Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Blown-up rendszer tervezése és megvalósítása

TDK dolgozat

Biczók Gergely
műszaki informatika V. évf.

Fodor Kristóf
műszaki informatika VI. évf.

Kovács Balázs
műszaki informatika V. évf.

Szabó Ágoston
műszaki informatika VI. évf.

Konzulensek:

Rónai Miklós Aurél
BME Számítástudományi és Információelméleti Tanszék

Turányi Zoltán Richárd
Ericsson Traffic Lab

Valkó András Gergely
Ericsson Traffic Lab

Budapest, 2002. október

Tartalomjegyzék

1. Bevezető

> „Információgazdag világban élünk. A probléma nem az elégtelen információból adódik, hanem éppen abból az információáradatból, amit képtelenek vagyunk feldolgozni. Az informatika szerepe, hogy csökkentse, és ne fokozza az információs zűrzavart."
>
> Herbert Simon (1971)

Napjainkban egyre jobban terjednek a digitális személyi asszisztensek (Personal Digital Assistant – PDA), melyeket a felhasználók bárhol és bármikor használhatnak. Ezen kívül nem kell azzal törődniük, hogy mikor, melyik gépen módosították fájljaikat, mert állományaikat egy helyen, a mindig kéznél lévő PDA-n tárolják. Kis méretük miatt a digitális személyi asszisztensek hordozása nem okoz gondot, hiszen azok akár zsebben is elférnek. Azonban használatuk a kis méret miatt hamar kényelmetlenné válhat, hiszen ezek az eszközök kicsi billentyűzettel, egérrel és kijelzővel rendelkeznek.

Felmerül az igény egy olyan megoldásra, amely lehetővé teszi, hogy a felhasználók a PDA-ba épített perifériák helyett, a PDA-hoz egyszerűen csatlakoztatható, kényelmes munkát biztosító külső eszközöket használhassanak, ha ilyenek az adott helyen rendelkezésre állnak. Példaként képzeljünk el egy olyan felhasználót, aki PDA-ját utazás közben, a buszon, a vonaton használja, majd amikor beérkezik munkahelyére, akkor készülékét egyszerűen rákapcsolja az asztalán található monitorra, billentyűzetre és egérre, majd ott folytatja munkáját, ahol abbahagyta. Elképzelhető, hogy a vonat üléseiben is lennének beépített kijelzők, billentyűzetek és pozícionáló eszközök, amelyekre útközben szintén rá lehetne csatlakoztatni a PDA-t, ezáltal a hosszabb utazások alkalmával is kényelmesen lehetne dolgozni.

A fenti problémát általánosítva, a kijelzőt, a billentyűzetet és az egeret szolgáltatásnak elképzelve, körvonalazódik egy olyan megoldás, amivel egy PDA-hoz bármilyen más szolgáltatás hozzákapcsolható lenne. Sőt, az elképzelést tovább általánosítva a rendszeren keresztül gyakorlatilag bármilyen szolgáltatást össze lehetne kapcsolni bármilyen másik szolgáltatással. Így megoldható lenne, hogy egy ad hoc hálózatban szétszórtan található szolgáltatásokat egy olyan személyi hálózattá (Personal Area Network – PAN) lehessen összeszervezni, ahol minden alkalmazás úgy látja, mintha az összes szolgáltatás ugyanazon a számítógépen lenne megtalálható. Erre a rendszerre úgy lehetne alkalmazást fejleszteni, hogy nem kell figyelembe venni a program futása közben fennálló tényleges hálózati struktúrát. Ennek megoldására szükség van egy protokollra és egy egységes programozási felületre (Application Programming Interface – API), amely megoldja a perifériák és más szolgáltatások PDA-hoz illetve egymáshoz történő egyszerű illesztését. Dolgozatunkban egy ilyen protokoll és a hozzá tartozó API megtervezésével foglalkozunk.

A rendszerünket „Blown-up computer”-nek („felrobbantott számítógép”) hívjuk, mert ez az elnevezés tükrözi leginkább az elképzelés lényegét: egy elosztott hardver- és szoftver-erőforrások használatán alapuló, vezetéknélküli rendszert, melyben az alkalmazások és az erőforrások ad hoc jellegűen kapcsolódnak egymáshoz. Az irodalmat kutatva úgy találtuk, hogy a koncepció a *pervasive* avagy *ubiquitous computing* („mindent átható” avagy „mindenütt jelenlévő” számítástechnika) körébe illetve az ad hoc hálózatok területéhez tartozik.

2. Irodalmi áttekintés és problémafelvetés

Ebben a fejezetben áttekintjük a *pervasive computing* alapelveit, valamint a témában jelenleg – egyetemeken és más kutatóközpontokban – futó projekteket, végül az általunk tervezett rendszer alapötletét mutatjuk be.

2.1. Mark Weiser víziója

A *ubiquitous computing*, azaz a „mindenütt jelenlévő” számítástechnika alapelveit Mark Weiser fogalmazta meg. A Xerox cég vezető fejlesztőjeként Weiser több olyan termék-prototípust is kidolgozott, amelyek az általa kigondolt elméleti alapra épültek. 1991-ben megjelent nagyhatású cikkében, melyben a 21. század számítógépéről alkotott vízióját írta le [Weiser91], megfogalmazta, hogy mit is tart ő a követendő fejlődési iránynak. Szerinte az információ technológiának úgy kellene beleivódnia az emberek hétköznapjaiba, ahogyan azt ma az írás teszi: ha látunk egy feliratot vagy jelet az utcán, azt egyből elolvassuk és értelmezzük anélkül, hogy tudatosulna bennünk az olvasás ténye. Véleménye szerint azok a legnagyszerűbb és leghatékonyabb technológiák, amelyek rejtve maradnak a szemünk elől, amiket úgy használunk, hogy nem is tudunk róla. Az általa elképzelt koncepció tulajdonképpen a virtuális valóság paradigma ellentéte, hiszen az éppen egy számítógépen belüli világ felépítését takarja. Ez utóbbi ugyanis hatalmas számítási apparátust mozgósít a világ modellezésére ahelyett, hogy láthatatlanul segítene kényelmesebbé tenni azt a világot, ami már valójában is létezik. Megkülönbözteti saját elképzelését attól az ötlettől is, mely szerint a számítógépek autonóm ügynökök lennének, melyeket felhasználhatnánk céljaink elérésére [Weiser93]. Ennek érzékeltetésére leírt egy példát: képzeljük el, hogy egy nehéz követ szeretnénk felemelni. Ha a hagyományos vagy az ügynökös szemléletet követnénk, akkor behívnánk szupererős segítőnket, és ő felemelné a tárgyat helyettünk. Mark Weiser megközelítését magunkévá téve azonban egyszerűen – anélkül, hogy ez tudatos erőfeszítést kívánna – erősebbé válnánk, és könnyedén a fejünk fölé lendítenénk a követ. Nem állítja azt, hogy a szokásos megközelítés nem lehet hatékony bizonyos esetekben, de álláspontja szerint a cselekvéseinket segítő számítógép maradjon csak a háttérben, figyelmünk forduljon inkább az ember és cselekedetei felé. Meg kell említenünk, hogy Weiser koncepciójának megvalósításához nagyot kellett fejlődnie a mobil számítástechnikának, hiszen többek között az elosztott működés, a mobil kliensek és a szükséges sávszélességek terén a cikkek születésekor még nem volt olyan használható technológia, amely lehetővé tette volna a megálmodott rendszer tényleges implementációját. Azóta azonban – mint ahogy azt a később tárgyalt projektek ismertetésekor láthatjuk – sok olyan technológia született, amelyet be lehet illeszteni a *ubiquitous computing* elképzelésbe.

A dolgozat további részében a *ubiquitous computing* („mindenhol jelenlévő” számítástech-

nika – Mark Weiser), a *pervasive computing* („mindent átható” számítástechnika – IBM), és az *invisible computing* („láthatatlan” számítástechnika) kifejezéseket és magyar megfelelőiket egymás szinonímáiként használjuk.

2.2. Elképzelések és kihívások

A *pervasive computing* által támasztott kihívásokat M. Satyanarayanan – a területet jól átfogó – cikke [Satya01] alapján vizsgáljuk meg. Kitérünk arra, hogy a „mindent átható” számítástechnika milyen rokonságban van az elosztott rendszerekkel és a mobil számítástechnikával.

Az elosztottság a ma is létező számítógép-hálózatok egyik legjellemzőbb tulajdonsága. Az elosztottságra épülő technológiák ma már biztosítják a hálózatban lévő eszközök távoli, biztonságos kommunikációját, a rendszer hibatűrését, a csomópontok folytonos elérhetőségét, valamint a távoli információ-hozzáférést. Természetesen ezek a tulajdonságok a „mindent átható” számítástechnikának is alapvető elemei kell, hogy legyenek.

Mikor a 90-es években megjelentek a vezeték nélküli hálózatok, a kutatóknak megoldást kellett találniuk arra, hogy a funkcionális elosztottsággal járó előnyöket a mobil hálózatokban is ki lehessen használni. Ezen cél érdekében módosították a már meglévő, és alkottak új hálózati technológiákat (Mobile IP [BPT96], ad hoc protokollok [BMJHJ98], TCP vezeték nélküli hálózatokra [BSAK95]), valamint mobil adathozzáférési algoritmusokat, tartalom szerint alkalmazkodó, adaptív (például proxy-kon futó konvertáló) alkalmazásokat, energiatakarékos technikákat, helyzetfüggő rendszer-viselkedési formulákat dolgoztak ki.

A *pervasive computing* célja, hogy egy olyan technológiát teremtsen meg, mely később beleolvad mindennapi életünkbe. Ahhoz, hogy ezt az elképzelést teljesíteni tudja, négy nagyon fontos szempontnak kell megfelelnie.

Először is ki kell tudni használni az intelligens területeket (smart spaces). Ilyen területekről akkor beszélünk, ha egy szokványos környezetbe – például épületben lévő szobába, folyosóra – valamilyen intelligens funkciót ellátó számítógépes infrastruktúrát építünk ki. Ha rendszerünket hatékonyan akarjuk használni, akkor a számítógépes rendszeren keresztül képesnek kell lennünk az épületben zajló események érzékelésére, valamint az épület irányítására, mint például automatikus hőmérséklet- és fényszabályozásra egy ember elektronikus azonosítója alapján. Más szempontból a felhasználó egy adott szoftverének különbözőképpen kell tudnia viselkedni, attól függően, hogy éppen hol tartózkodik.

A második szempont a láthatatlanság: Mark Weiser elképzelése szerint a „mindent átható” számítástechnikának a felhasználói tudat elől teljesen el kell tűnnie. Gyakorlatilag ennek egy értelmes megközelítése, hogy a felhasználókat csak minimálisan terhelje le a rendszer használata. Ha a működés folyamatosan megfelel az emberek elvárásainak, és kevés meglepő eredményt kapnak vissza, egy idő után akár tudatalatti szinten fognak kommunikálni vele.

