Debreceni Egyetem
Informatikai Kar

Kétszemélyes játék Bluetooth kapcsolaton megvalósítva

Témavezető:
Dr. Fazekas Gábor
egyetemi docens

Készítette:
Szabó Zoltán
programtervező matematikus

Debrecen
2008.

Tartalomjegyzék

1. fejezet

Bevezetés

1.1. A probléma felvázolása

Dolgozatom célja egy példaalkalmazás segítségével bemutatni a Java platformra épülő mobil Bluetooth alkalmazások egyes lehetőségeit. Az alkalmazás a TicTacToe játék mobiltelefonos változata. A játékot két játékos játszhatja egymás ellen egy-egy mobil eszköz segítségével.

1.2. Technológia

Az alkalmazás a Java 2 Micro Edition platformra épül. A kommunikáció Bluetooth kapcsolaton folyik.

1.2.1. Java 2 Micro Edition

A Java 2 Standard Edition (J2SE) az asztali személyi számítógépek platformja, a Java 2 Enterprise Edition (J2EE) pedig a vállalati és szerver alkalmazásoké. Ezek mellett a fogyasztói és beágyazott eszközökön a Java technológiát a Java 2 Micro Edition (J2ME) valósítja meg. Ide tartoznak a mobiltelefonok, PDA-k és egyéb beágyazott eszközök.

A J2ME a Stabdard és az Enterprise Edition-höz hasonlóan szabványos Java API-kból áll, melyeket a Java Community Process (JCP) program keretében fejlesztenek olyan csoportok, melyek vezető készülékgyártókat és szoftverfejlesztőket tömörítenek.

1.2.2. Bluetooth technológia

A Bluetooth adatátviteli technológiát a Bluetooth Special Interest Group (SIG) fejlesztette ki. Ezzel a témával a következő fejezet foglalkozik.

2. fejezet

Bluetooth technológia

A Bluetooth adatátviteli technológiát az Ericsson fejlesztette ki azzal a céllal, hogy egymástól néhány méteres távolságban elhelyezkedő berendezéseket és készülékeket vezeték nélkül kössenek össze. Tipikusan a mobiltelefonok perifériáinak a telefonhoz kapcsolása volt a cél (kihangosító, számítógép, másik telefon stb.). Maga a Bluetooth-rendszer a szabadon igénybe vehető 2,4 GHz-es ISM-sávot használja, azonban a biztonság és a hálózatba köthetőség fokozására speciális protokollt használ az adatátvitel folyamán, amely nagyon egyszerű és biztonságosan üzemeltethető, ugyanakkor gazdaságosan előállítható integrált áramkörre épül. Maximális átviteli sebesség közel 1 Mibit/s is lehet, bár a valós alkalmazások ennél kisebb átviteli sebességet érnek el (96 ... ~700 Kibit/s). Az átfogható távolság antennától függően néhány métertől néhány száz méterig terjedhet. A legnagyobb megengedett teljesítmény 10 mW (10 dBm) lehet.

Maga a Bluetooth protokoll többféle párosítást tesz lehetővé. Lehet két készülék között pont-pont összeköttetésként használni és lehet több készülék között, virtuális hálózatba kötést is létrehozni. Szintén a rendszer alapfunkciója a titkosíthatóság. Szoftveres konfigurálhatósággal három biztonsági szint érhető el. A legalacsonyabb esetben a készülékekhez bárki csatlakozhat jóváhagyás nélkül, a középső szint esetében szintén, azonban jóváhagyásra minden csatlakozáskor szükség van. A legmagasabb biztonsági szint csak akkor engedi a készüléket csatlakoztathatni, ha a bejelentkező előre regisztrált a fogadóeszközben.

2.1. Történelem

A Bluetooth technológia kialakulása 1994-ben kezdődött, amikor az Ericsson elindított egy tanulmányt a mobiltelefonokat különböző tartozékaikkal való összekapcsolásának lehetőségeiről. A mérnökök egy kis teljesítményű, alacsony költségű rádió hullámot kerestek a kábeles kapcsolat helyettesítésére. Továbbá arra is ráébredtek, hogy ahhoz, hogy a technológia sikeres legyen, szabadon használhatónak kell lennie. 1998 elején az Ericsson, az Intel, az IBM, a Nokia és a Toshiba létrehozta a Bluetooth Special Interest Group-ot (SIG) a Bluetooth technológia nyílt specifikációjának kifejtésére. Az alapító vállalatok 1998 májusában nyilvánosságra hozták az SIG-t, és meghívtak más vállalatokat, hogy csatlakozzanak. 1999 júliusában megjelent a Bluetooth specifikáció 1.0-ás verziója. 1999 decemberében négy újabb vállalat -3Com, Agere, Microsoft, Motorola- csatlakozott.

Azóta a technológia elterjedt, és további vállalatok csatlakoztak az SIG-hez, és engedélyt kaptak BluTooth-t támogató termékek forgalmazásához. A tagok hamar hozzáférhetnek az újabb specifikációkhoz, és lehetőségük van kinyilvánítani a véleményüket.

És hogy honnan is származik az elnevezés? Egy 940 és 981 között uralkodó dán király, Harald Blåtand után. Az ő vezetékneve szabad fordításban „kék fog”. Uralkodása alatt egyesítette Dániát és Norvégiát. Mivel a technológia kifejlesztésének egyik fontos célkitűzése a telekommunikációs és a számítástechnikai ipar egyesítése, a csapat egy volt történész tagjának ötlete alkalmasnak tűnt a név megválasztására.

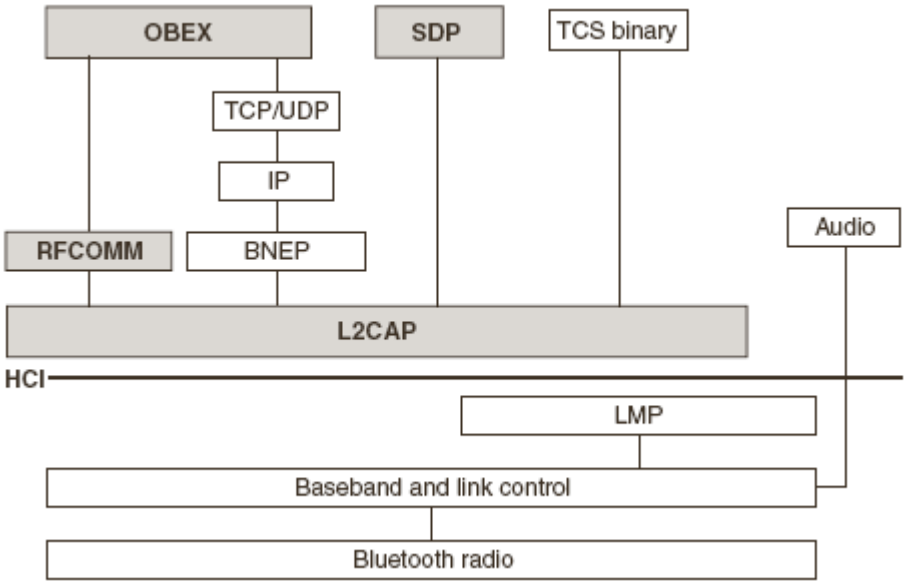
2.2. Bluetooth specifikáció

A Bluetooth specifikáció leírja a kábel nélküli eljárást a különböző gyártók Bluetooth eszközeinek kompatibilitásának biztosítására. Definiálja a teljes rendszert a hardvertől egészen a felhasználói szintig. A specifikáció elég bonyolult, mivel a széleskörű témát fed le. A legmagasabb szinten a specifikáció két részre bomlik. Az első rész a specifikáció veleje.

Leírja a Bluetooth protokollt és a kapcsolódó részeket, mint a tesztelés és minősítés. A Bluetooth protokoll az OSI szabvány alapján épül fel. A második rész a Bluetooth profilokat írja le. Ezek alapvetően felhasználói modellek. Azt írják le, hogy az eszközök hogyan használják a BT protokollt. Egy Bluetooth profil a protokoll rétegek alkalmazhatóságának halmaza.

