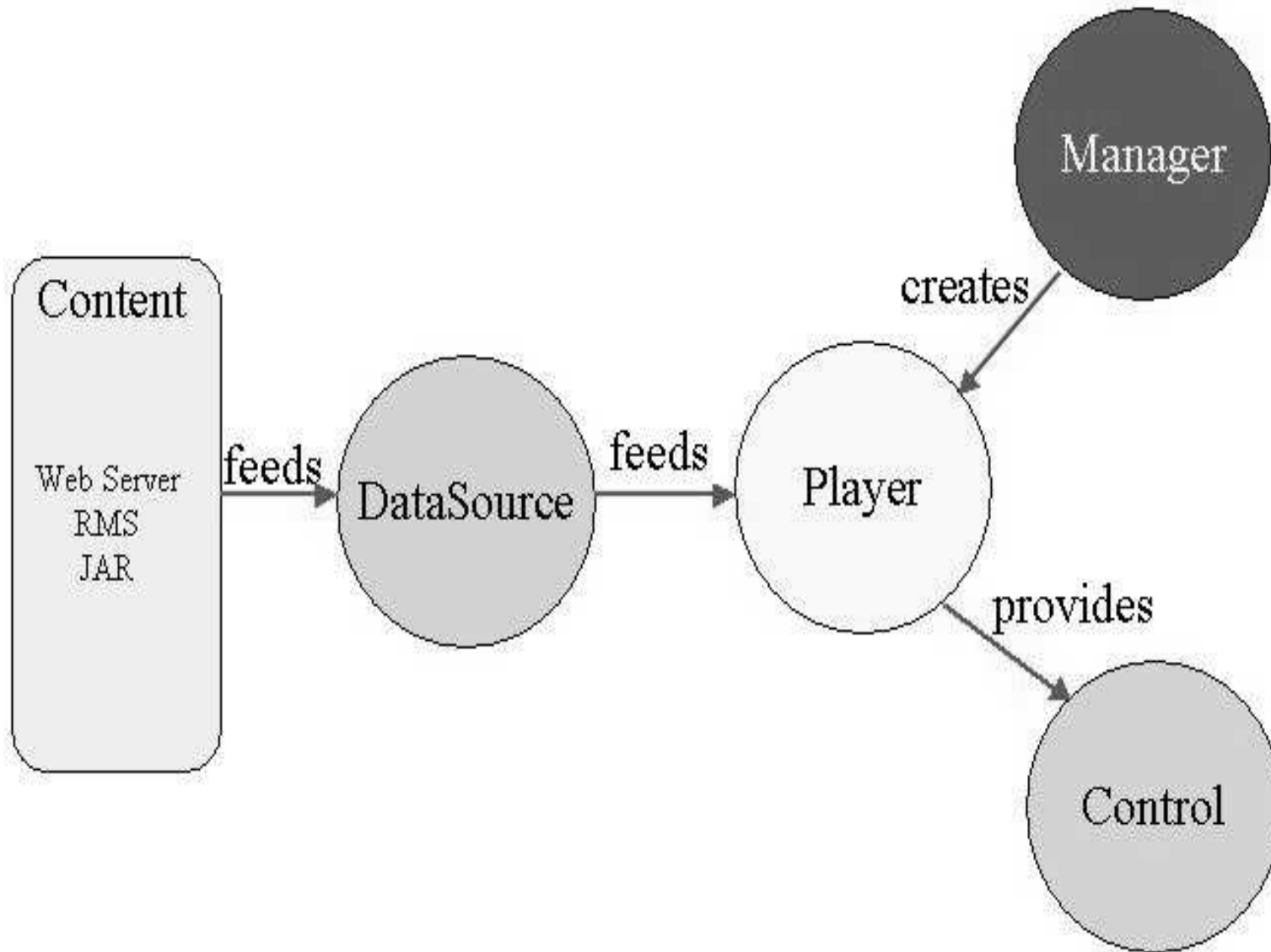
Procesare multimedia

Există două părți ale procesării multimedia:

- *La nivel de protocol* - citirea datelor dintr-o sursă (fișier, flux) într-un sistem de procesare.
Clasa responsabilă: **DataSource**.
- *La nivel de conținut* - parsarea, decodificarea și redarea informației într-un dispozitiv fizic audio sau video.
Clasa responsabilă: **Player**.