A harmadik szempont a helyi skálázhatóság: minél több eszköz van egy intelligens területen belül, annál több interakció áramlik a felhasználó és a környezetében lévő entitások között. Ez fogyasztja a sávszélességet, növeli az energiafelvételt, és ezzel akár kellemetlenséget is okozhat a rendszert igénybevevő személyek számára. Még tovább súlyosbítja a helyzetet több felhasználó jelenléte. A korábbi skálázhatósági megoldások nem foglalkoztak a fizikai távolsággal. Egy web kiszolgáló annyi klienst szolgált ki, amennyit tudott, nem törődve azzal, milyen távol vannak tőle. A *pervasive computing*-ban az interakciók sűrűségének csökkennie kell, ha az egyik eszköz távolodik az adott intelligens területtől, különben a rendszer és felhasználója is feleslegesen lenne leterhelve. Ugyanakkor egy mozgó, több ezer kilométer távol lévő felhasználó intézhet kéréseket egy távoli intelligens terület felé.

A negyedik szempont az egyenlőtlen feltételekkel rendelkező területek eltakarása. A *ubiquitous computing* beépülése az infrastruktúrába meglehetősen változatos lesz, és olyan faktoroktól is függ, mint szervezettségi struktúra, gazdasági és üzleti modellek. A teljes beépülés, ha valaha lesz olyan, nagyon távol van még. Ebből kiindulva hatalmas lesz az egyes környezetek intelligenciája közti különbség. Valószínű, hogy egy szoba, egy iroda jobban fel lesz szerelve, mint más helyek. Ezek a különbségek zavaróak lehetnek a felhasználó szemszögéből, és ellentmondanak a rendszer filozófiájának: a láthatatlanságnak. Egyik lehetőség ennek megoldására, hogy csökkentsük a felhasználó által észlelt különbségeket azzal, hogy az ő saját környezeti intelligenciájával kompenzáljuk a „buta” környezeteket. Például egy kapcsolat nélküli működésre is képes rendszer eltakarhatja egy terület hálózati lefedettségének hiányát.

A „mindent átható” számítástechnika gyakorlati megvalósítása és megtervezése rengeteg problémát vet fel, ezek közül a legfontosabbak:

- a felhasználói szándék nyomonkövetése,
- a vezetékes hardver infrastruktúra kihasználása, mobil eszközök tehermentesítése,
- adaptációs stratégiák: applikációk alkalmazkodása a rendszer feltételei alapján, hálózat alkalmazkodása az alkalmazások feltételei alapján (QoS),
- eszközök megfelelő fizikai, teljesítménybeli méretezése, magas szintű energiamenedzsment,
- környezetfüggő viselkedésmód (Context Awareness),
- egyensúly megtalálása a proaktivitás és az átlátszóság között. A proaktivitás itt a jövőbeli események megjóslását jelenti, mely ha nincs kellően megtervezve, akkor sértheti a rendszer láthatatlanságát,
- biztonság és bizalom,

- különböző szintekről származó információk összefésülése. Egy alacsony szintű erőforrás-információt sok esetben hasznos lehet összevetni egy magasabb szintű környezeti információval.

Az „átlátszó” számítástechnika termékeny forrása lesz a következő évtizedek kutatásainak. Nagyon sok területen kell új eredménynek születnie, olyanokon is, melyek nem kapcsolódnak szorosan a számítógép-rendszerekhez. Ilyenek például az ember-gép kapcsolatok, a szoftver ügynökök és a mesterséges intelligencia. Az ezen területekről származó képességeket bele kell építeni azokba a jövőbeli rendszerekbe, melyek már képesek a fejezetben tárgyalt szempontoknak eleget tenni.

2.3. AURA

Az AURA projekt [Aura02] a Carnegie Mellon University (CMU) gondozásában fut. Mivel ezen az egyetemen kifejezetten sok informatikai kutatás folyik, közvetlenül rendelkezésre áll több, mára már kiforrottnak nevezhető technológia, amelyeket a fejlesztők az AURA rendszerébe integrálhatnak. Ez jelentősen meggyorsítja az implementációt, így a koncepció hamarabb testet ölthet.

Az AURA projekt egy futurisztikus világképet definiál. A kutatók a *ubiquitous computing* egyik klasszikus célját tartják szem előtt: olyan rendszert akarnak kifejleszteni, amely meghúzódik a háttérben, és miközben hatékonyan segíti a munkavégzést, nem vonja el túlzottan a felhasználó figyelmét. A fejlesztők fundamentálisan új alapokra helyezik a rendszertervezést, ami azt jelenti, hogy minden szinten (hardver, operációs rendszer, middleware, alkalmazások, felhasználói felület) új koncepciókat próbálnak ki. A rendszer proaktív, azaz a benne található rétegek képesek megjósolni a felsőbb rétegek még ki nem adott kéréseit. A rétegek ezen kívül folyamatosan figyelik a rendszer működési mechanizmusát, és a maximális hatékonyság elérése érdekében folyamatosan „utánhangolják” magukat. Említésre méltó még, hogy a rendszer össze kívánja fogni a rendelkezésre álló hordható (wearable), hordozható (handheld), asztali (desktop) és szerverszintű (infrastructure) számítástechnikai eszközöket, valamint előtérbe helyezi a természetes beszéddel történő utasításokat, ezáltal maximálisan felhasználóbaráttá téve a rendszert. A kutatók célja az, hogy minden felhasználó számára egy személyre szabott, láthatatlan információs aurát biztosítsanak, mely minden pillanatban rendelkezésre áll, helytől és helyzettől függetlenül.

Víziójuk megvalósításához a projekt résztvevői a következő, már meglévő technológiákat használták fel: a Coda fájlrendszer biztosítja az elosztott, sávszélességhez alkalmazkodó és a pillanatnyi szétkapcsolódást toleráló (disconnected operation) fájlkezelést. Az Odyssey az erőforrás-nyilvántartásért és -nyomonkövetésért, valamint a rendszerelemek alkalmazásérzé-

keny adaptációjáért felel. A Spectra a távoli eljáráshívásokat optimalizálja. A tervezők az alkalmazási réteg fölé egy Prism nevű réteget helyeztek el, amely a rendszer fent említett proaktivitását és a rétegek automatikus „utánhangolását” biztosítja. Nagyon érdekes a projekt honlapján megtalálható, általuk készített videó is, amiben a rendszer leendő múködéséről kaphatunk képet. Látható – és hallható –, hogy a vezérlést emberi hanggal kívánják megoldani. Az egyik példa szerint az AURA közli a felhasználóval, hogy üzenete érkezett, amely azonban bizalmas minősítésű, és ezért csak megfelelően zárható teremben lévő kivetítőn tekintheti meg, majd a felhasználót el is kalauzolja a legközelebbi ilyen teremhez. A rendszer kísérleti verziója már működik a CMU campus területén.

2.4. Oxygen

Az Oxygen a *pervasive*, ember-központú számítástechnika körébe tartozó, az MIT-n indított, a DARPA és az Oxygen Alliance által támogatott projekt [Oxygen02]. Olyan felhasználói technológiákat sorakoztat fel, melyek igazodnak az emberek igényeihez. A beszéd és vizuális technológiák segítségével úgy tudunk az Oxygen-nel kommunikálni, mintha az egy igazi személy lenne, ezzel a felhasználó rengeteg energiát takarít meg. Az automatizálást, személyre szabott adathozzáférést, valamint az eszközök együttműködését támogató technológiák rengeteg különböző feladat elvégzését teszik lehetővé. Az Oxygen rendszertechnológiái biztosítják, hogy a felhasználói technológiákat tulajdonképpen tartózkodási helytől függetlenül alkalmazhassuk, akár otthon, az irodában vagy utazás közben. Ezen rendszer a következő területekre fókuszál:

- az elosztottság és mobilitás – az erőforrások és szolgáltatások globális elérése,
- szemantikus tartalom – amire gondolunk, nem csak amit mondunk,
- formálódás és változás – a legfontosabb egy dinamikus világ számára,
- személyes információ – biztonság, személyes interakciók.

Az Oxygen-t kétféle eszközön keresztül érhetjük el. Telepített Enviro21-vel (E21) a környezetünkben, vagy hordozható Handy21-ek (H21) segítségével. Ezek az univerzálisan hozzáférhető eszközök támogatják a számításokat, a kommunikációt és érzékelésre is képesek. Az E21-ek irodákba, épületekbe, lakásokba, járművekbe telepített elemek, melyekhez helyhez kötött funkciókat tudunk társítani, például egy szoba hőmérsékletét tudjuk figyelni és módosítani, garázsajtót becsukni még akkor is, ha attól több ezer kilométer távolságra vagyunk. Ezek a mobil eszközök nagy számítási erőt kell, hogy képviseljenek, hiszen hang- és képfeldolgozási feladatokat is el kell látniuk. A H21-ek lehetnek mobiltelefonok, csipogók, rádiók, kamerák, PDA-k

(Personal Digital Assistant). Energiatakarékossági okokból a H21-ek képesek feladatokat átadni az E21-ek számára. A rendszer megálmodói szerint az emberek mindennapi életük során felmerülő feladataikat az Oxygen segítségével fogják megoldani. A rendszer által biztosított univerzális hálózat és számítási erő segítségével a különálló kis eszközök egymással kommunikálva és együttműködve hajtják végre a szükséges műveleteket. A résztvevő eszközök megtalálják a szükséges erőforrásokat, összekötik őket, ellenőrzik a folyamatokat és reagálnak a változtatásokra.

Az N21 hálózat az önmagukat azonosító mobil eszközök dinamikusan változó beállításait támogatja. Az elemeket és a szolgáltatásokat nem csak helyzetük, hanem funkcióik szerint is tudjuk azonosítani. Az információkhoz és a szolgáltatásokhoz biztonságosan és személyre szólóan tudunk hozzáférni, így a rendszert akár magánéletünkben is alkalmazhatjuk. Az N21-ek többféle, alacsony teljesítményigényű kommunikációs protokollt támogatnak, melyekkel kis és közepes hatótávolságú hálózatokat tudunk menedzselni, együttműködő régiókat tudunk létrehozni, módosítani és megszüntetni. Az Oxygen szoftver architektúrája az eszköz- és hálózati szint felett vezet be változtatásokat. Úgy lett megalkotva, hogy a projekt céljait a ma rendelkezésre álló szoftverek szolgáltatásaival valósítsa meg. Legfőbb feladata, hogy levegye a felhasználó válláról azt a terhet, melyet egy *pervasive* rendszer irányítása és megfigyelése okoz.

Nézzük meg például, hogy lehet egy üzleti konferenciát megszervezni az Oxygen segítségével. Helena felhívja párizsi munkahelyéről Ralph-ot New Yorkban. Ralph E21-e, mely a telefonjához csatlakozik, felismeri Helena számát. Franciául válaszol vissza a lánynak, hogy Ralph éppen a nyári szabadságát tölti és megkérdi, mennyire sürgős a hívása. Az E21 többnyelvű beszédszintetizátora és automatizálási rendszere, melyet Ralph bekonfigurált, hogy kezelje a sürgős hívásokat olyanoktól, mint Helena, felismer a lány válaszában egy kulcsszót, és továbbítja a hívást a Ralph szállodájában lévő H21-ére. Amikor Ralph Helenával beszél, úgy dönt, hogy George-ot is bekapcsolja a beszélgetésbe. Megbeszélik mindhárman, hogy jövő héten találkoznak Párizsban. Megkérik automatikus naptáraikat a saját E21-ükön keresztül, hogy egyeztessék az időbeosztásaikat, és ellenőrizzék, hogy mikor van repülőjárat New York-ból és Londonból Párizsba. Következő kedd délelőtt 11 óra jónak tűnik, így mindhárman rábólintanak. Az automata rendszer végrehajtja a szükséges helyfoglalásokat.