2.3. Bluetooth protokoll

A 2.1. ábra a Bluetooth protokollt mutatja. A szürke téglalapok a Java APIs for Bluetooth wireless technology (JABWT) tartalmazza. A protokoll alkot/megfogalmaz



2.1.ábra. Bluetooth protokoll

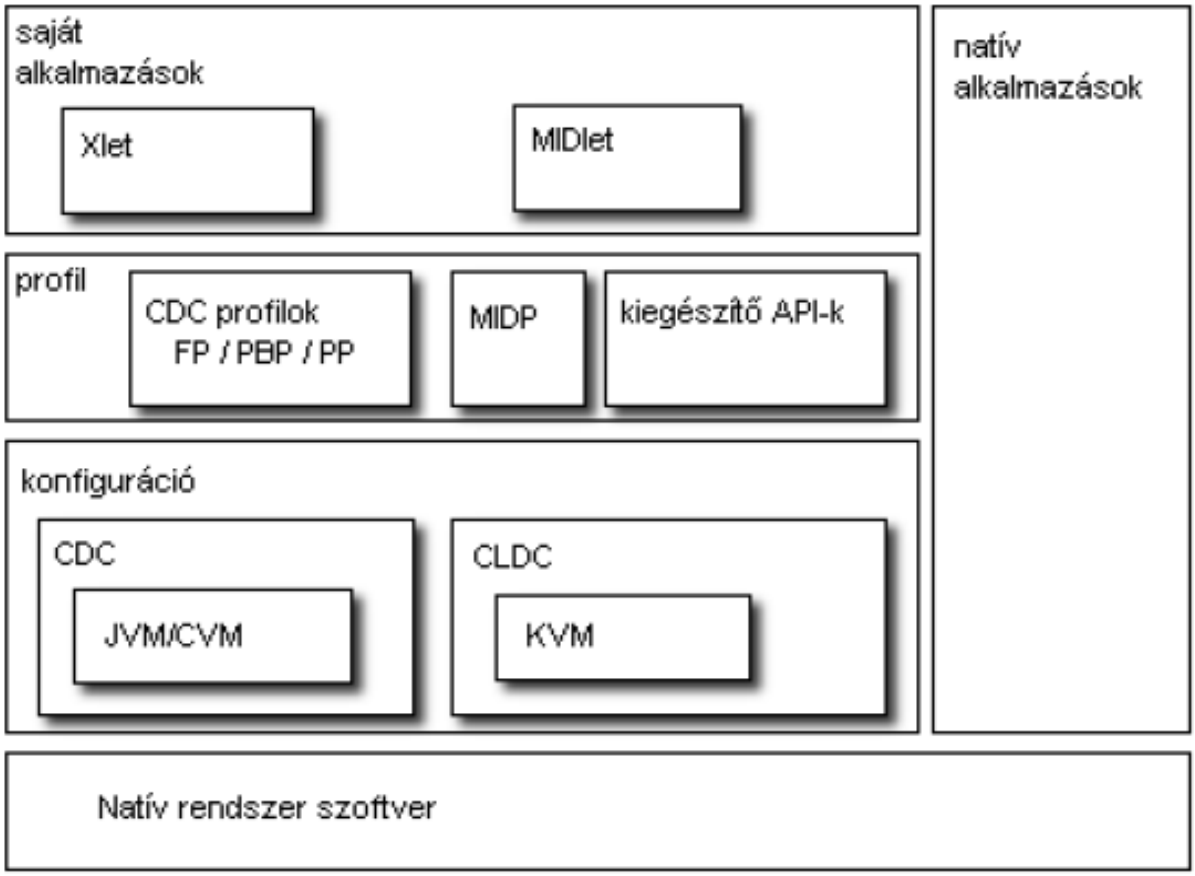
- A **Bluetooth radio** réteg leírja a Bluetooth eszközök követelményeit.
- A **Baseband and link control** az eszközök fizikai összeköttetéséért felelős réteg.
- Az **Audio** igazából nem egy réteg. Azért szerepel itt, mert alapvetően része a Bluetooth kommunikációnak. Hang adatok közvetlenül továbbítódnak a Baseband rétegből és a rétegbe.

- A **Link Manager Protocol (LMP)** felelős a kapcsolat beállításáért és kialakításáért a Bluetooth eszközök között, kezeli és továbbítja a Baseband rétegből származó csomagokat. Továbbá ez a réteg felelős a biztonságért. Elvégzi az azonosítást és a titkosítást.
- A **HCI** az interfész az alsó és felső rétegek között.
- A **Logical Link Control and Adaptation Protocol (L2CAP)** összefogja az alsóbb rétegek által létrehozott különböző logikai kapcsolatokat és továbbítja a felsőbb rétegek felé.
- Az **SDP** eszközrendszert nyújt az alkalmazásoknak hogy elérjék a szolgáltatásokat.
- Az **RFCOMM** protokoll kezeli a soros portokat az L2CAP-n keresztül. Egy eszköznek lehet több konkurens kapcsolata, és több eszköz kapcsolódhat össze egyszerre.
- A Bluetooth eszközöknek képesnek kell lennie hálózatok kialakítására és információváltásra. A **Bluetooth Network Encapsulation Protocol (BNEP)** betömöríti a különböző hálózati protokollokból származó csomagokat és közvetlenül továbbítja az L2CAP-n keresztül.
- A **Telephony Control Protocol Specification, Binary (TCS binary)** réteg definiálja a hívásirányítást, a hang létrehozást és adathívásokat Bluetooth eszközök között. A réteg az L2CAP-n alapszik.
- A többi réteg átvett protokoll. A feljebb tárgyalt rétegekre épülnek (pl. az **OBEX** az RFCOMM-ra, az **IP** pedig a BNEP-re).

3. fejezet

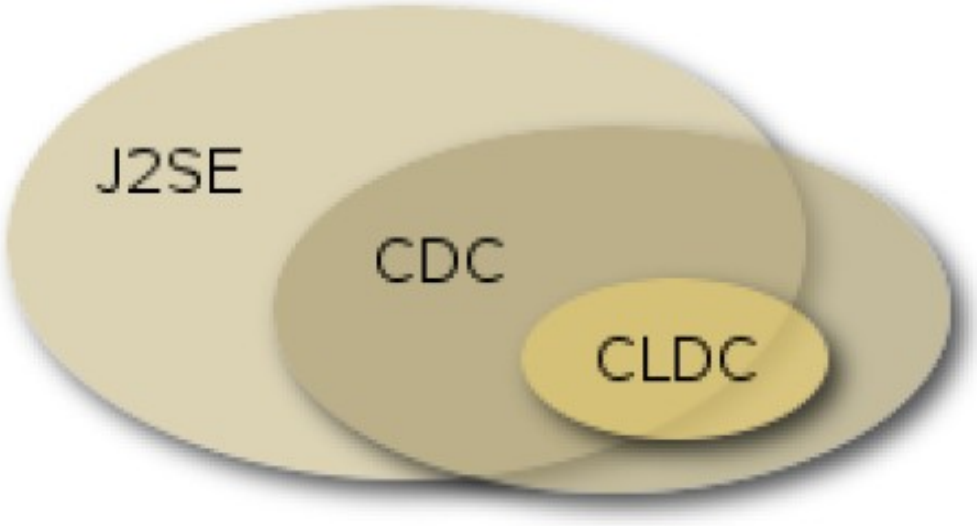
Java 2 Micro Edition

A J2ME mobil eszközök számára kifejlesztett technológiák és specifikációk gyűjteménye. Egy konfiguráció adja az architektúra alapját. Erre épül az adott eszköz működését meghatározó profil és egyéb kiegészítő API-k. Ezek képezik az alapot saját alkalmazásaink elkészítéséhez.



3.1. ábra. A J2ME architektúra

A konfigurációk szerepe az, hogy elfedje az eszközök különbségeit, és szabványos alapot adjon az alkalmazásfejlesztéshez. Azaz az alkalmazások minél több készüléken legyenek futtathatók változtatás nélkül. Tartalmaznak egy Java virtuális gépet és egy minimális API-t. Az API tartalmazza a Java SE API-jának egy részét.



3.2. ábra. Az API-k kapcsolata

A konfigurációkra épülő profilok határozzák meg a fő funkcionalitást és egészítik ki a konfiguráció által kínált API-kat egy adott eszközkategóriában. Segítségükkel a fejlesztők használhatják az eszköz felhasználói felületének részeit és a hardvert, például a fényeket, a hangot és a rezgést.

3.1. Technológiák kisméretű mobil eszközök számára

Kisméretű eszközökön olyan készülékeket értünk, melyek korlátozott erőforrásokkal bírnak. 16 vagy 32 bites processzorral rendelkeznek. Memóriájuk160 és 512 között van. Ide tartoznak a mobiltelefonok vagy egyes PDA-k.

