Mobile Information Device Programming (15)

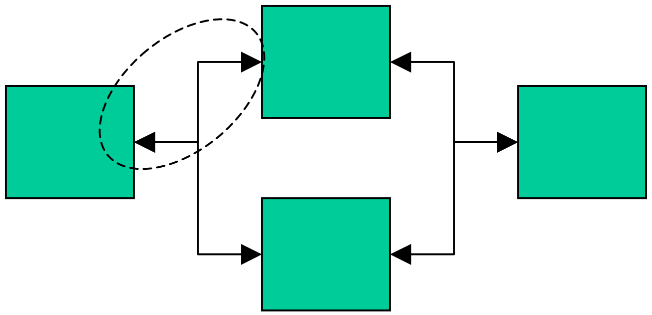
Lecturer: Alireza Mousavi
School of Engineering & Design
www.brunel.ac.uk/~emstaam

Topics

- Wireless Network programming using Sockets
- Wireless Network programming using UDP
- Security for MIDlet Suites

Socket Connection

- A socket connection is one end-point two way communication link between applications running on a network.

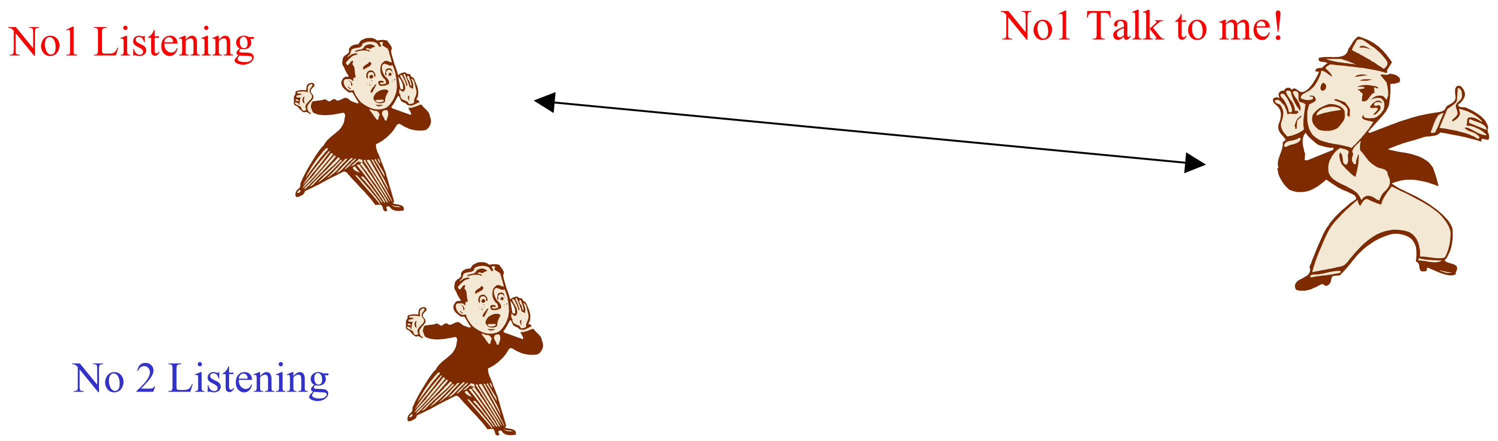


- Socket connections are the most basic and reliable connections between two wireless devices or wireless device to server
- You can use this connection for various client/server applications
- Not every manufacturer supports socket connection on their MIDP devices. "*Be aware of the properties of the device before you write your applications*"

Sending & Receiving data

To use a socket:

- First establish a <u>connection</u> between sender and receiver
- One will be listening for a request for connection and the other will be asking for a connection (*like master/slave*)
- Once the connection is established the system can transmit data



Sending & Receiving data cont.

To **receive** data from the remote server:

1. Established InputConnection
2. Obtain an InputStream from the connection

To **send** data to the remote server:

1. Establish an OutputConnection
2. Obtain an OutputStream from the connection

Also see MIDP 14 lecture notes

Process of network programming in J2ME

1. Open a socket connection with a server or another wireless device using *Connector.open( )* method

2. Create the *InputStream* or *OutputStream* using socket connection to send or receive data

3. Use **read** or **write** operations on the *InputStream or OutputStream* objects

4. Close the connections before leaving the application

Where UDP is suitable

- Compact and fast data transfer
- Loss of a few bits/bytes here and there is not critical for example on a audio/video on mobile phone (static data as opposed to long pauses that may happen in audio stream using socket connection)
- Frequent transmission of similar data / loss of some parts does not matter
- There is no socket communication available

Wireless network programming with Datagram

- Datagram is an independent and self-contained data sent over the network
- Datagram’s arrival, time of arrival and content are not guaranteed
- It is a packet-based communication and **does not** have a dedicated open connection as **Stream** does.
- GPRS supports this type of data transmission
- Datagram communication is based on User Datagram Protocol (UDP)

Coding MIDlets to support Datagram Comms.