2.5. Portolano

A Washington-i Egyetemen dolgozó Portolano csoport is indított egy projektet, melyben a „láthatatlan” számítástechnika által nyújtott lehetőségek megvalósítását tűzték ki célul. A fejlesztők a feladat megfogalmazásakor különböző kutatási részterületeket jelöltek ki. A csoport véleménye szerint [Porto99] megfelelő eredmény eléréséhez szükség van egy teljesen újfajta, adaptív felhasználói felület kidolgozására, amely érzékeli a felhasználó cselekedeteit, képes

megjósolni a felhasználó szándékait, valamint hatással vannak rá a rendszert körülvevő környezet jellemzői. A feladat megoldásához vizsgálni kell horizontálisan rétegződő és megfelelő felderítő mechanizmussal rendelkező hálózat alapú szolgáltatások működését. Találni kell egy alkalmas útvonalválasztó rendszert, valamint ki kell építeni egy adatcentrikus aktív hálózati infrastruktúrát [PortoWeb99]. Az új eszközök, alkalmazások és szolgáltatások bevezetésének elősegítéséhez könnyen használható fejlesztési és telepítési környezeteket kell kialakítani.

A projekt fejlesztői úgy gondolták, hogy a felhasználók majd egy láthatatlan felületen keresztül fogják elérni a rendszert. A koncepció szerint explicit parancsok kiadása helyett a vezérlés itt cselekvéssel történik. A lényeg, hogy a felület egy jól tervezett eszköz rejtett része legyen, amely úgy illik bele a használó feladatának körülményeibe, hogy szinte láthatatlan marad. Ezen változó felületek alapjául teljesen új generációs apró szenzorok szolgálhatnának, melyeket életünk mindennapjaiban használt tárgyakba építenénk bele (Invisible User Interface).

Sok erőforrás-felderítéssel kapcsolatos kutatás indult, mely szintén nagyon fontos eleme egy ilyen koncepciónak, például az RDP (Resource Discovery Protocol) és a Sun Microsystems által készített Jini rendszer. Mindketten egy lokális szolgáltatás adatbázist tartanak nyilván, ahová a szolgáltatások regisztrálják magukat. Mikor egy kliens lekérdezi a környezetében létező szolgáltatásokat, az RDP visszaad egy egységes erőforrás nevet, míg a Jini a klienseket egy Java objektumként megvalósított proxyval látja el (Resource Discovery). A folyamatosan változó kapcsolatok és az ad hoc hálózatok miatt arra is szükség lehet, hogy az adat saját maga találja meg a szolgáltatásokat az őt hálózatra küldő alkalmazás segítsége nélkül. Emiatt kell egy olyan hálózati képesség, mely egy csomóponton képes végrehajtani az adatcsomagban lévő kódot, ezáltal az adott eszközön olyan felderítő függvényeket tud a kód meghívni, melyekkel kideríthetni, hogyan juthat el a kívánt szolgáltatáshoz, és melyik a legoptimálisabb útvonal a cél felé (Data-Centric Networking). Az adatok biztonságos célbajutásához szükség lehet egy ellenőrzött replikációs és átirányítási mechanizmusra is. Tehát például egy adat megtalálhatja saját maga másolatát a hálózaton (több példány van belőle, természetesen véges élettartammal), párhuzamosan különböző szolgáltatásokhoz irányítva. Például egy fényképet rendelhetünk egyszerre egy fényképalbumhoz, egy naptárhoz és egy nyomtatóhoz is. Azonban ha egy fényképet elküldünk egy fényképalbum-szolgáltatáshoz, nyilván fontos, hogy az a megfelelő formátumban legyen tömörítve, így ha valahol nem áll rendelkezésre a megfelelő tömörítő szoftver, annak is a kívánt helyre kell vándorolnia. Mindezek mellett az sem árt, ha az adatokat igénytől függően el tudjuk tárolni (például egy kamera a fényképeket, diavetítés céljából). Nehézségekbe ütközik az adatfogadó szolgáltatás által küldött visszaigazoló (ACK) üzenet célba juttatása is, hiszen az adat forrása addigra már eltűnhetett a hálózatból, vagy egy teljesen új helyre vándorolhatott. Tehát szükség van egy elosztott helymeghatározási szolgáltatásra is. A rendszer egyik legnagyobb előnye a szolgáltatások elosztott, globális elérése. A felhasználó igénye szerint az erőforrások

helytől, eszköztől függetlenül elérhetők, nem csak az emberek, hanem a hálózatban résztvevő összes eszköz számára is (Distributed Services).

2.6. Endeavour

Az Endeavour projectet a Berkeley egyetem indította azzal a céllal, hogy a kutatók által kifejlesztett rendszer segítségével az emberek sokkal kényelmesebben juthassanak az őket érdeklő információkhoz, könnyebben használhassanak eszközöket, és hatékonyabban tudjanak kommunikálni egymással [Endeav99].

A munka mozgatórugója tulajdonképpen a számítástechnika emberközelivé varázslása. A kutatók a személyre szabott információmenedzsmentet tartják a leglényegesebb alkalmazási területnek. Igyekeznek figyelembe venni azt a tényt, hogy az emberek tudást hoznak létre, nem pedig egyszerű adattömeget. A projekt résztvevői fokozottan felhívják a figyelmet arra, hogy az információ technológia vívmányai mindig rendelkezésre kell, hogy álljanak: az információhoz való hozzáférés nem hiúsulhat meg egy megszakadt összeköttetés, egy lefagyott számítógép vagy egy inkonzisztens információhalmaz miatt. Ha megvalósul egy ilyen mértékű rendelkezésre állás, az nagyban hozzájárulna az emberi produktivitás növekedéséhez. A rendszer lelke egy olyan *Information Utility* (IU), amely megfelelően skálázható akár egy világméretű hálózat igényeinek kielégítésére is. Ennek legforradalmibb tulajdonsága az ún. folyékony szoftver (fluid software) támogatása. Ez azt jelenti, hogy a számítási, adattárolási és egyéb megszokott funkciók automatikusan elosztják magukat a rendelkezésre álló eszközök között. Ehhez természetesen mobil kódra van szükség, hiszen ezeknek a programoknak teljesen különböző platformokon kell futniuk. „Nomád” adatoknak (nomadic data) olyan adatokat nevezünk, melyek képesek önmaguk sokszorosítására, és adott esetben oda tudnak folyni a hálózatban, ahol szükség van rájuk. Segítségükkel javítható a rendszer teljesítőképessége és rendelkezésre állása is. Az IU tervezése során a kutatók úgy igyekeznek eljárni, hogy koncepciójuk használható legyen a maitól radikálisan eltérő hardverrendszerek esetén is. A tervezők nagy hangsúlyt helyeznek az újrafelhasználható komponens alapú tervezésre és a formális módszerekkel történő verifikációra is. Egy másik jelentős innovációt tesz lehetővé a mindenhol egyszerre jelenlévő IU azon különleges képessége, hogy eddig soha nem tapasztalt mértékben tud információt gyűjteni az emberi cselekvésekről, és ezeket hatékonyan rendezi is. Ezáltal automatikusan implicit információt tud kinyerni az emberek egymás és más – mesterséges – információforrás iránt tanúsított viselkedéséről, azaz a rendszer tanulhat és egyre jobban alkalmazkodhat az emberi viselkedéshez.

2.7. Speakeasy

A mobil, „mindent átható és mindenütt jelenlevő” számítástechnika gondolata abból indul ki, hogy egy olyan világban élünk, ami be van hálózva intelligens és egymással kapcsolatban álló eszközökkel. Ahhoz, hogy ezek egy csoportja egymással kommunikálni tudjon, szükséges, hogy egymásról előzetesen már valamilyen ismeretekkel rendelkezzenek. Általánosságban ismerniük kell partnerüket, tudniuk kell az együttműködés részleteiről, beleértve, hogy mikor, mit és hogyan mondjanak. A *Speakeasy* projekt egy úgynevezett „egymáshoz illeszkedő számítástechnikát” (recombinant computing) képzel el, ahol az egymással kapcsolatot felépítő eszközök előzetesen csak a legszükségesebb ismeretekkel rendelkeznek egymásról [Speakeasy02]. Támogatást nyújt ad hoc, felhasználó által konfigurált összeköttetések létesítéséhez; sablont biztosít adatküldéshez, felhasználói vezérléshez, új szolgáltatások felderítéséhez ill. környezetfüggő viselkedésmódhoz.

Két egymással kapcsolatban álló eszköz között mindig van egy kezdeti minimális megállapodás, amely tartalmazza kettejük kommunikációjának formáját – ezek az általuk exportált interfészek, amelyet a másik fél ismer. Például egy webszervernek és egy böngészőnek a kommunikációhoz ismernie kell egy közös protokollt (HTTP) és egy közös adatformátumot, illetve jelentést (általában HTML vagy XML), amiről előzetesen meg kell állapodniuk. A mai világban ez tipikusan előre rögzítve van az adott alkalmazásban. A *Speakeasy* fejlesztői állítják, hogy erre a jövőben nem lesz szükség, elég, ha egymásról csak elenyésző ismeretekkel rendelkeznek. Egyúttal felteszik a kérdést, hogy milyen szintű előzetes megállapodás szükségeltetik.

Definiáltak egy rendszert, amelyben ezen előzetes ismereteket a lehető legkisebb mértékre redukálták. Szerintük három építőelem kell egy együttműködéshez. A kapcsolathoz először is szükséges egy rögzített „tartományfüggetlen interfész” (domain-independent interface), amely minden, a rendszert ismerő készülékben benne van. Rögzített tartománytól függő interfészek szerintük nem célravezetőek, mert ezek minden egyes tartományra egyediek, ebből kifolyólag nagyon sok van belőlük, és sokan nem értik meg őket. A tartományfüggetlenség tehát a kommunikáció egységességét biztosítja. Másodszor, a flexibilitás és dinamizmus nyújtására mobil kódokat alkalmazhatunk. Ezek használatakor egy kérés kiadásával a hálózatban egy futtatható kód egy adott eszközhöz vándorol, azzal a céllal, hogy kibővítse annak képességeit. Ez felhasználható arra is, hogy kiterjessze a rendelkezésre álló interfészt, azáltal, hogy a mobil kód ismer és használni tud egy, az entitáson addig nem létező interfészt. Például egy nyomtatóról elmehet egy mobil kód egy szövegszerkesztőt futtató eszközhöz, ami ezután nyomtatni is fog tudni, vagy egy projektorról elvándorolhat a kód egy prezentációt futtató PDA-ra, amely a kód segítségével használni tudja majd a kivetítőt és meg fogja tudni jeleníteni az előadás diáit. Végezetül harmadikként szükség van „felhasználó-a-folyamatban” beavatkozásra (user-in-the-loop interaction). A rendszer kitalálói szerint egy applikációtól nem várható el, hogy megértse

a vele kommunikáló entitás által küldött általános adatot, nem értelmezheti annak jelentését. Ehhez mindenképpen szükség van a felhasználóra, aki például tudja, hogy nyomtatásnál mit jelentenek a *duplex* és *landscape* fogalmak.