3.1.1. Connected, Limeted Device Configuration (CLDC)

A CLDC fejlesztésénél az alapvető elvárás a lehető leggyorsabb működés és a minél kisebb memóriaigény volt. A CLDC által használt virtuális gép a Kilobyte Virtual Machine (KVM). Amint az a nevében is benne van, kilobyte-os nagyságrendű a memóriaigénye és a mérete. E kettő feltételhez a JVM képességeit jelentősen szűkíteni kellett. Ebből a adódóan a KVM nem kompatibilis a Java SE-ben használt JVM-mel.

A CLDC csak a legalapvetőbb osztályokat definiálja, hogy minél inkább független maradjon az alkalmazásmodellektől. Az API a következő Java csomagokat tartalmazza:

- java.lang
- java.util
- java.io
- javax.microedition.io

Ezek közül az első három a megfelelő Standard Edition-beli csomag részhalmaza. A negyedik a Generic Connection Framework (GCF) implementációja, amely a hálózat hozzáférést valósítja meg.

3.1.2. Mobile Information Device Profile (MIDP)

A MIDP-t a CLDC-t használó eszközökre fejlesztette ki a a Mobile Information Device Profile Expert Group (MIDPEG) a Java Community Process keretében. Ez a fejlesztő csoport több olyan céget tömörít, amelyek a mobil informatikában érdekeltek. Mobilszolgáltatókat, készülékgyártókat, szoftverfejlesztőket. A MIDPEG definiálja a Mobil Információs Eszközök (MID) minimális hardverkövetelményeit:

- Kijelző: minimális méret: 96x54 pixel, torzítási arány: 1:1, színmélység: 1 bit
- Memória: 128 KB nem törlődő memória a MIDP komponenseknek, 8KB nemtörlődő memória az alkalmazás által létrehozott perzisztens adatoknak és 32 KB törlődő memória a Java futtatásának
- Input: vagy egykezes-, vagy kétkezes billentyűzet, vagy érintőképernyő
- Hálózat: kétutas, rádiós, esetleg időszakos hozzáférés korlátozott sávszélességgel

A profil néhány dolgot feltételez az eszköz operációs rendszeréről, mint például a hardver, illetve az alkalmazás életciklusának kezelését, vagy a hálózathoz való hozzáférés biztosítását.

A MIDP kiegészíti a CLDC által adott alap API-t és a fejlesztéshez egy szabványos platformot biztosít. A specifikáció a következő részekből áll:

- MIDlet alkalmazásmodell
- felhasználói felület
- prezisztens adattárolás
- hálózatkezelés
- időzítők (a java.util.Timer és a java.util.TimerTask osztályok a J2SE-ben)

3.2. Technológiák fogyasztói és beágyazott eszközök számára

Ebbe a körbe tartoznak a PDA-k, VOIP telefonok, mobil kommunikátorok, routerek, hálózati nyomtatók vagy beágyazott java futtatható környezetek enterprise-class szerver alkalmazásokhoz. Ezek nagyobb kapacitásúak, mint az előbb tárgyalt kisméretű mobil eszközök. Memóriájuk 2 MB körül van, és gyakran széles sávú hálózateléréssel rendelkeznek.

3.2.1. Connected Device Configuration (CDC)

A CDC egy szabványalapú keretrendszer hálózatkapcsolt fogyasztói és beágyazott eszközökre fejlesztendő alkalmazásokhoz. Támogatja a teljes virtuális gép specifikációt, és az általa kínált API a Standard Edition módosított osztályaiból áll. Ezek interfészei úgy vannak méretezve, és implementációi úgy vannak tervezve, hogy megfeleljenek a kis memóriával rendelkező hardverkörnyezetnek. A jobb erőforrás-kihasználás miatt néhány J2SE-n alapuló osztálykönyvtárnak módosították az interfészeit, és vannak olyanok, amelyeket teljesen elhagytak. Ennek eredményeként egy olyan rugalmas futtatókörnyezet jött létre, amely az ilyen kategóriájú eszközökön kényelmesen működhet. Ez a Compact Virtual Machine (CVM), a kompakt virtuális gép.

3.2.2 Profilok

Foundation Profile. A Foundation Profile a legegyszerűbb profil erre a konfigurációra. A CDC API-val együtt alapvető eszközöket kínál az alkalmazásfejlesztéshez, úgy mint a hálózat és I/O támogatás. Azonban nem tartalmaz semmiféle grafikai vagy GUI támogatást.

Personal Basis Profile. A Personal Basis Profile egy struktúrát ad olyan lightweight komponens eszközkészletek fejlesztéséhez, amik AWT alapú GUI alapokra, JavaBeans futtatási támogatásra és Xlet alkalmazás programozási modellre épülnek. A Personal Basis Profile tartalmazza a Poundation Profile API-t.

Personal Profile. A Personal profile teljes AWT, applet, és korlátozott JavaBeans támogatást nyújt. Emellett része a Personal Basis Profile API.

3.3. A MIDlet alkalmazásmodell

A mobil alkalmazások a konfigurációkra, a profilokra és esetleg egyéb kiegészítő API-kra épülnek. A CDC-t használó kategóriában ezeket az alkalmazásokat Xletnek, a CLDC fölé fejlesztetteket pedig MIDletnek hívjuk. Számunkra jelenleg a MIDlet az érdekesebb, mivel a TicTacToe játék MIDlet. Így az Xlet-ekkel most nem foglalkozom.

A MID alkalmazások egy szokványos jar fájlba csomagolva töltődnek a telefonra adatkábelen, Bluetooth-on, infrán, esetleg WAP-on keresztül, vagy egyéb más módon. Egy ilyen jar egy midlet készlet, amely több midlet-et tartalmaz. Egy MIDlet a jar-ban található összes osztályt el tudja érni. Ezt a készletet a letöltés után az Application Management Software (AMS) telepíti, és kezeli a továbbiakban a jar leíró fájlja (manifest.mf) alapján. Létezik még egy Java alkalmazásleíró (JAD) fájl is a jaron kívül, amely tartalma hasonlít a manifest fájléhoz. Egy MIDlet készlet telepítése a JAD feltöltésével kezdődik. Ennek tartalmából az AMS eldönti, hogy képes-e futtatni az alkalmazást, vagy hogy ez az alkalmazás már telepítve van-e a telefonon. Az adatok ellenőrzése után a JAD-ban található címről letölthető a jar archivum. Néhány telefon típus nem igényli az alkalmazásleíró meglétét,

hanem a jart közvetlenül is képes kezelni.

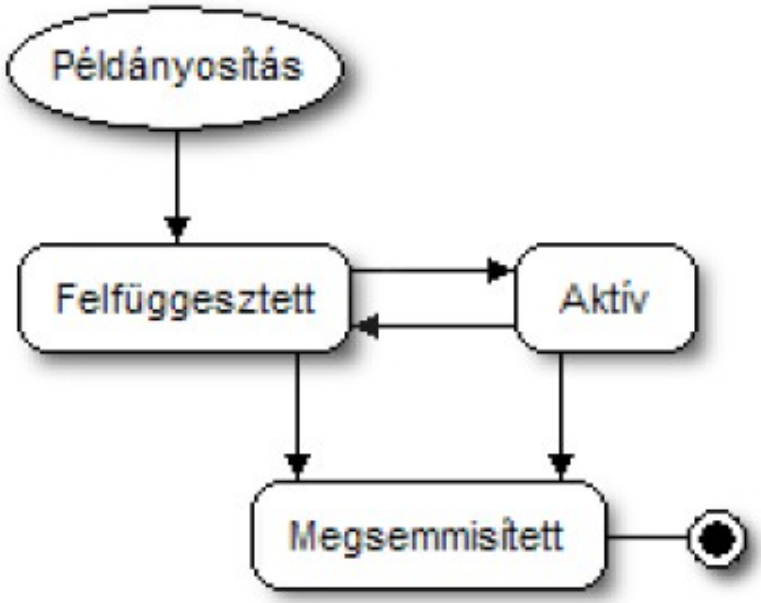
Minden midlet a MIDlet osztály leszármazottja, és implementálnia kell annak alábbi három metódusát:

- public void startApp()
- public void pauseApp()
- public void destroyApp(boolean unconditional)

Ezek a MIDlet életciklusának kezelésében játszanak szerepet.

3.3.1. Életciklus

A MIDP API **javax.microedition.midlet** csomagja a midletek életciklusával foglalkozik.