Take the following steps:

1. Establish a datagram connection
2. Create a *send datagram* object with destination and message body
3. Send the message using the established datagram connection
4. Create a *receive datagram* object with pre-destined buffer (use the *getMaximumLength( )* method to determine the packet size
5. Keep the connection open whilst the data fills the buffer
6. Free up the datagram connection after use

Datagram Connections

```
DatagramConnection  DC = (DatagramConnection)
    Connector.open("datagram://localhost : 5000");
```

Method

Host

port

```
DatagramConnection  DC = (DatagramConnection)
    Connector.open("datagram://127.0.0.1: 9000");
```

Target host in "client" mode

Further reading: R. Riggs et al (2003), Programming Wireless Devices with J2ME, Sun microsystems

MIDlet Suites Security

Mobile devices should provide the means for users to take the full advantage of services provided on the WWW and at the same time protect their privacy.

The major players in Mobile Systems security:

1. Network Operators: Providers of specialised services for their customers e.g. Vodafone, O2, Bt CellNet …

2. Mobile Device Manufacturers: Providers of fashionable, easy to use and exciting devices e.g. Motorola, Nokia, …

3. Consumers: Would like devices that are trendy, secure, easy to use, fun …

Assumptions

R. Riggs et al (2003), Programming Wireless Devices with J2ME, Sun microsystems

In order to maintain flexibility in implementation with respect to security mechanisms one needs to assume:

1. MIDlets only need to worry about security exceptions that may occur when using APIs
2. A MIDlet suite is subject to a single *Protection Domain* and its allowed actions
3. The internal representation of protection domains and *Permissions* is implementation specific
4. The user interface for configuration settings and results of authentication is implementation-dependent and outside the scope of MIDP 2.0
5. Device security and authentication modules should protect it from unauthorised interference
6. Security should restrict access not security-sensitive functionality

MIDlet Suites *Untrusted & Trusted*

- Untrusted MIDlet Suites:
  - Are suites that the origin and integrity of the JAR file cannot be verified by the device
  - They execute in a restricted environment where access to protected APIs are restricted (not allowed or user permission is required)

  The APIs which untrusted midlet suites require confirmation are:

  javax.microedition.io.HttpConnection (HTTP)

  javax.microedition.ioHttpsConnection (HTTPS)

- Trusted MIDlet Suites:
  - Are suites that the integrity, authenticity and source of the MIDlet is authorised by the device

Trusted MIDlet Suite

1. Permissions
2. Requesting Permissions
3. Protection Domain
4. Interaction Modes for user Permission
5. Granting Permissions

1. Permissions

- Devices use permission to provide access to protected APIs
- A MIDP implementation gives trusted applications more access to security-sensitive APIs than they give to untrusted ones
- The implementation checks permissions prior to invoking any protected function
- MIDlet suites must have granted permission before implementation provides access to protected functions [For a summary of permissions defined by MIDP see: R. Riggs et al (2003), Programming Wireless Devices with J2ME, Sun microsystems]
- The permission names are case sensitive and need to begin with the package that contains them for example: *javax.microedition.io.Connector.datagram* (permission for datagrams)

2. Requesting Permission

- A MIDlet suite that requires access to protected APIs or functions must request the required permissions by containing them in the JAR manifest or JAD:

```
MIDlet-Permissions: javax.microedition.io.PushRegistry
MIDlet-Permission-Opt: javax.microedition.io.Connector.ssl
```

3. Interaction Modes for User Permission

- A *User* permission provides or denies permission to use a specific API or a function.

  It grants permission by using one of the following Interaction Modes:

  - blanket: Grants full permission for every invocation of API or function by a MIDlet suite
  - session: Grants permission to API or functions by the MIDlet on session basis (i.e. until the MIDlet terminates)
  - oneshot: Grants permission only for the current invocation of an API or function. This mode prompts the user on each invocation of API or function

4. Protection Domain

- A *Protection Domain* defines a set of permissions and Interaction Modes
- The device will put each MIDlet suite into a *Protection Domain*
- Domain Permissions are the only type of permissions that devices can grant to the MIDlets

Permissions that are in protection domain are

1. Allowed: given access to protected functions without user interface

2. User: access provided subject to user authorisation

5. Granting Permissions for trusted MIDlet suites

The process of authorising MIDlet suites consist of:

1. Definition of a set of permissions for the protected APIs and functions on the device.

2. A protection domain consisting of a set of *Allowed* and *User* permissions

3. A set of permissions described in the JAR manifest or JAD files, under *MIDlet-Permissions* and *MIDlet-Permissions-Opt* attributes

4. The author of permissions

MIDlet suite authorisation requirements

- The *MIDlet-Permissions* and *MIDlet-Permissions-Opt* attributes in the JAR and JAD file should be identical
- All requested permissions must be known to the implementation
- All requested critical information must exist in the protection domain

Signing a MIDlet suite

- The author (developer or organisation) will be signature holder
- The signer is responsible for ensuring the sanity of the application provided (not malicious or harmful)
- The electronic signature should have a public and private key(further information E-Enterprise Systems Lecture notes)
- The signature protects the JAR, MIDlet classes, and any other resources. **Note** JAD is not secured

An Example on MIDlet suite signing

The developer own the signature

- Create the MIDlet
- Encode permission requests into the JAR manifest
- Generate a public and private key and certify by one or more protection domain root certificates
- The certificate is used to sign the MIDlet's JAR and create the associated entries in the application descriptor
- MIDlet can be distributed amongst MIDP 2.0 enabled devices that has the appropriate protection domain root certificate