Minden *Speakeasy* eszköznek illetve szolgáltatásnak meg kell valósítania pár rögzített interfészt. Ezek négy kategóriába sorolhatók:

1. adatátvitel: megmondja, hogyan adjon át egy eszköz információt a másiknak
2. kollekció: hogyan vannak bizonyos eszközök szolgáltatásfelderítés vagy egyesítés céljából csoportosítva
3. metaadat: az entitások hogyan mutatják meg másoknak a magukról szóló leírást
4. vezérlés: miként engedik meg az entitások más felhasználóknak (és entitásoknak), hogy rajtuk változtatást eszközöljenek

A rendszer kiötlői két tucat interfészt valósítottak meg a legkülönfélébb célokra, szem előtt tartva, hogy azok minél egyszerűbbek legyenek. Többek között létezik internet rádió, UNIX és Windows NT fájlrendszert ismerő fájlszerver, csevegőprogram, nyomtató, mikrofon és hangszóró alkalmazás is.

2.8. Problémafelvetés

A fentiekben ismertetett kutatásokat végző csoportok elképzeléseik egy részét már megvalósították, azonban a megoldások messze állnak még a teljes működőképességtől. Nagy kérdés, hogy – ha majd a rendszereket egyszer teljesen implementálják – az emberek hozzá tudnak-e majd szokni használatukhoz. Ez egy érdekes paradoxon, hiszen Mark Weisernek és a témát most kutatóknak is egyértelmű szándéka az volt, hogy mindennapi életünk során ne kelljen külön energiát fordítanunk az általuk alkotott renszer használatára, hanem ez szinte ösztönös legyen.

Az AURA előnye az, hogy több, már valóban működő technológiát használ fel, és a rendszer tesztverziója működik; ellenben a projekt honlapján lévő videó olyan beszédfelismerő-rendszer létezését sugallja, amely képes kötetlen szövegből kiszűrni a vezérlési információkat, valamint a rendszer proaktivitásának megvalósíthatósága is kérdéseket vet fel.

Az Oxygen rendszer nagy hangsúlyt fektet a sokféle mobil eszköz támogatására és a felhasználóbarát elérésre, ugyanakkor olyan mértékű intelligenciát szeretne az egyes készülékekbe építeni, amely – véleményünk szerint – napjainkban még nem áll rendelkezésre.

A washingtoni Portolano csoport két fő célt tűzött ki maga elé: egy használható, adaptív felhasználói felület kidolgozását és intelligens, adatcentrikus hálózatok kifejlesztését. Ezek nagyon érdekes informatikai területek, de nem nevezhetők még kiforrottnak.

A személyre szabott információmenedzsment áll az Endeavour kutatói érdeklődésének középpontjában, valamint forradalmi újításokat terveznek a vándorló számítási, adattárolási és egyéb funkciók terén is. Az elképzelés érdekes, a kutatási eredmények ígéretesek. Említésre méltó az a tény, hogy e projekt – hozzáférhető – dokumentációja a legrészletesebb az összes közül.

A Speakeasy-koncepció számítástechnikai eszközök közötti minimális interfészek specifikálására, mobil ágensek használatára és mobil programkód fejlesztésére fókuszál. A cél használható, ugyanakkor minimális méretű interfészek definiálása, azonban ezek felépítése és tartalma – ugyanúgy, mint a rendszer biztosági aspektusai – még kutatásra várnak.

Érzékelhető, hogy a rendszerek megálmodói egészen futurisztikus környezetet szeretnének kialakítani, és elképzeléseik egyelőre távol esnek a megszokott emberi gondolkodásmódtól. (Ez alól talán a Speakeasy jelenthet kivételt.) Ezek a projektek mind valamiféle fejlett mesterséges intelligencia kifejlesztését célozzák meg, és a dokumentációkat olvasva világosan lehet érezni, hogy ennek megvalósítása nem lesz egyszerű feladat.

Mi egy jobban megfogható problémát jártunk körül. Rendszerünk tervezésekor az a cél vezérelt minket, hogy egy olyan életképes elképzelés szülessen, amely a felhasználók mai szemléletmódjával jól összeegyeztethető, de ezzel együtt *ténylegesen* megkönnyíti a munkát, és kiterjeszti egy hordozható, kisméretű számítógép lehetőségeit. Az irodalmi áttekintésben megvizsgált projektekkel ellentétben egy kisméretű, jól funkcionáló és koncepcionálisan egyszerű rendszert terveztünk meg.

Rendszerünk működéséhez szükség van az ad hoc hálózatok területén született megoldásokra. Ebben a témakörben sok különböző rádiós technikát, újfajta adatkapcsolati rétegeket és speciális routing algoritmusokat fejlesztettek ki. Az általunk kidolgozott rendszer épít egy már meglévő és működő ad hoc hálózatra, de nem függ a hálózat konkrét megvalósításától, a használt protokolloktól. A *Blown-up* koncepció lényege az, hogy az ad hoc hálózatot egy olyan személyi hálózattá alakítsuk, ahol a hardver- és szoftver-erőforrások ad hoc módon kapcsolódnak egymáshoz, egy elosztott rendszert alkotnak. Ennek érdekében specifikáltuk a protokollunkat, és definiáltunk hozzá egy programozási felületet (API-t), melyen keresztül a programozók elérhetik a rendszer szolgáltatásait.

2.9. A dolgozat felépítése

A 3. fejezetben áttekintést adunk koncepciónk lényegéről. Példákkal is megvilágítjuk a rendszer működését, és leírjuk azt, hogy miért gondoljuk hasznosnak az általa nyújtott szolgáltatásokat. A fentieken kívül ismertetjük még a dolgozatban használt terminológiát.

A 4. fejezetben részletesen tárgyaljuk az általunk tervezett protokollt. Leírjuk az egyes rétegek jellemzőit, az általuk megvalósított funkciókat, felsoroljuk a rétegek közti párbeszé-

det biztosító üzeneteket, valamint részletezzük a fontosabb kommunikációs fázisokat. Végül specifikáljuk az egész rendszert használhatóvá tévő programozói felületet (API – Application Programming Interface).

A 5. fejezetben foglalkozunk az eddig végzett implementációval, és elemezzük ennek szerepét a koncepció egészének kialakításában.

Az utolsó, azaz 6. fejezetben összefoglaljuk az általunk végzett munkát, és megemlítjük a jövőre vonatkozó elképzeléseinket.

3. A *blown-up* koncepció

A mobil számítástechnikai eszközök terjedése nagyban befolyásolhatja a számítástechnikáról eddig kialakult képet, ugyanis a hordozható számítógépek és egyéb hordozható elektronikus eszközök nem csupán rögzített társaik vezetéknélküli másai, rendelkeznek más, önálló tulajdonságokkal is. Például ezen eszközök hálózatba szervezése nem olyan triviális dolog, mint vezetékkel összekötött, fixen telepített készülékek esetén. Így a vezetéknélküli, ad hoc módon szerveződő hálózatok manapság népszerű kutatási témának számítanak. Éppen ezért jelentős eredmények is születtek ezen a területen, például a rádiós technológiák terén meg kell említenünk a IEEE 802.11-et [WLAN99], a HiperLAN2-t [HLAN2] és a Bluetooth-t [JH98, BBSpec]. A közegelérési alrétegekbeli (Medium Access Control – MAC) megoldások közül a MACA-t (Medium Access Control with Collision Avoidance) [Karn90] említenénk meg, amit a 802.11 használ ad hoc módban. Az IETF-en (Internet Engineering Task Force) külön munkacsoport foglalkozik a mobil ad hoc hálózatokkal. Az ad hoc útvonalválasztó algoritmusok közül az ismertebbek az AODV (Ad hoc On-demand Distance Vector), a DSR (Dynamic Source Routing) [DPR00], DSDV (Destination Sequenced Distance Vector) [PB94], és a TORA (Temporarily Ordered Routing Algorithm) [PC97, BMJHJ98].

A *Blown-up* rendszer feltételezi, hogy azok az eszközök amelyeken fut, egy ad hoc hálózatot alkotnak és így képesek egymással kommunikálni. Egy ad hoc hálózat önmagában nem alkot számítástechnikai környezetet, csak egymással kommunikálni képes vezetéknélüli eszközök halmaza. Rendszerünk ezt a környezetet hivatott megvalósítani, arra ad megoldást, hogy az ad hoc hálózatban résztvevő eszközökön futó szolgáltatásokat, alkalmazásokat egyszerűen össze lehessen kapcsolni egymással, egy olyan PAN-ná, ahol minden alkalmazás úgy látja, mintha az összes szolgáltatás ugyanazon a számtógépen lenne megtalálható. A *Blown-up* rendszer az egész ad hoc hálózatot egy számítógépnek láttatja, így megkönnyítve a programozók munkáját, akik az ad hoc hálózatokra illetve PAN-okra készítenek alkalmazásokat. A programozóknak a rendszert használva nem kell foglalkozniuk a dinamikusan változó ad hoc hálózat tényleges felépítésével, így egyszerűbben fejleszthetnek elosztott alkalmazásokat. Az alkalmazások elosztott működéséhez lényeges, hogy a rendszer egységesen kezelje a különbözö perifériákat, alklamazásokat.

Amikor a perifériákat említettük, nem tértünk ki arra, hogy ezek milyenek lehetnek. Valószínűsíthető, hogy körük idővel egyre bővülni fog a személyi számítógép ma létező perifériáihoz képest. Éppen ezért a tervezendő rendszernek rugalmasnak kell lennie, hogy a teljesen új eszközöket is bele lehessen majd illeszteni. Fontos különbség a PC-s világ és a mi elképzelésünk között, hogy míg a PC konfigurációja a futás közben csak minimálisan változhat, addig egy személyi ad hoc hálózatban minden pillanatban történhet változás. Az egyik percben még

az egyik ember használ egy bizonyos egeret, a másikban pedig ugyanaz az eszköz egy másik személy PAN-jának része lesz.

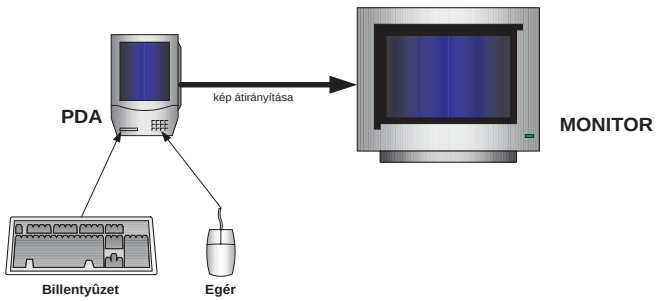
A *Blown-up* rendszer nem csak a perifériákhoz való hozzáférést oldja meg. A személyi hálózat fejlettebb eszközei szoftveres szolgáltatásokat is nyújthatnak a felhasználó illetve a többi eszköz számára. Például titkosításhoz szükséges kulcsokat generáló alkalmazást biztosíthatnak olyan eszközök számára, amelyekenek nincs elegendő számítási kapacitása e kulcsok előállításához, de szükségük van ezekre a biztonságos kommunikációhoz. Elképzelhető zene- és videólejátszó szolgáltatás, amik a korlátozott erőforrásokkal rendelkező eszközök (például PDA) helyett elvégzik a nagy teljesítményű processzort igénylő MPEG (Motion Picture Experts Group) [MPEG] dekódolást.