3.3. ábra. Egy MIDlet életciklusa

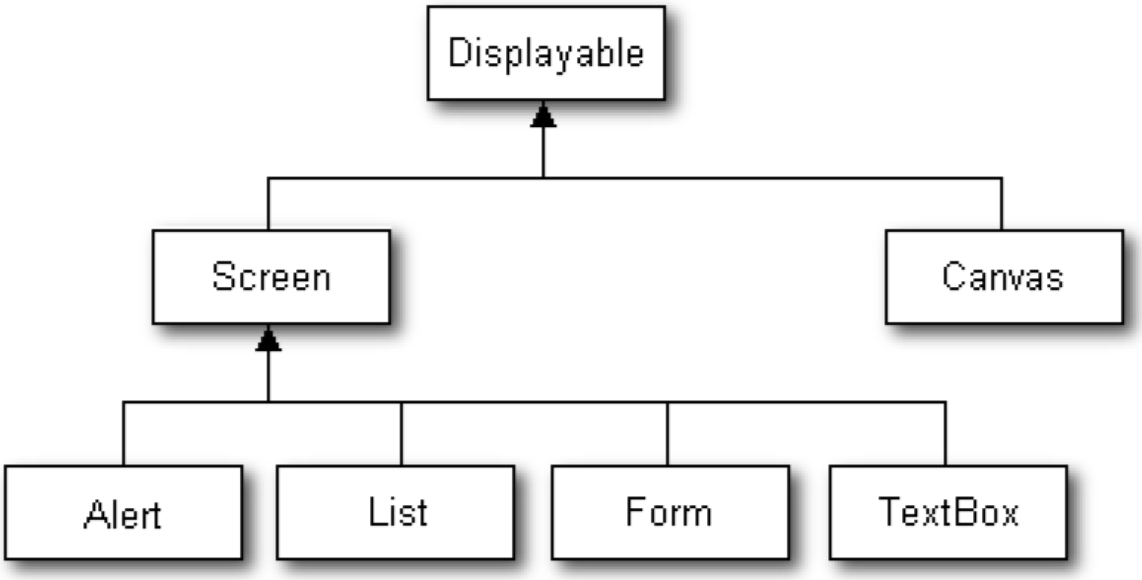
A MIDlet életciklusának három állapota van. Állapotváltozásokat kezdeményezhet maga az alkalmazás a MIDlet osztálytól örökölt metódusainak meghívásával, vagí külső események hatására az AMS értesíti a midletet az életciklus-változásokról. Ekkor szintén a

megfelelő metódus hívódik meg.

A midlet a telepítéskor megsemmisített állapotba kerül. Ekkor nincs betöltődve a KVM-be. Az alkalmazás indításakor az AMS meghívja a midlet startApp() metódusát, és az aktív állapotba kerül. A futás szüneteltetésekor a pauseApp() metódus hívódik meg és az alkalmazás felfüggesztett állapotba kerül. A szüneteltetés magától is bekövetkezhet például egy bejövő telefonhívás alkalmával, vagy ha új üzenet érkezik a telefonra. Bár ebben a kérdésben a készülékgyártók MIDlet implementációi különböznek egymástól. Van ugyanis olyan implementáció, amely az előbb említett események hatására nem függeszti fel a működést, csak úgy veszi, hogy az alkalmazás képernyőjét egy másik képernyő eltakarja. Amikor az alkalmazást befejezzük, mielőtt az AMS törölné a midletet, meghívja a destroyApp() metódust. Ha ennek a paramétere nem igaz, akkor a midlet megakadályozhatja a befejezést, ha igaz, akkor nem.

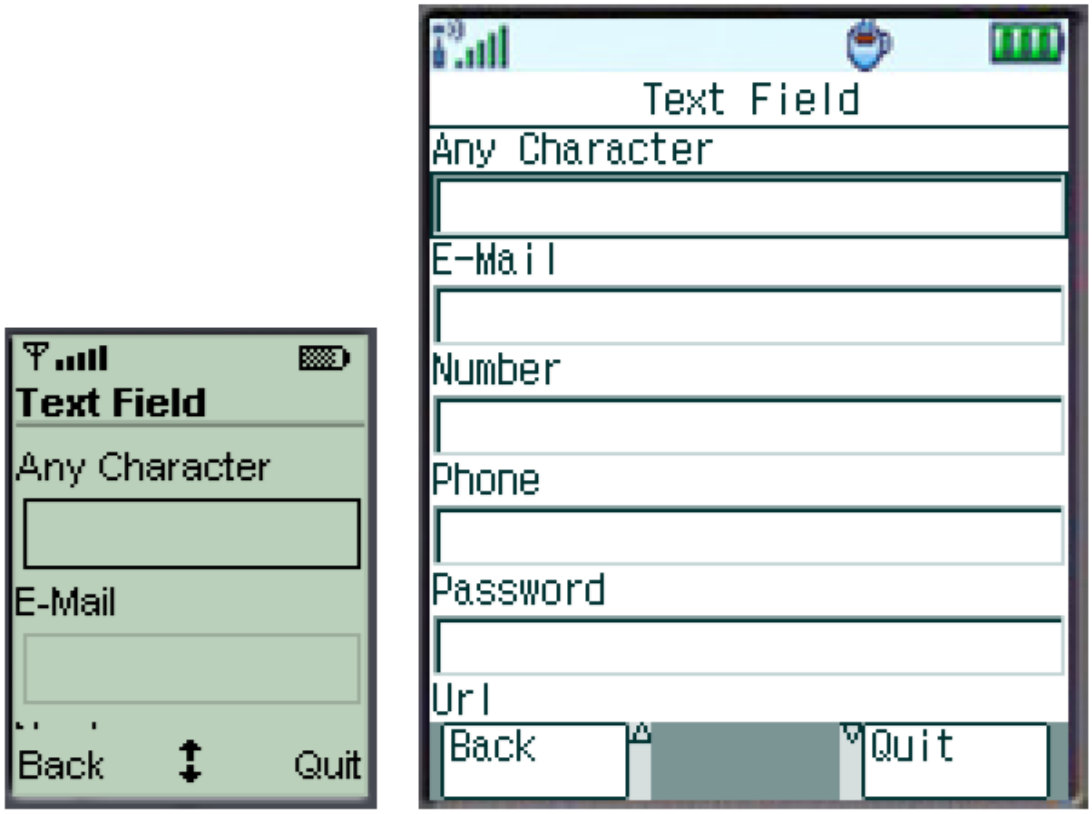
3.3.2. Felhasználói felületek

A **javax.microedition.lcdui** a felhasználói felületekért felelős csomag. A kis méret- és erőforrásigény elérése érdekében a KVM nem támogatja az olyan létező felhasználói felület API-kat, mint az AWT és a Swing. Ezek nemcsak nagy méretük miatt alkalmatlanok a mobiltelefonos használatra, hanem azért is, mert a mobil eszközök eltérő megközelítésből indulnak ki a felhasználói felületekkel kapcsolatban. Ezért ezek helyett egy új, a kis memóriával rendelkező eszközökre illeszkedő, és azok különbözőségeiből fakadó követelményeknek is megfelelő API-t hoztak létre. Az LCDUI központi fogalma a képernyő. Az alkalmazások futása során a felhasználó képernyőről-képernyőre halad. A kijelzőn mindig az aktuális képernyő látható, amit a kijelző objektum (Display) setCurrent(Displayable d) metódusával állíthatunk be. A MIDP API-ban a képernyőt a Displayable osztály reprezentálja.



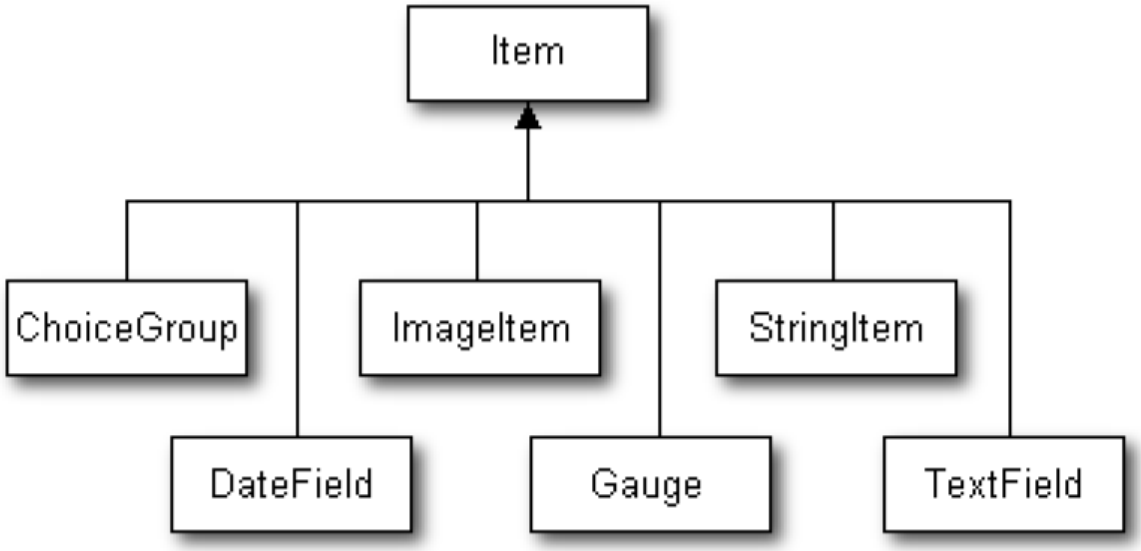
3.4. ábra. A képernyők osztályhierarchiája