MMAPI oferă și un mecanism de tip *factory*, clasa **Manager**, ce permite crearea obiectelor `Player` din obiecte de tip `DataSource`, sau `InputStream`

Arhitectura MMAPI



Crearea unui *Player*



```
Player player = Manager.createPlayer(String url);  
//url este de forma: <protocol>:<adresa>
```

Ciclul de viață:

```
UNREALIZED -> REALIZED -> PREFETCHED <-> STARTED -> CLOSED  
      realize()      prefetch()      start()      close()
```

Tipuri de control - Pentru fiecare tip media, metoda `getControl` returnează un obiect capabil să controleze redarea: `MIDIControl`, `VolumeControl`, `VideoControl`, etc.



Redarea audio

```
...
Player p;
VolumeControl vc;
try {
    p = Manager.createPlayer("http://server/somemusic.mp3");
    p.realize();
    // get volume control for player and set volume to max
    vc = (VolumeControl) p.getControl("VolumeControl");
    if(vc != null) {
        vc.setVolume(100);
    }
    // the player can start with the smallest latency
    p.prefetch();
    // non-blocking start
    p.start();
}
catch(IOException ioe) { ... }
catch(MediaException e) { ... }
...
```

Citirea datelor dintr-un flux

```
...
RecordStore store;
int id;
// play back from a record store
try {
    InputStream is = new ByteArrayInputStream
        (store.getRecord(id));
    Player player = Manager.createPlayer(is, "audio/X-wav");
    p.start();
}
catch (IOException ioe) { ... }
catch (MediaException me) { ... }
...
```

Redarea video

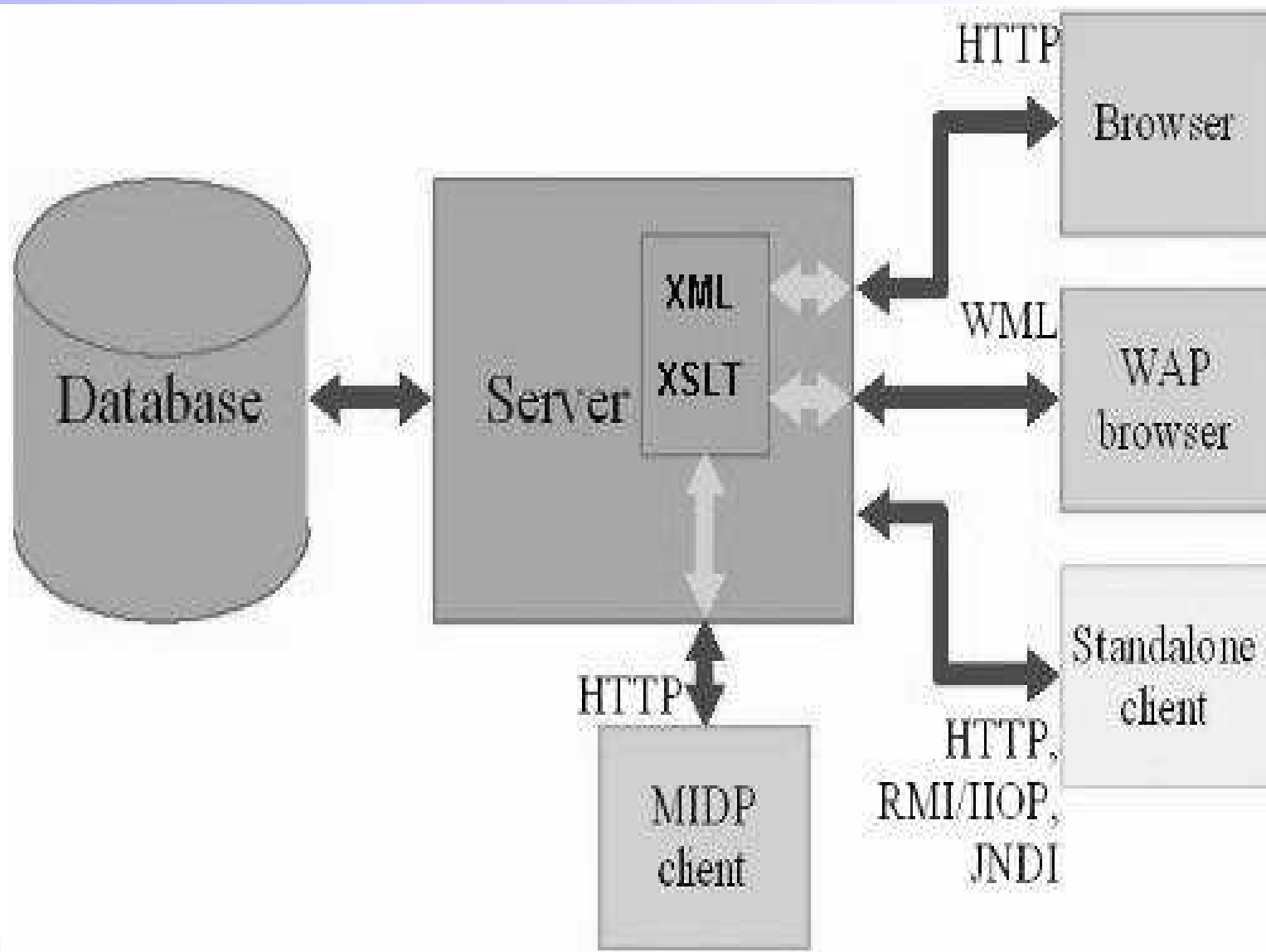
```
...
Player p;
VideoControl vc;
try {
    p = Manager.createPlayer("http://server/somemovie.mpg");
    p.realize();
    // get video control
    vc = (VideoControl) p.getControl("VideoControl");
    ....
    p.start();
}
catch(IOException ioe) {
}
catch(MediaException me) {
}
...
```