Annak érdekében, hogy hatékonyan lehessen alkalmazásokat fejleszteni egy ilyen architektúrára, lehetővé kell tenni a rendszerfunkciók elérését a programozók számára. A rendszerünkhöz alkotott programozói felület (API) ezt a célt szolgálja, a benne definiált függvények használatával a fejlesztők *Blown-up* kompatibilissé tehetik programjaikat.

3.1. Esettanulmány

Az alábbi példával szeretnénk bemutatni, hogy milyen rendszert lehetne felépíteni a *Blown-up* rendszer koncepciójára.

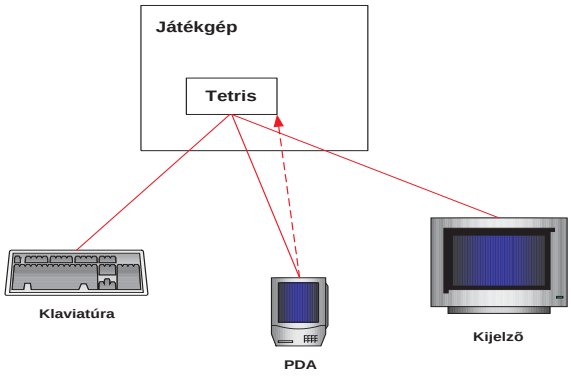
Zoltán reggel megérkezik az irodájába. Mivel a munkahelyéig tartó úton végig a PDA-ján dolgozott, belefáradt, hogy a kis méretű billentyűzetet kell használnia és a kis méretű képernyőt kell néznie. Elindítja PDA-ján a *Blown-up* vezérlő alkalmazást és megtekinti a szobájában található eszközök által felajánlott szolgáltatásokat. Utasítja a vezérlő alkalmazást, hogy építsen ki kapcsolatot az asztalán lévő monitorral, billentyűzettel és egérrel, hogy folytatni tudja a vonaton, utazás közben elkezdett prezentáció kidolgozását (1. ábra). Azt követően, hogy a vezérlő kiépíti a szükséges összeköttetéseket, Zoltán kényelmesebben folytathatja munkáját a nagyobb méretű eszközök segítségével. Miután befejezte a prezentációt, Zoltán egy kis felü-



1. ábra. Kényelmes munkakörnyezet PDA-hoz

dülésre vágyik. Eszébe jut, hogy a szomszéd szobában dolgozó munkatársánál, Andrásnál, van egy játékgép, amelyen Tetris játék is fut. Ezért Zoltán fogja a PDA-ját, átmegy a kollégájához, és „kölcsönkéri” a Tetris-t. Ez úgy valósulhat meg, hogy a játékkonzol fizikailag egy külön eszköz, amely eddig András személyi hálózatának részét képezte, de mostantól Zoltán PAN-jához kapcsolódik. Ehhez teljesülnie kellett annak a feltételnek, hogy András nem használta éppen akkor a játékot. Zoltánnak annyit kell tenni, hogy a PDA-ján futó vezérlő alkalmazás segítségével kiválasztja kedvenc játékát, és összeköti a PDA képernyőjével, valamint egy, a szobában található billentyűzettel. Ebben az esetben játékszoftver a játékgépen fut, adatbeviteli eszközként az András szobájában lévő rádiós billentyűzet szolgál, és az alkalmazás képe Zoltán PDA-jának képernyőjére kerül

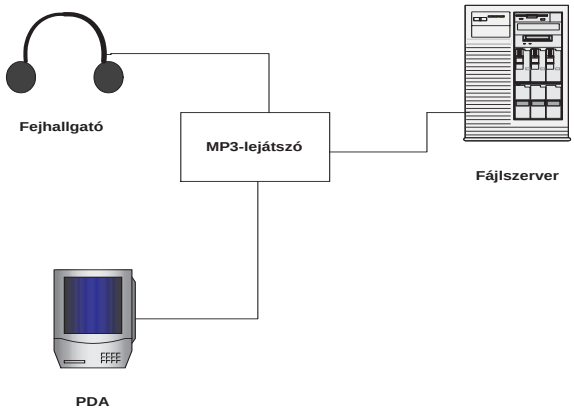
Zoltán András nagyméretű képernyőjét is használni szeretné, így a vezérlőfelületen kiadja a parancsot, hogy PDA-ja kapcsolódjon rá a nagy képernyőre is, hogyha majd Andrásnak nem lesz rá szüksége, akkor játék közben egyszerűen átkapcsolhasson rá. Amíg András a nagy képernyőn dolgozik, addig Zoltán a PDA kijelzőjét használja, majd amikor András már lemond a monitor használatáról, akkor átengedi Zoltánnak a nagy megjelenítőt (2. ábra).



2. ábra. Itt az ideje játszani!

Később Zoltán zenét szeretne hallgatni, így az András íróasztalán lévő mp3-lejátszó berendezés vezérlő felületét a PDA-jára irányítja. A szoba sarkában lévő fájlszerveren levő mp3 dalokat szeretné a lejátszóba betölteni, ezért ezeket is egymáshoz rendeli. Már csak egy fejhallgatóra van szüksége – ilyen is van a szobában –, amit összeköt a zenedoboz kimenetével, és máris elkezdheti a zenehallgatást (3. ábra).

Egy kis idő elteltével Andrásnak is megjön a kedve a játékhoz. Betársulna kollégájához egy kétszemélyes Tetris-csatára, amihez szüksége van egy játékirányító eszközre. Mivel a szobában csak egy billentyűzet van, és azt Zoltán használja, így előveszi mobiltelefonját, és ennek gombjait használja majd a játék vezérlésére. Ennek érdekében Zoltán leállítja az egyszemélyes Tetris játékot, a PDA-ján lévő vezérlővel összeköti András *Blown-up* kompatibilis mobiltelefonját a

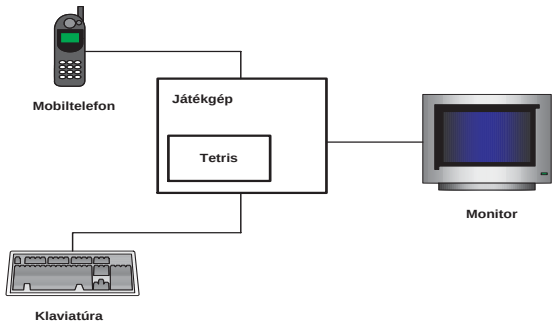


3. ábra. Zenehallgatás

Tetris-alkalmazással. A Zoltán által eddig használt klaviatúra és képernyő szintén kapcsolatban marad a játékkal. Így már ketten is tudnak játszani egymás ellen (4. ábra).

A rendszer tervezésének szempontjából érdekes lehet egy olyan eset, amikor Zoltán egyszerre kívánja igénybe venni az mp3-lejátszót és a Tetris-játékot. Ha feltételezzük, hogy az mp3-lejátszónak van képernyőre irányítható kezelőfelülete is, akkor egyszerre láthatná ezt a képet és a Tetris képét is. Felmerül a kérdés, hogyha Zoltán váltani szeretne a két alkalmazás között, akkor azt hogyan teheti meg. Erre a kérdésre a választ a későbbi fejezetekben adjuk meg.

Az esettanulmányból két lényeges, a rendszerrel szemben támasztott követelmény derül ki. Az egyik az, hogy a használhatóság szempontjából kulcsfontosságú a felhasználó számára minél könnyebben és gyorsabban elvégezhető kapcsolat-konfiguráció – senki sem akar sok időt tölteni az eszközei közötti kapcsolatrendszer felépítésével. Ha túl bonyolult a kapcsolatok beál-



4. ábra. Indulhat a kétszemélyes Tetris. . .

lítása, akkor az embereknek nem lesz kedve használni a rendszert. Ennek érdekében valószínűleg csak minimális felhasználói interakciót érdemes elvárni, és néhány hasznos automatizmust kell beleépíteni az architektúrába. Fontos lehet még a biztonságos adatcsere biztosítása is, nem engedhetjük meg, hogy egy idegen személy, PAN-unk hatósugarán belülre érve, lehallgassa az ad hoc hálózatban az entitások között folyó párbeszédeket. A másik nagyon fontos elvárható tulajdonság, hogy a programok könnyen tudjanak kommunikálni az egyes eszközökön elszórva lévő alkalmazásokkal, valamint a perifériákkal. Ehhez szabványos felületet kell nyújtani az entitások felé, amelyek így hatékonyan végezhetik a szükséges információcserét.

3.2. A *Blown-up* rendszer és a *pervasive computing*

Az eddig leírtakból nem feltétlenül érzékelhető, hogy a Blown-up rendszer pontosan milyen vonásaiban követi a *pervasive computing* irányelveit. Először a hasonlóságokra térünk ki, aztán a különbségeket tárgyaljuk.

Az általunk tervezett rendszer elrejti a különbségeket az alkalmazások és a perifériák között azáltal, hogy közösen, szolgáltatás néven kezeli őket. Amennyiben valaki alkalmazásokat szeretne igénybevenni, akkor nem kell tudnia, hogy azok fizikailag melyik eszközön futnak, ezáltal helyfüggetlen lesz az alkalmazások elérése. Fontos megemlíteni, hogy a rendszer részévé válhatnak rögzített számítógépek és egyéb eszközök is, így nagyobb tárterületet vagy számítási kapacitást igénylő műveleteket is végrehajthatunk a rendszerarchitektúra keretein belül. Előnyös tulajdonsága a koncepciónak, hogy segítségével olyan programok készülhetnek, melyek írásakor a fejlesztőnek nem kell pontosan ismernie a program „futási környezetét”. Ez alatt azt értjük, hogy nem kell törődniük a futás idején fennálló, konkrét ad hoc hálózat felépítésével. Másrészt úgy lehet alkalmazásokat fejleszteni ad hoc környezetre, mint hagyományos számítógépre. Egész egyszerűen a programok csak akkor fognak futni, ha ehhez megfelelő környezet alakul ki. Rendszerünk a felhasználó figyelmét csak rövid ideig és kis mértékben vonja el a tényleges munkavégzéstől, azáltal, hogy csak minimális beavatkozást igényel a szolgáltatások összekapcsolásához.

A rendszer használata során sok olyan szituáció adódhat, amikor egyszerűen szükséges, hogy a felhasználó vezérelje az egyes entitások összekapcsolódását. Tipikusan ilyen egy képernyő vagy egy billentyűzet igénybevétele: az ember nyilvánvalóan meg akarja mondani, hogy pontosan melyik monitorra óhajtja átirányítani digitális személyi asszisztensének kimenetét vagy, hogy pontosan melyik billentyűzeten kíván gépelni. Ezekben az esetekben nem követjük szorosan a láthatatlan számítástechnika alapelvét, vagyis megköveteljük az emberi beavatkozást a kapcsolatok kialakításánál.