A Displayable két közvetlen kiterjesztése a Canvas és a Screen osztály. A Canvas és leszármazottjai az **alacsony szintű felhasználói felületek**, amelyek teljes hozzáférést biztosítanak a kijelzőhöz. A Canvas osztály definiál egy paint(Graphics g) absztrakt metódust, amelyet a leszármazottaknak implementálniuk kell, és amely a képernyő kirajzolását végzi a Graphics objektumon keresztül. A kijelzőnek ez a fajta programozása nagyon fontos például játékok fejlesztésénél. A TicTacToe játékban a táblát megjelenítő képernyő is a Canvas leszármazottja. A Screen osztály a **magas szintű felhasználói felületek**et megvalósító osztályok őse. A magas szintű azt jelenti, hogy a képernyőtartalmak elemeinek a kirajzolása a telefon szoftverének a feladata. Az határozza meg az elemek méretét, színét, helyét, stb. A Programozó dolga, hogy a képernyőn megjelenítendő tartalmat megadja. Így ugyanaz a képernyő különböző eszközökön különbözőképpen jelenhet meg.



3.5. ábra. Egy űrlap megjelenése két különböző eszközön

A Screen osztályból származik közvetlenül az Alert, a List, a TextBox, amelyek egy egész képernyőt foglalnak el, tartalmuk kötött, szerkezetüket nem lehet megváltoztatni; és a Form osztály, amely űrlapok gyors és egyszerű készítését tesz lehetővé. Egy Formon különböző elemeket helyezhetünk el, mint például képek, szöveg elemek, szövegbeviteli mezők vagy választási lehetőségek. Ezek az elemek az Item osztály leszármazottjai.



3.6.ábra. Az Item osztály leszármazottjai

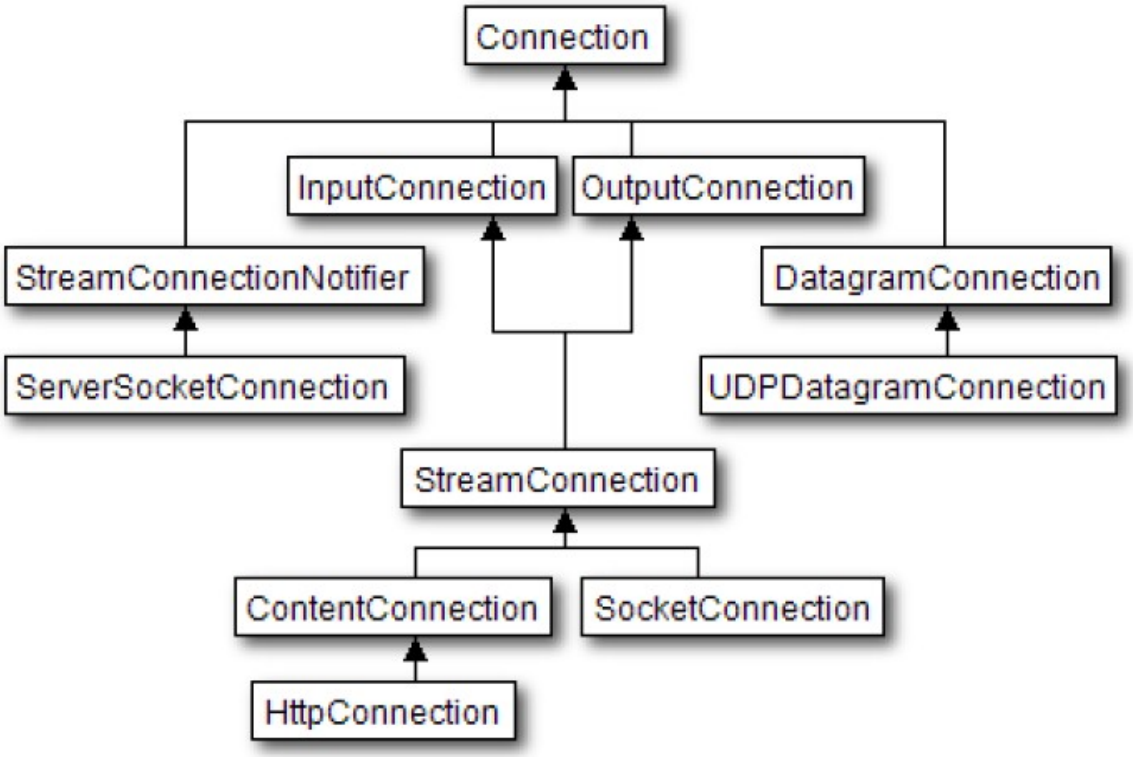
Az alacsony- és magas szintű API-k nemcsak a tartalom megjelenítés módjában különböznek, hanem az események kezelésében is.

- Az alacsony szintű API hozzáférést enged a billentyűzethez, azaz különböző gombnyomás eseményekhez eseménykezelőt tudunk rendelni. Egy alacsony szintű esemény bekövetkezésekor egy az eseménynek megfelelő metódus hívódik meg, amelyben az eseményeket kezelhetjük. Ilyen metódus például a keyPressed(int keyCode), vagy a keyReleased(int keyCode). A keyCode az eseményt kiváltó gomb kódját jelent, amelyek a Canvas osztályban statikus adattagokként szerepelnek.
- A magas szintű felhasználói eseményeknek két fajtája van:
  1. Egy Item típusú elem állapotának megváltozása. Ezek figyelését az elemhez rendelhető, egy az ItemStateListener interfészt implementáló példány végzi. Az esemény bekövetkezésekor a példánynak az eseménykezelő interfésztől örökölt itemStateChanged(Item item) metódusa hívódik meg. Az Item típusú paraméter az eseményt kiváltó elemet hivatkozza.

2. Egy Command esemény bekövetkezte. Az ilyen típusú események figyelése a CommandListener interfész egy implementációjának a feladata, amelynek meg kell valósítania az interfész commandAction(Command c, Displayable d) metódusát, ami esemény bekövetkeztével hívódik meg. A paraméterek itt is az eseményt kiváltó parancs és az azt tartalmazó képernyőt jelenti.

3.3.3 Hálózatkezelés

A **javax.microedition.io** csomag a Generic Connection keretrendszert (GCF) valósítja meg. A legegyszerűbb általános kapcsolattípust a Connection interfész írja le. Ezt terjeszti ki a csomag többi interfésze, amelyek specializálják a kapcsolat típusát.



3.7. ábra. A Generic Connection Framework interfészhierarchiája

A MIDP négyféle kapcsolattípust támogat:

- http
- Datagram
- Socket
- Comm (logikai soros port kapcsolat)

Ezek közül a http kötelezően támogatott típus.

Egy kapcsolat megnyitása a Connector osztály open() metódusával történik. Ennek a kötelező paramétere egy String, ami a cél URL-t reprezentálja a kapcsolattípusnak megfelelően. Visszatérési értéke egy megfelelő kapcsolatobjektum.