XML în J2ME

Ce este XML ?

- Tehnologie pentru aplicații Web.
- Organizare structurată a informației.
- Separă conținutul de prezentare.
- Asigură portabilitatea datelor.
- Simplifică tranzacțiile / căutarea pe Web.
- Folosește taguri și attribute.
- Permite definirea de tag-uri proprii.
- Standard pentru reprezentarea bazelor de date de dimensiuni mici și medii pe Web

Rolul XML



Tipuri de parsere

- **Model** - citesc întreg documentul în memorie
- **Push** - parsează documentul și generează evenimente
- **Pull** - citesc fragmente succesive ale documentului

Exemple: JAXP, ASXMLP, kXML, MinML, NanoXML, Tiny XML, XParseJ (6 - 14 kB)

JAXP

JAXP reprezintă un pachet opțional J2ME oferit pentru configurația CLDC și profilul MIDP, fiind un subset al JAXP J2SE.

Caracteristici:

- Suportă modelul SAX
- Nu suportă modelul DOM
- Suportă spații de nume
- Este format din pachetele: `javax.xml.parsers`, `org.xml.sax`, `org.xml.sax.helpers`
- Este inclus în J2ME Wireless Toolkit.

Crearea unui parser



```
SAXParserFactory factory = SAXParserFactory.newInstance();  
SAXParser parser = factory.newSAXParser();
```

```
InputSource source = new InputSource("<text>Salut</text>");
```

```
/* sau  
RecordStore store;  
int id;  
...  
InputStream is = new ByteArrayInputStream(store.getRecord(id));  
InputSource source = new InputSource(is);  
*/
```

```
DocumentHandler handler = new MyHandler() ;  
parser.parse(source, handler);
```



Crearea unui *DocumentHandler*

```
public class MyHandler extends DefaultHandler {
    public void startDocument() {}
    public void endDocument() {}

    public void startDocument() {}
    public void startElement(String uri,
                             String localName, String qName,
                             Attributes attributes) {}
    public void endElement(String uri,
                           String localName, String qName) {}
    public void characters(char[] ch, int start, int length) {}

    public void error(SAXParseException e) {}
    public void fatalError(SAXParseException e) {}
    public void warning(SAXParseException e) {}
}
```