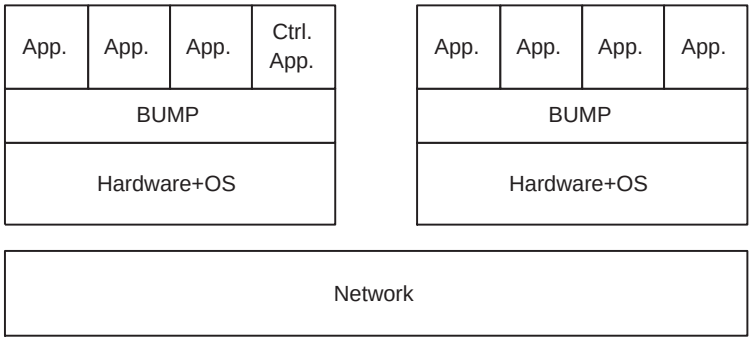
Ez természetesen szükséges az előbb említett okok miatt – (egyelőre) semmilyen intelligencia nem tudja kitalálni, hogy a felhasználó piros vagy kék egérrel szeretne dolgozni. A

ubiquitous computing témájában futó projektek nagy részének egyik elsődleges célja, hogy az informatikai infrastruktúra szinte láthatatlanul szolgálja az embereket, ösztönösen lehessen a rendszert használni. Nekünk ez nem volt célunk, inkább egy ad hoc hálózatokban jól használható, a felhasználó lehetőségeit kiterjesztő és kényelmesebb munkát biztosító rendszer tervezését tartottunk szem előtt.

4. Rendszerarchitektúra és protokollverem

4.1. Rendszerfelépítés és tervezési döntések

A *Blown-up* rendszert *middleware*-nek terveztük, az alkalmazások, illetve az operációs rendszer és hardver közé (5. ábra). A *Blown-Up Micronet Protocol* (BUMP) a nagyobb teljesítményű eszközökön az operációs rendszer részeként vagy afölött, míg a gyengébb képességű – operációs rendszer nélküli – eszközökön (pl. egér, billentyűzet) közvetlenül a hardvert elérve működhet.



5. ábra. Rendszerarchitektúra

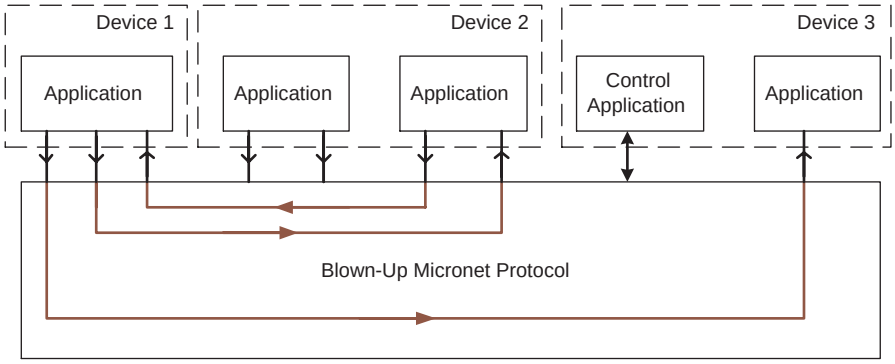
A *Blown-up* rendszer segítségével az eszközökön futó alkalmazások a hálózatban található többi alkalmazás által nyújtott szolgáltatást úgy látják, és tudják igénybe venni, mintha azokat közvetlenül a saját eszközük nyújtaná (6. ábra). Az esettanulmányban vázolt Tetris (3.1. fejezet) játék például úgy látja a szobában található képernyőt, mintha azzal ő rendelkezne.

A rendszer tervezésekor úgy döntöttünk, hogy az egyes eszközökön futó alkalmazások adatbeviteli és -kiviteli csatornáik alapján jelenjenek meg a rendszerben. Minden egyes alkalmazást csatornáin keresztül lehet igénybe venni. Ha az egyiknek van egy szabad kimeneti csatornája – például egy billentyűzetnek a kimenete –, akkor azt felajánlja egy, azt felhasználni tudó alkalmazásnak (akinek van ilyen bemenete); ha szabad bemeneti csatornája van – például egy képernyőnek a bemenete –, azt hozzáférhetővé teszi. Ezzel a szemlélettel nem csak a perifériákban, de alkalmazásokban is gondolkodunk: egy programra egyszerűen úgy tekintünk, mint be- és kimenetek egy struktúrált halmazára. Bemeneteire adatokat vár, és kimeneteire adatokat rak – pontos belső működése rejtve marad a többi alkalmazás elől. Ezeket az alkalmazásokat kívánjuk egymással összekötni a felhasználó – és esetleg a programok – igénye szerint.

Az összeköttetéseket pont-pont típusúnak definiáltuk, minden kapcsolatban csak két entitás egy-egy kivezetése érintett. Adatküldés szempontjából minden összeköttetés egyirányú – hiszen bemeneteket kötünk össze kimenetekkel (6. ábra) –, üzenetnyugtázási célokból azonban gya-

korlatilag kétirányú. Sok esetben előfordulhat, hogy egynél több összeköttetést kell egyszerre kiépíteni, hogy egy applikáció által felkínált szolgáltatást igénybe lehessen venni (a 3. fejezetben leírt Tetris példánál maradva a Tetrisnek egyszerre van szüksége egy billentyűzet-bemenetre és egy képernyő-kimenetre). Az ekkor egyszerre felépülő összeköttetéseket kapcsolatrendszernek nevezzük.

Egy kimenetet illetve bemenetet egyszerre több bemenettel illetve kimenettel is össze lehet kötni. Ezek közül azonban adatáramlás szempontjából mindig csak az egyik van használatban, mégpedig az amelyiken az úgynevezett *fókusz* van. Fókuszt átadni egyik csatornáról a másikra úgynevezett *fókuszváltás*sal lehet. Ezt az eljárást leginkább a különböző operációs rendszerekben használt „ALT-TAB” ablakváltásokhoz lehet hasonlítani: a háttérben mindegyik alkalmazás fut, de a felhasználó aktuálisan csak eggyel foglalkozik. Ismét az esettanulmányt vizsgálva, fókuszváltásra van szükség, amikor a mp3-lejátszó szolgáltatás helyett Zoltán a Tetris játékot szeretné a képernyőn látni.



6. ábra. Applikáció által látott világ

Rengeteg különféle program és hardvereszköz nagyon sokféle be- és kimenetet ajánlhat fel egymásnak. Egy egér-kimenetet természetesen csak egy egér-bemenettel van értelme összekötni; egy véletlenszám-generátor kimenetéhez pedig nem nagyon érdemes egy képernyő-bemenetet hozzárendelni. Ezekből a példákból is látszik, hogy bemenet- és kimenettípusokat kell definiálni és minden egyes összeköttetés létrehozása előtt típusellenőrzésre van szükség.

A hálózatban mindenképpen kell legalább egy vezérlő entitásnak lennie, ami a kapcsolatokat menedzseli. Bár egyes alkalmazások is betölthetnek lokálisan vezérlő szerepet – de csak olyan kapcsolatrendszerben, melyben ők is érintve vannak –, azonban csak a fő vezérlőközpontnak van joga tetszőleges alkalmazásokat egymáshoz rendelni. Egy tipikus irányító lehet egy PDA, mely elegendő számítási kapacitással rendelkezik egy egész PAN felügyeletéhez. A rugalmasság érdekében nem teszünk olyan megkötést, hogy csak egy vezérlő intelligencia lehet a hálózatban, megengedjük többnek a jelenlétét is.

A hálózatban sokfajta számítástechnikai eszköz működhet együtt: asztali számítógép, szerver, noteszgép, mobiltelefon, sőt akár speciális célhardver is, amelyeket csak egy fajta szoftver futtatására terveztek. Ez utóbbi típus képviselője lehet például valamilyen műszer, játékgép vagy szűkebb értelemben vett számítógép-periféria, mely rádiós interfésszel rendelkezik: egér, billentyűzet, monitor.

Célunk volt, hogy az eszközök kommunikációjához elegendő legyen az általunk tervezett *Blown-Up Micronet Protocol* ismerete és használata. Rendszerünk lehetőséget teremt arra, hogy az alkalmazások fejlesztői az általunk definiált programozói felület (Application Programming Interface – API) használatával egyszerűen tudjanak egy elosztott, ad hoc jellegű PAN-ra alkalmazásokat írni. Az API C nyelven írt könyvtárában található függvények használatával a programozók felkészíthetik alkalmazásaikat a BUMP támogatására.

4.2. Terminológia

Az egyes fogalmak félreértelmezésének elkerülése végett most rögzítjük a későbbiekben általánosan használt elnevezések pontos jelentését.

Közös néven gyűjtjük össze a perifériákat és az applikációkat, melyek a rendszerbe kerülvén meghirdetik magukat: ezek a *szolgáltatás*ok, melyeket a felhasználó illetve más entitások igénybe vehetnek. Mivel a ma elterjedt hardvereszközök csak megfelelően megírt programok segítségével illeszthetők bele a blown-up rendszerbe, ezért sokszor az *alkalmazás* kifejezést használjuk akkor is, ha perifériák csatlakoztatásáról van szó.

Az IC-k (Integrated Circuit - integrált áramkör) témakörében megszokott elnevezés analógiájára az egyes szolgáltatások be- és kimeneteit *láb*aknak (PIN) nevezzük. Egy láb adatáramlás szempontjából egyirányú (kimeneti vagy bemeneti). Egy alkalmazásnak lehetnek opcionálisan és kötelezően igénybeveendő lábai is, ezt a későbbiekben tárgyaljuk.