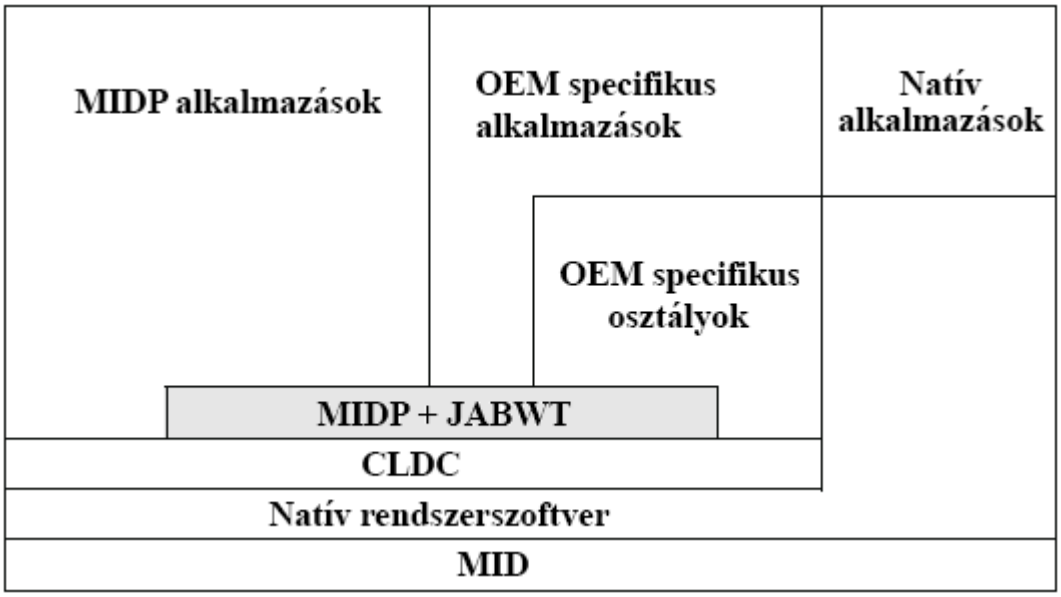
3.3.4. Prezisztens tárolás

A MIDP egy egyszerű lehetőséget kínál adatok tárolására és visszatöltésére az alkalmazás leállítása és újraindítása után is. Ennek megvalósítása a **javax.microedition.rms** csomagban érhető el. Az RMS a Record Management System rövidítése. Az RMS kezeli a tárolt adatokat, ami egy rekord orientált adatbázis modellen alapul. Használatakor névvel ellátott tárolókat (RecordStore) hozhatunk létre, amelyek a MIDlet készlethez kötődnek és ennek törlésekor maguk is törlődnek. A rekordtárolóhoz a MIDlet készlet tagjai férhetnek hozzá, más MIDlet készlet tagjai nem. Ezekhez rekordként adhatunk adatokat és olvashatunk belőlük. A rekordok bájt tömbök, amelyek számozva kerülnek a tárolóba. Kiolvasásuk történhet sorszámuk alapján, vagy kérhetjük a rekordok felsorolását a RecordEnumeration segítségével.

A mobiltelefonok erre a célra néhány 10 KB és több megabájt között változó méretű területet biztosítanak.

3.4. Java APIs for Bluetooth Wireless Technology (JABWT)

A JABWT-t a Java Community Process (JCP) szakértő csoport definiálta. Az alapvető API-k egy a CLDC-n alapuló opcionális csomag J2ME eszközök számára. A JABWT alapvető célja olyan API-k definiálása, amelyek lehetővé teszik Bluetooth alkalmazások fejlesztését a Java nyelv egyszerű, de hatékony módján. Egyesíti a Bluetooth és a Java technológia előnyeit.



3.8. ábra. A JABWT elhelyezkedése a MIDP architektúrában

3.4.1. A JABWT felépítése

A funkcionalitás alapján a JABWT három fő kategória osztható:

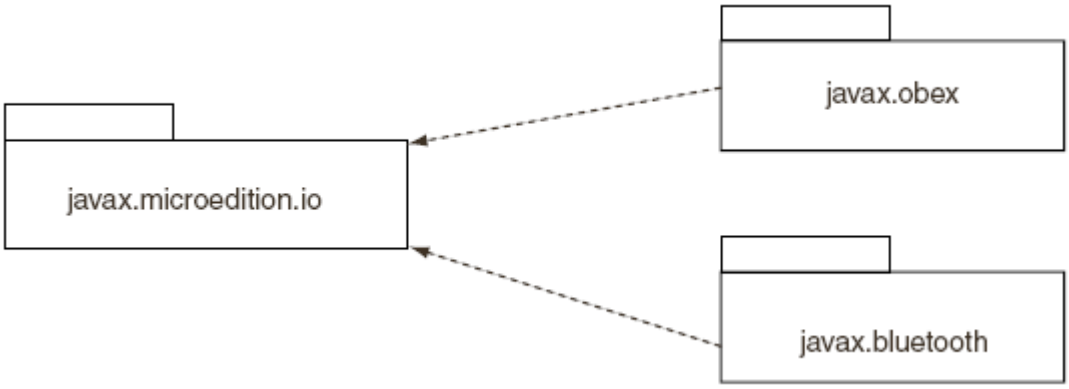
- eszközök, szolgáltatások keresése és szolgáltatásnyújtás
- kommunikáció
- kapcsolatok kezelése

3.4.2. Java csomagok

A JABWT alapvetően két Java csomagból áll:

- javax.Bluetooth
- javax.obex

Mind a kettő különálló opcionális csomag. Ebből következik, hogy a CLDC implementáció tartalmazhatja külön-külön őket, vagy mindkettőt, vagy egyiket sem. A **javax.Bluetooth** csomag a Bluetooth API-t tartalmazza, a **javax.obex** csomag pedig az OBEX API-kat .



3.9.ábra. Csomagstruktúra

4. fejezet

Alkalmazás specifikációk

4.1. Áttekintés

4.1.1. Általános leírás

Az alkalmazás a TicTacToe játékot valósítja meg mobil környezetben. A játékot két személy játszhatja egymás ellen, két külön készüléken. A kommunikáció Bluetooth kapcsolaton keresztül valósul meg.

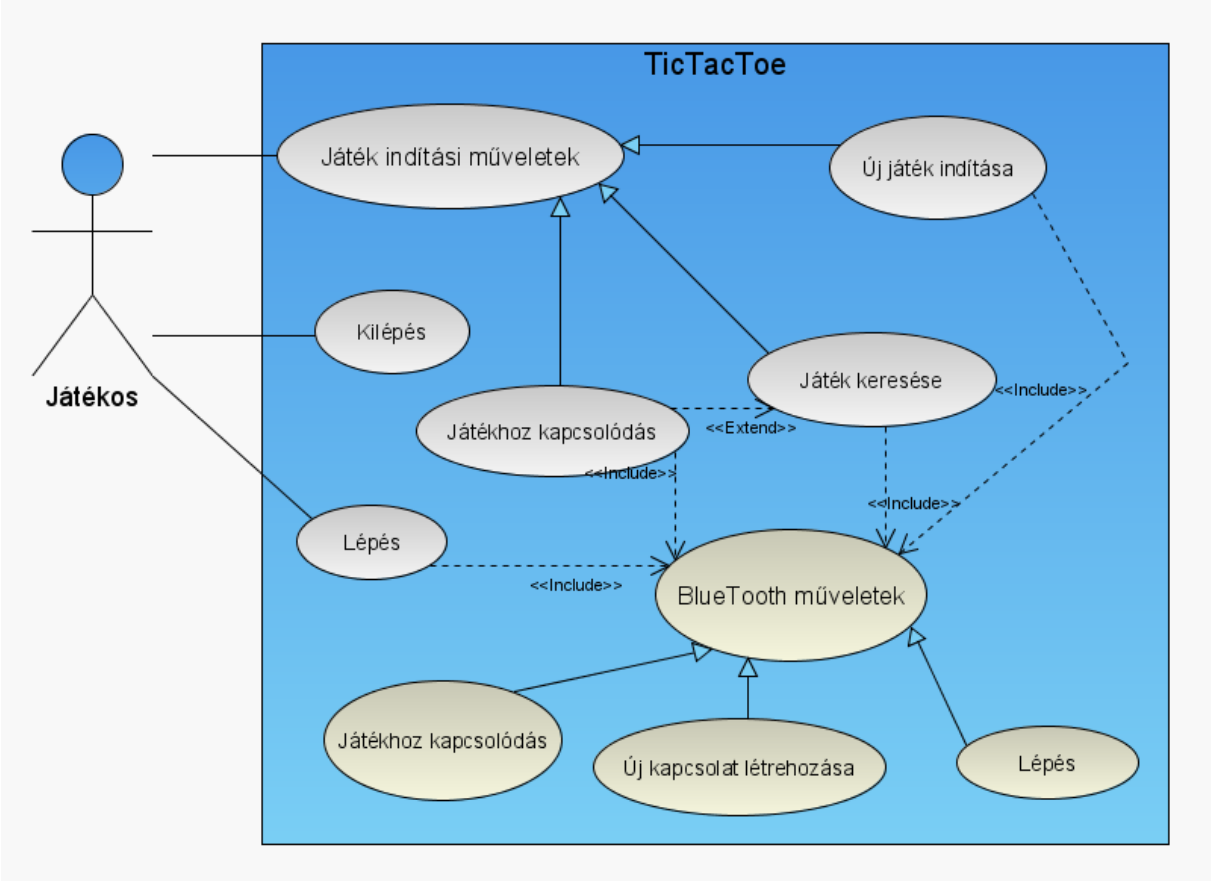
4.1.2. Általános követelmények

- A játék a 3x3 mezős TicTacToe szabályai szerint folyik
- Ha egyszerre kettőnél több játékos jelentkezik játszani, akkor csak egyszerre egy páros játszhat egy kapcsolatban.
- Az egy játékban résztvevő alkalmazásoknak szinkronban kell működniük.