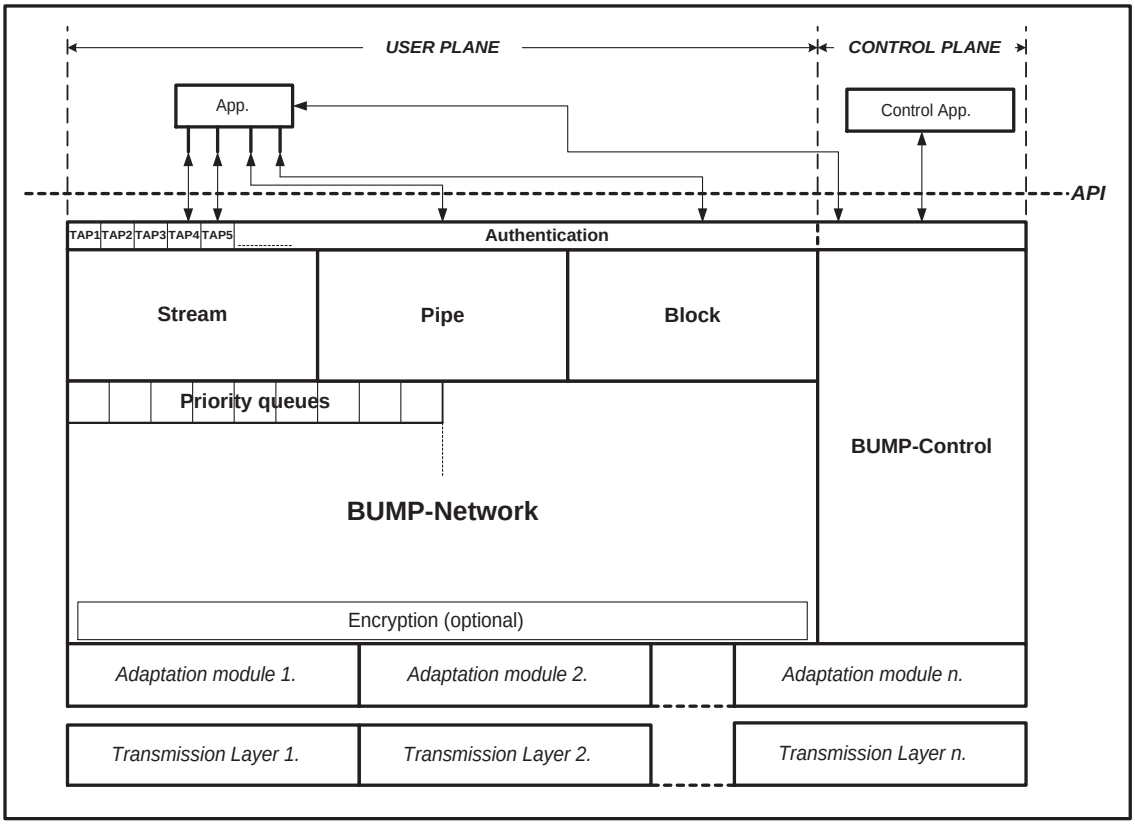
*Eszköz*öknek azokat az entitásokat hívjuk, amelyeken az egyes szolgáltatások futnak. Az eszközöket BUMP címeikkel különböztetjük meg. Az eszközökön futó szolgáltatások száma az adott eszköztől függ: egy egér eszköz nagy valószínűséggel csak egy darab egér szolgáltatást fog nyújtani, míg egy személyi számítógép jóval többet kínálhat fel.

*Csatorná*nak hívjuk a két láb közötti kapcsolatot.

4.3. A protokoll rétegeinek áttekintése

A *Blown-up* rendszer a *Blown-Up Micronet Protocol*-t használja, mely három rétegből épül fel: tartalmaz egy szállítási, egy BUMP-Network és egy adaptációs réteget. Ezek fölött helyezkednek el az alkalmazások, illetve ezek alatt találhatók az információ átvitelhez szükséges rétegek. Amint az a 7. ábrán látható, a protokollvermet további két részre, egy **felhasználói** és

egy **vezérlési síkra** (BUMP controller – BUMPC) osztottuk fel. Az alkalmazások az előbbit tényleges kommunikációra, utóbbit az összeköttetések menedzselésére használják.



7. ábra. BUMP rétegszerkezet

Az alkalmazások lábaikkal a modulárisan felépülő BUMP szállítási réteghez (felhasználói sík) csatlakoznak. Mindegyik láb egy adott típusú szállítási modult vesz igénybe. A lábak ezeken keresztül küldhetnek adatokat a többi applikációnak, illetve fogadhatnak információt tőlük. Amennyiben egy felhasználó a hálózatban található eszközöket szeretné egymással összekötni vagy a már létező kapcsolatokat módosítani, úgy a vezérlési síkon elhelyezkedő vezérlő modulnak küld parancsokat.

A **szállítási réteg** feladata az alkalmazástól érkező bitek helyes sorrendbe állítása és azoknak a túloldali vevő azonos rétegéhez való továbbítása, az adott szállítási rétegben specifikáltak szerint. Megkülönböztethetünk szállítási típusokat aszerint, hogy az adatátvitel folyam vagy tranzakciós jellegű, megbízható-e, használ-e folyamszabályozást, illetve hogy mekkora adategységeket kezel.

A **BUMP-Network** feladata a tényleges kapcsolatok kezelése: ez a réteg tartja nyilván, hogy melyik helyi alkalmazás melyik lába mely másik (távoli vagy helyi) alkalmazás melyik lábával van összekötve csatorna által. A BUMP-Network fogadja a felette levő rétegtől érkező

üzeneteket, és az alatta levő **adaptációs rétegen** át továbbítja őket azon eszköz BUMP-Network rétegének, amelyen az összeköttetés másik lába található, majd ezen eszköz BUMP-Network rétege átadja az üzenetet a szállítási rétegének.

A hálózatban a lábak közti összeköttetéseket a **BUMP controller** építi fel, bontja le és menedzseli. A lábak közötti kapcsolatok a rendszer filozófiájából adódóan pont-pont jellegűek, de egy speciális alkalmazás illesztésével pont-multipont kapcsolatokat is lehet szimulálni (például egy alkamazással, melynek egy billentyűzetbemenete és sok billentyűzetkimenete van). A BUMPC felel a hálózat eszközei által felkínált szolgáltatások felderítéséért is.

4.4. Felhasználói sík

4.4.1. Szállítási réteg

Minden egyes kommunikálni kívánó alkalmazás rendelkezik egy vagy több lábbal. A lábak a szállítási réteghez, pontosabban annak egy-egy konkrét moduljához csatlakoznak, egy szállítási réteg elérési ponton (Transport Access Point – *TAP*) keresztül. Az alkalmazások megadhatják, hogy mely lábuk melyik modulhoz csatlakozzon, attól függően, hogy milyen típusú csatornát szeretnének hozzá kapcsolni. Egyelőre három transzport modult képzeltünk el különböző tulajdonságokkal, de a rendszer későbbiekben kiegészíthető más, például QoS-t biztosító vagy real-time átvitelt támogató modulokkal is. Az általunk definiált folyam[1] (stream), csővezeték[2] (pipe) és blokk (block) típusú szállítási modulok tulajdonságai az 1. táblázatban láthatók.

	Folyam	Csővezeték	Blokk
adategység	kisebb, mint kb. 100 bájt	kisebb, mint kb. 100 bájt	1 blokk
jelleg	folyam	tranzakciós	tranzakciós
megbízhatóság	nem megbízható	megbízható	megbízható
folyamszabályozás	lassítás/gyorsítás	van	van
sávszélesség igény	kicsi	kicsi	nagy és löketszerű

1. táblázat. Folyam, csővezeték és blokk típusú szállítási modulok

A szállítási rétegben tartjuk nyilván az egyes applikációk lábait. Mindegyik lábhoz hozzá van rendelve

- a **tulajdonosa**, amely megadja, hogy melyik alkalmazás a láb tulajdonosa,
- egy **prioritás**, amely jelzi, hogy küldéskor a láb milyen elsőbbségekkel rendelkezik,

[1] leginkább egy patakhoz hasonlítható, ahol a víz elszivároghat

[2] ez egy zárt csőnek képzelhető, ahol biztosan annyi víz folyik ki, mint amennyi a csőbe bekerült

- **iránya**, mely megadja, hogy milyen irányban folyhat a lábon adat, azaz a láb írható vagy olvasható, illetve

- egy **kétértékű változó**, amely írható láb esetén jelzi, hogy az adatküldés engedélyezett, vagy le van tiltva.

Egy láb akkor van letiltott állapotban, ha az applikáció kimenetként használja és éppen nincs senkivel sem összekötve, vagy ha össze van kötve, de a csatorna túlsó végpontja több partnerrel is kapcsolatban áll és éppen nem az őket összekötő csatornán van a fókusz.

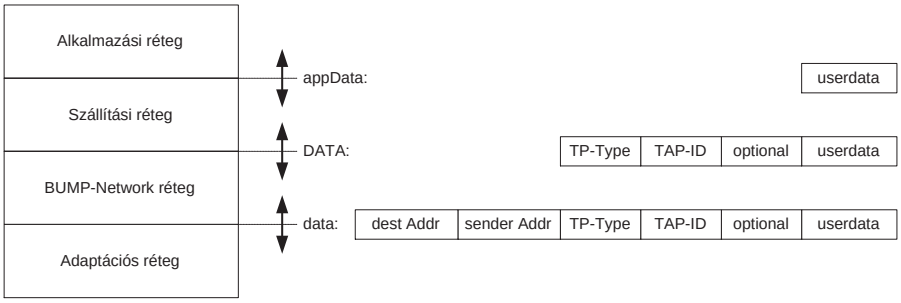
Ha egy alkalmazás az egyik lábára adatot küld *appData* üzenet formájában (8. ábra), azt a szállítási réteg modulja belerakja egy *DATA* üzenetbe, amely tartalmazza a láb (*TAP-ID*) és a modul azonosítóját (*TP-Type*), illetve egyéb modulspecifikus paramétereket (például megbízható átvitel esetén az átvitt keret sorszámát). Az alkalmazás ezt az üzenetet adja át a BUMP-Network rétegnek, ami ezután átkerül a vele összeköttetésben álló eszközhöz. A partner BUMP-Network rétege fogadja az üzenetet és feladja azt a megfelelő modulnak egy *DATA* üzenetben, ami a fejlécben található mező alapján értesül arról, hogy melyik *TAP*-jára érkezett az üzenet (8. ábra). A partner alkalmazás ezután a TAP-on keresztül egy *appData* üzenet formájában kapja meg a neki szánt információt.

Csővezeték típusú átvitel

A BUMP protokollhoz részletesen a csővezeték típusú szállítási modult terveztük meg. Ez egy csúszóablakos, azon belül egy szelektív ismétlést alkalmazó protokoll, amely jól alkalmazható zajos csatornák esetében. Csővezeték típusú átvitelt alkalmazva a fogadó fél egy keretet nem dob el csak azért, mert még nem kapta meg helyesen az előtte állót, hanem átmenetileg eltárolja azt.

Az elküldött üzenetek sorszámozva vannak, a helyesen vett üzenetek után a fogadó oldal mindig nyugtával válaszol (8. ábra, *DATAACK* üzenet). Az adó és a vevő fél is nyilvántart egy-egy ablakot, amelyben az előbbi tárolja, hogy mely sorszámú üzeneteket küldte el és nem kapott még rá nyugtát, az utóbbi, hogy mely üzeneteket várja, illetve mely üzeneteket raktározza átmenetileg (sorrend felcserélődés miatt, újraküldés elkerülése végett). A 9. ábrán a vevő ablakában fekete számokkal vannak jelezve a várt üzenetek, és piros számmal a vett, de még nem nyugtázott keretek sorszámai. Az ábrán látható adó ablakában – fekete számokkal – az elküldött, de még nem nyugtázott üzenetsorszámok találhatóak.

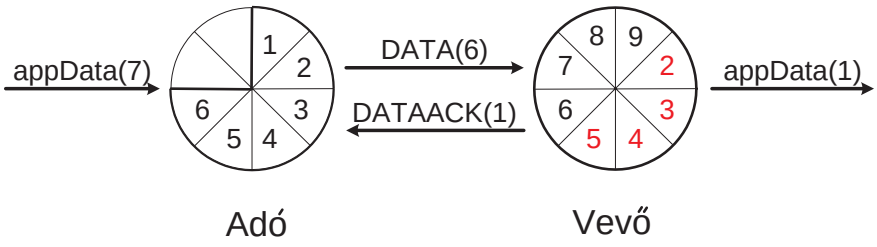
Az adó ablaka – mivel nem biztos, hogy mindig van adat – változó méretű, míg a vevőé fix nagyságú. Az ablakok cellái egy sorszámmezőből és egy pufferből állnak. Egy pufferben az adott sorszámhoz tartozó üzenetet tárolhajuk el.