4.1.3. Rendszerkövetelmények

- A rendszer platformfüggetlen.
- A programozási nyelv: Java.
- A célhardver: CLDC1.0/MIDP1.0 képes mobil eszköz (elsősorban mobiltelefon) Bluetooth kapcsolattal.

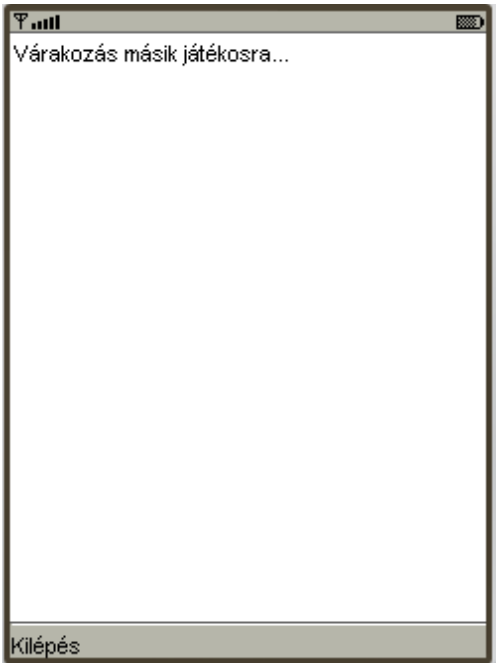
4.2. Felhasználói esetek



4.1. ábra. Felhasználói esetek

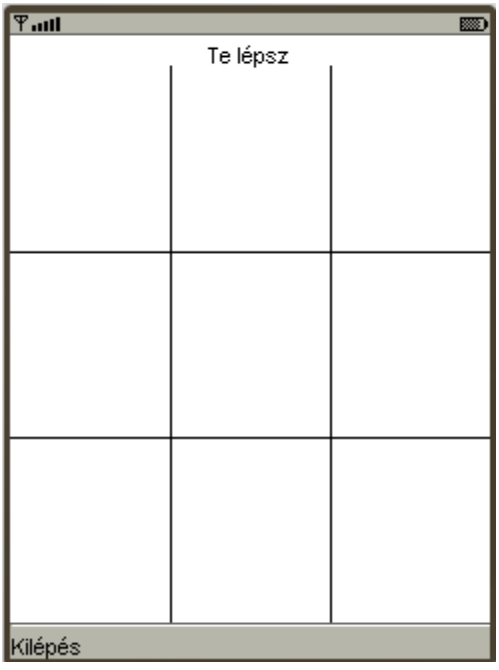
4.3. Felhasználói felületek

A főmenü képernyő

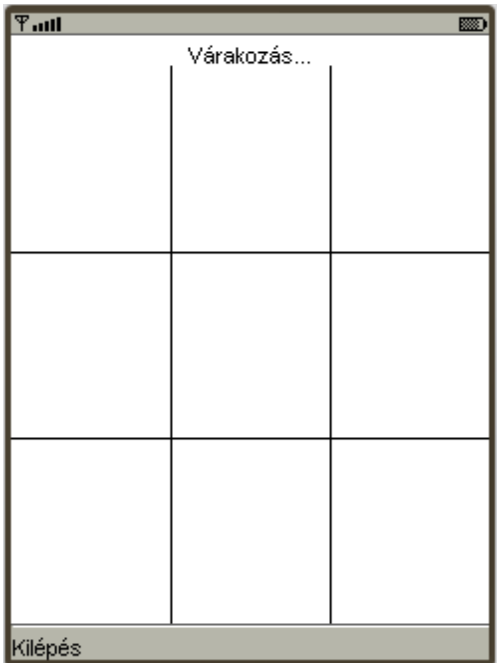


Master játékos képernyője a kapcsolat engedélyezése
után slave játékosra várakozás közben

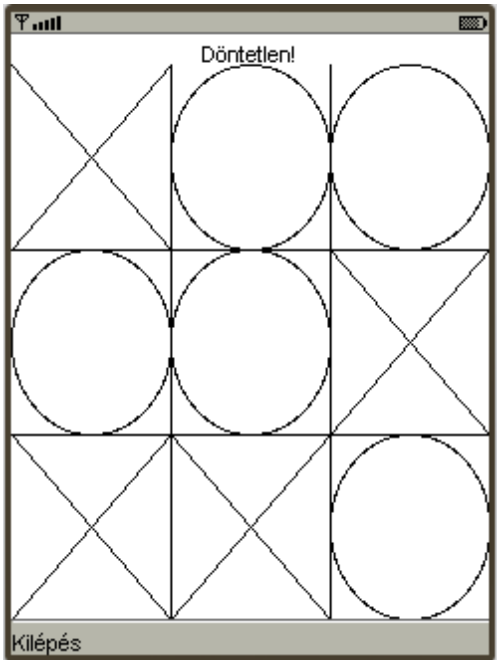
Slave játékos képernyője master keresése után



Master játékos kezdő táblája



Slave játékos kezdő táblája



Egy lehetséges végeredmény

5. fejezet

Alkalmazás architektúra

Az alkalmazás két logikai részből épül fel:

- felhasználói interfész
- másik alkalmazással való kapcsolattartás

A felhasználói interfész megjeleníti a felhasználó számára a lehetséges felhasználói felületeket, másrészt figyeli a játékos, és az ellenfél lépéseit, vagy egyéb műveleteit, és megjeleníti a bekövetkezett változásokat.

A másik alkalmazással való kommunikációs mechanizmus a protokollunknak megfelelően küld információkat a másik alkalmazás felé és vár válaszokra. Egy lépés esetén létrehozza a kapcsolatot, elküldi a lépést leíró kérés tartalmat, majd feldolgozza a másik fél által adott választ. Vagy az ellenfél lépésére várva kapcsolódik, aminek adatait majd a válaszban kap meg.

5.1. Kommunikáció és szinkronizáció

A kommunikáció megvalósításához a Bluetooth protokollt választottam. Ennek oka, hogy azt napjainkban majdnem minden minden Java-képes mobiltelefon megvalósítja, így a Bluetooth kapcsolat miatt nem szűkül az alkalmazást futtatni képes eszközök köre.

6. fejezet

Az alkalmazás megvalósítása

Az alkalmazás egy MIDP alkalmazás, amelynek a fő osztálya a TicTacToe a MIDlet osztály leszármazottja. Továbbá implementálja a CommandListener interfészt, tehát kezeli a magas szintű eseményeket, meghatározván a parancsok viselkedését. Valamint implementálja a Runnable interfészt is, aminek a kommunikációban van szerepe.

Az alkalmazás másik fontos eleme a magát a játék megjelenését megvalósító TCanvas osztály, amely a Canvas osztály leszármazottja. A TCanvas egy alacsony szintű eseménykezelő és alacsony szintű felhasználói felület.

A többi osztály a Bluetooth kapcsolatot és kommunikációt valósítja meg. Ezek az osztályok a Device, a Service és a Client.

6.1. A MIDlet

A TicTacToe példányosításakor létrejönnek az alkalmazásban használt parancs példányok. Az alkalmazás indításakor a TicTacToe osztály startApp() metódusa hívódik meg. Ez lekérdezi az eszköz kijelzőjének adatait, és letárolja azokat, létrehozza a kezdőképernyőt, továbbá inicializálja a Bluetooth könyvtárat. Majd elindítja az alkalmazásnak egy szálát.

```
protected final void startApp() throws MIDletStateChangeException {
    if(canvas == null) {
        cmdExit = new Command("Kilépés", 7, 1);
        canvas = new TCanvas(this);
        canvas.addCommand(cmdExit);
        canvas.setCommandListener(this);
        width = canvas.getWidth();
        height = canvas.getHeight();
        startTime = System.currentTimeMillis();
        msPerFrame = 1;
        events = new byte[8];
        eventsClone = new byte[8];
        eventValues = new int[8];
        eventValuesClone = new int[8];
        eventData = new Object[8];
        eventDataClone = new Object[8];
        running = true;
        setup();
        lastFrameTime = startTime - (long)msPerFrame;
    }
    redraw = true;
    display.setCurrent(canvas);
    thread = new Thread(this);
    thread.start();
}
```

6.1.ábra. A startApp() metódus

```
public void setup() {
  display = Display.getDisplay(this);
  runtime = Runtime.getRuntime();

  font = Font.getDefaultFont();
  fHeight = font.getHeight();
  fBaseline = font.getBaselinePosition();
  textFont(font);

  bt = new Bluetooth(this);

  board = new int[3][3];
  thirdX = width / 3;
  thirdY = (height - fHeight) / 3;
}
```

6.2.ábra. A setup() metódus

A játék vezérlése különböző állapotok segítségével történik a TicTacToe osztály két metódusának hívásával. A draw() metódus az adott állapothoz tartozó képernyőt jeleníti meg a TCanvas osztályon keresztül. A keyPressed() metódus pedig az adott billentyű lenyomásának hatására bekövetkezett változásokat végzi el. Ezen felül különböző Bluetooth események is okozhatnak állapotváltozást.

```
//játékállapotok
final int STATE_START = 0;  //kezdőállapot
final int STATE_FIND  = 1;  //játék keresése
final int STATE_HOST  = 2;  //új játék kezdése
final int STATE_PLAY  = 3;  //játék
final int STATE_OVER  = 4;  //játék vége

//tábla mezőinek értékei
final int PIECE_NONE  = 0;  //üres mező
final int PIECE_X     = 1;  //mező értéke X
final int PIECE_O     = 2;  //mező értéke O

//végeredmény értékei
final int RESULT_NONE = 0;          //nincs még végeredmény
final int RESULT_X    = PIECE_X;    //X nyert
final int RESULT_O    = PIECE_O;    //O nyert
final int RESULT_DRAW = 3;          //döntetlen

int state;
int result;
```

6.3.ábra. A játék vezérlését segítő állapotok

6.2. Főmenü

A főmenü egy adott képernyő, amelyen szövegesen megjelenik két lehetőség, amely közül választhatunk a készülékünk számbillentyűi segítségével:

1. *Új játék kezdése*: e a menüpont választásával a Bluetooth példány start() metódusa hívódik meg, azaz elindul egy új master szál. Továbbá egy új képernyő jelenik meg a *Várakozás másik játékosra...* szöveggel. Miután csatlakozott egy játékos, megjelenik a master játékos kezdőképernyője, és indul a játék.

2. *Játék keresése*: e menüpont választásával a Bluetooth példány find() metódusa hívódik meg, azaz létrejön egy új slave szál. Elkezdi keresni a létező játékokat, azaz a master szálakat, és a keresés folyamatát, és végül az eredményét kiírja a képernyőre.

```
                    :
                    :
                    :
public void keyPressed() throws Exception {
  if (state == STATE_START) {
    switch (key) {
    case '1':
      bt.start("TicTacToe");
      state = STATE_HOST;
      break;
    case '2':
      services = null;
      bt.find();
      state = STATE_FIND;
      msg = "Eszközök keresése...";
      break;
    }
  }
                    :
                    :
                    :
```

6.4.ábra. Az *Új játék kezdése* és a *Játék keresése* parancs feldolgozása

```
public void start(String s) {
    if(serverThread == null)
        try {
            local.setDiscoverable(0x9e8b33);
            String s1 = "btspp://localhost:" + uuid.toString() + ";name=" + s;
            server = (StreamConnectionNotifier)Connector.open(s1);
            ServiceRecord servicerecord = local.getRecord(server);
            servicerecord.setAttributeValue(8, new DataElement(8, 255L));
            servicerecord.setDeviceServiceClasses(0x400000);
            Service service = new Service(null, servicerecord, this);
            serverThread = new Thread(service);
            serverThread.start();
        }
        catch(Exception exception) {
            serverThread = null;
            throw new RuntimeException(exception.getMessage());
        }
}
```

6.5.ábra. A start() metódus

```
public void find() {
    boolean flag = false;

    synchronized("10203040506070809A0B0C0D0E0F010") {
        if(discoverThread == null) {
            discoverThread = new Thread(this);
            flag = true;
        }
    }

    if(flag) {
        find = true;
        discoverThread.start();
    }
}
```

6.6.ábra. A find() metódus

6.3. Játék választása

A főmenü 2. pontjának választása után megjelennek a lehetséges választható játékok 0-tól felfelé sorszámozva. A slave játékos a készülék számbillentyűivel választhat a kívánt játék sorszámának megnyomásával. Miután csatlakozott, megjelenik a kezdőképernyője, és indul a játék.

```
                    :
                    :
                    :
else if (state == STATE_FIND) {
  if (services != null) {
    if ((key >= '0') && (key <= '9')) {
      int i = key - '0';
      if (i < services.length) {

        c = services[i].connect();
        yourturn = false;
        yourpiece = PIECE_O;
        state = STATE_PLAY;
      }
    }
  }
}
                    :
                    :
                    :
```

6.7.ábra. A *Játék választása* parancs feldolgozása

```
public Client connect() throws Exception {
    Client client;
    StreamConnection streamconnection =
            (StreamConnection)Connector.open(record.getConnectionURL(0, false));
    client = new Client(streamconnection);
    client.device = device;
    client.open();

    return client;
}
```

6.8.ábra. A Service osztály connect() metódusa

6.4. A játék menete

A játék menete során megjelenik az aktuális tábla, és egy segéd információ. Ennek lehetséges értékei:

- *Te lépsz*
- *Várakozás...*
- *Győztél!*
- *Vesztettél!*
- *Döntetlen!*

A táblát egy 3x3-mas int tömb reprezentálja, amelyben az elemek a tábla mezőit jelentik. Ha az elem értéke:

0: akkor a mező üres
1: a mezőn X jel található (ez a slave játékos jele)
2: a mezőn O jel található (ez a master játékos jele)

A játékot mindig a master játékos kezdi. A játékosok készülékük számbillentyűivel tudnak választani az üres mezők közül.

```
                        :
                        :
                        :
else if ((state == STATE_PLAY) && yourturn) {
  int row = -1, col = -1;
  switch (keyCode) {
  default:
    switch (key) {
    case '1':
      row = 0;
      col = 0;
      break;
    case '2':
      row = 0;
      col = 1;
      break;
    case '3':
      row = 0;
      col = 2;
      break;
    case '4':
      row = 1;
      col = 0;
      break;
    case '5':
      row = 1;
      col = 1;
      break;
    case '6':
      row = 1;
      col = 2;
      break;
    case '7':
      row = 2;
      col = 0;
      break;
    case '8':
      row = 2;
      col = 1;
      break;
    case '9':
      row = 2;
      col = 2;
      break;
    }
  }
  if ((row >= 0) && (col >= 0)) {
    if (board[row][col] == PIECE_NONE) {
      board[row][col] = yourpiece;
      yourturn = false;
      c.writeInt(yourpiece);
      c.writeInt(row);
      c.writeInt(col);
      c.flush();
      result = checkResult();
      if (result != RESULT_NONE) {
        state = STATE_OVER;
      }
    }
  }
}
                        :
                        :
                        :
```

6.9.ábra. Egy lépés feldolgozása

Összegzés

Nagy valószínűséggel a mobiltelefon a napjainkban legtöbbet használt kommunikációs eszköz. A legtöbb készülékben elérhető a Java ME támogatás és a Bluetooth támogatás. A legelterjedtebb mobil Java alkalmazások a játékok. Ezen belül is a különböző táblajátékok nagyon népszerűek.. A bemutatott alkalmazás architektúrája nem csak a TicTacToe játék, hanem egyéb más kétszemélyes játék megvalósítására alkalmas. Ezek a játékok az ingyenes Bluetooth kommunikációt használva nagyon keresettek lehetnek.

Irodalom

- C Bala Kumar, Paul J. Kline, Timothy J. Thompson: Bluetooth Application Programming with The Java APIs
- André N. Klingsheim: J2ME Bluetooth Programming
- Móricz Attila: Java MIDlet programok Készítése I.
- Paller Gábor: MIDletek
- CLDC specifikáció [JSR 30]
- MIDP specifikáció [JSR 37]
- CDC specifikáció [JSR 36]
- Java APIs for Bluetooth Wireless Technology [JSR 82]
- http://www.elektro-net.hu/cikkek/veznelkphoenix.htm