8. ábra. Adatküldés

Az adó szállítási rétege beolvas egy üzenetet, ha van szabad puffere (azaz az aktuális ablakméret kisebb, mint a maximális méret), megnöveli az ablakméretet, elküldi az üzenetet és felhúz egy időzítőt. Amint kap egy nyugtát, a visszaigazolt sorszámhoz tartozó cella tartalmát törli, az ablakméretet eggyel csökkenti, valamint megnézi, hogy van-e további elküldésre váró adat. Amennyiben igen, úgy megismétli az előbb leírtakat. Ha az időzítő lejár, újraküldi az üzenetet. Mindezt addig ismétli, míg a próbálkozások száma elér egy megadott értéket. Ha az üzenettovábbítás ezek után is sikertelen, az adó hibát jelez az alsóbb rétegek felé.

A vevő fél egy üzenet vétele esetén megnézi annak sorszámát. Ha az ablaka tartalmazza azt, akkor az adónak nyugtát küld, ha ezt az üzenetet még nem vette, akkor el is raktározza azt. Ezután a vevő beletekint az ablakába. Ha ott a legkisebb sorszámhoz tartozó cella pufferében van adat, akkor azt kiszedi és továbbadja a felette álló rétegnek, majd ebbe a cellába bejegyzi az ablakban található számok legnagyobbikának eggyel növeltjét, jelezve ezzel, hogy melyik keretet várja. Következő lépésében ismét ellenőrzi, hogy a legkisebb sorszámú cella tartalmaz-e üzenetet, ha nem, akkor vár annak megérkezéséig.



9. ábra. Csővezeték típusú adatátvitel

4.4.2. BUMP-Network réteg

A BUMP-Network tartja nyilván az eszköz lábait összekötő csatornákat (TAP-TAP összerendeléseket), tartalmazza az adaptációs réteg felé irányuló, prioritásos adatküldéshez szükséges üzenetsorokat, valamint található benne egy titkosító modul, amellyel a hálózatba küldött üzenetek adat részét lehet titkosítani. A megfelelő kódoló és dekódoló kiválasztása további tanulmányozást igényel, ám nekünk nem volt célunk ennek vizsgálata, csupán a titkosítás használati lehetőségének biztosítása.

A BUMP-Network réteg a csomagtovábbítás irányának meghatározásához egy táblázatot használ, melyben eltárolja az adott csatorna szállítási típusát (*TP-Type*), szállítási réteg elérési pontjának azonosítóját (*TAP-ID*), prioritását (*Priority*), a csatorna túlodalán lévő eszköz címét (*remoteAddr*) és *TAP-ID*-ját, a csatorna kapcsolatrendszerének azonosítóját (*sessionID*), valamint egy kétértékű változót, mely csak a fókuszban lévő csatornáknál van bebillentve.

Amikor a szállítási rétegből átadunk egy továbbítandó üzenetet, a hálózati réteg kiolvassa az üzenetből, hogy melyik *TAP*-ról (*TAP-ID*) származik és milyen szállítási modulon (*TP-Type*) keresztül érkezett. Mindkét azonosítóra azért van szükség, mert például létezhet folyam és csővezeték típusú hozzáférési pont egyforma *TAP-ID*-vel. A BUMP-Network a nyilvántartott táblázat és a fenti paraméterek alapján tudni fogja, hogy az adatot melyik csatornán keresztül kell továbbítania. Csomagokat csak a fókuszban lévő csatornán lehet, és kell átküldeni. A szállítási réteg mindig csak ennek a csatornának az adataival foglalkozik, a többi addig felfüggesztett állapotban van. Miután a BUMP-Network meghatározza a csatorna másik végpontjának paramétereit, kicseréli az üzenetben a helyi *TAP-ID*-t a távoli eszköz szállítás elérési pontjának azonosítójára (*remoteTAP-ID*), majd kiegészíti a fejlécet az eszköz hálózati címével (*remoteAddr*). Ezzel előállt az elküldendő titkosítatlan üzenet (*data*, lásd a 8. ábrát).

A kész adatcsomagot a BUMP-Network az előbb kiolvasott prioritás érték alapján a megfelelő kimeneti sorba helyezi. Minden kimeneti sor különböző prioritással rendelkezik, ezáltal megoldható, hogy egy üzenet elküldését előnyben részesítsük egy másikhoz képest. Ez a BUMP használhatóságához elengedhetetlen követelmény, hiszen a felhasználó például nem örülne, ha lassúsága miatt az egere használhatatlanná válna egy nagyobb méretű fájl átküldése alatt. A BUMP-Network az üzeneteket a kimeneti sorokból egy prioritást kezelő algoritmus alapján veszi ki. Erre már sokféle algoritmus létezik, de egyelőre még nem tartottuk fontos szempontnak, hogy közülük kiválasszuk a számunkra leginkább megfelelőt.

A vevő oldalon a BUMP-Network réteg feladata, hogy az üzenetben található információt átadja a szállítási réteg megfelelő moduljának. Ehhez csupán bele kell néznie az üzenet fejlécébe és onnan ki kell olvasnia a szállítási típust (*TP-Type*). Ez a vevő oldal modulját egyértelműen meghatározza, így a réteg egyszerűen eldöntheti, hogy melyik szállítási modul a címzett.

A hálózati réteg tehát az adó és vevő oldali *TP-Type* és *TAP* párok összerendelésével csa-

tornákat alakít ki. A *TAP*-ok be-, illetve kimenet jellegűek lehetnek, ezért minden csatorna adatáramlás szempontjából egyirányú. Ám mivel adatküldés vezérlésére alkalmas üzenetek átvitelére is szükség van, ezért minden csatorna vezérlési szempontból kétirányú.

4.4.3. Adaptációs réteg

A *Blown-Up Micronet Protocol* legalsó rétege, az adaptációs réteg alakítja át a BUMP-Network réteg üzeneteit az átviteli rétegeknek megfelelő alakra. Mivel a BUMP-Network hálózati réteg helyett is használható, az átviteli rétegek alatt adatkapcsolati réteget és fizikai közeget is értünk egyszerre (ez lehet például TCP/IP, WLAN vagy Ethernet). Egyelőre az IP olyannyira elterjedt, hogy érdemes azt használni, de rendszerünk számára rengeteg olyan funkciót tartalmaz, ami indokolatlanul bonyolulttá teheti az esetenként csak primitív szolgáltatást biztosító eszközöket (például egér).

Az adaptációs réteg kicserélésével protokollunk tetszőleges adatkapcsolati rétegre illetve fizikai közegre ráültethető. Alkalmazható lesz például WaveLan (IEEE 802.11) hálózatokban valamilyen ad hoc routing protokolkiegészítéssel, felette TCP/IP-vel, vagy akár közvetlenül Bluetooth technológiára épülve, kihasználva annak piconet képességeit.

4.5. Vezérlési sík

4.5.1. A BUMP controller

A BUMP controller látja el a vezérlési sík feladatait, melyek a következők:

- a saját eszközön található applikációk által felajánlott szolgáltatások reklámozása,
- más eszközök által felkínált szolgáltatások figyelése és összegyűjtése, ezeknek kérésre a helyi alkalmazások számára történő elküldése,
- más eszköz utasítására a saját applikációk lábainak távoli vagy helyi lábakkal történő összekötése,
- saját eszközén futó vezérlő vagy vezérlő tulajdonsággal bíró alkalmazás számára kapcsolatrendszer felépítése, menedzselése és a használat befejeztével annak bontása.
- a BUMP-Network-ön keresztül küldött adatcsomagok figyelése, és ha egy saját alkalmazás már régóta nem küldött egy kimeneti lábán adatot, akkor a csatorna túloldalán levő eszköznek egy speciális üzenet küldése annak jelzésére, hogy a kimeneti lábbal rendelkező alkalmazás még fut.

Az eszközön futó bármely applikáció küldhet a BUMPC-nek utasításokat. Ezek funkciójuk alapján foglalkozhatnak alkalmazások regisztrációival, valamint összeköttetések menedzselésével. A vezérlő alkalmazások képesek az általuk kezdeményezett összeköttetések menedzselésre, míg felhasználói programok csak akkor (és csak korlátozott jogokkal) módosíthatják a kapcsolatokat, ha abban saját lábuk is részt vesz. Tehát míg egy speciális vezérlő program képes arra, hogy távoli eszközöket kössön össze a megvalósuló összeköttetésrendszerben való részvétel nélkül (leszámítva azt az esetet, amikor a vezérlő program felépített kapcsolatot bont), addig erre egy egyszerű felhasználói alkalmazásnak nincs jogosultsága. Ennek ellenőrzését a vezérlő modul felett elhelyezkedő autentikációs egység végzi: nyilvántart minden BUMP-ot irányítani kívánó applikációt, azok jogait, és ezen bejegyzések alapján vizsgálja a programok kéréseit.

A BUMPC szolgáltatás-nyilvántartással és összeköttetés-vezérléssel kapcsolatos táblázatokat kezel. Ezeket az új applikációk regisztrálásának és törlésének ismertetése, illetve egy kapcsolatrendszer felépítési, fenntartási és bontási folyamatának bemutatása során fogjuk részletesen tárgyalni.

4.5.2. Applikáció regisztrálása

A *Blown-up* rendszer alkalmazásainak szüksége van legalább egy bemenetre vagy kimenetre. Ahhoz, hogy ezeken át adatot lehessen küldeni vagy fogadni, az alkalmazást a rendszerbe regisztrálni kell. Az ebben a fejezetben ismertetésre kerülő, vezérlési síkon át küldött üzenetek segítségével meg kell adni a rendszernek, hogy az applikáció milyen ki- illetve bemeneti célokat ellátó lábakkal rendelkezik.

A BUMPC-nek a következőket kell tudnia egy nála bejegyzett alkalmazásról:

- az **alkalmazás neve** (*AppName*), ami az eszköz címével együtt az alkalmazást a többi entitás számára egyértelműen azonosítani fogja,
- az alkalmazás **leírása** (*AppDescr*), amiből kiegészítő információkat tudhatunk meg a programról (például az alkalmazás leírását, funkcióját),
- a szolgáltatás belső, rendszeren belüli **azonosítója** (*AppID*),
- az egyes **lábak tulajdonságai**, melyek a következők:
  - a **láb típusa** (*PIN-Type*), mely jelzi, hogy milyen fajta információt kezel (egér esetében például *mouse*). A típus megadása a rendszer számára ellenőrzési lehetőséget biztosít a lábak összekapcsolásához, segítségével kiszűrhető, hogy egy hibásan

megírt vezérlő két eltérő funkciójú lábat (pl. egy egér kimenetet egy képernyő bemenettel) kössön össze[3],

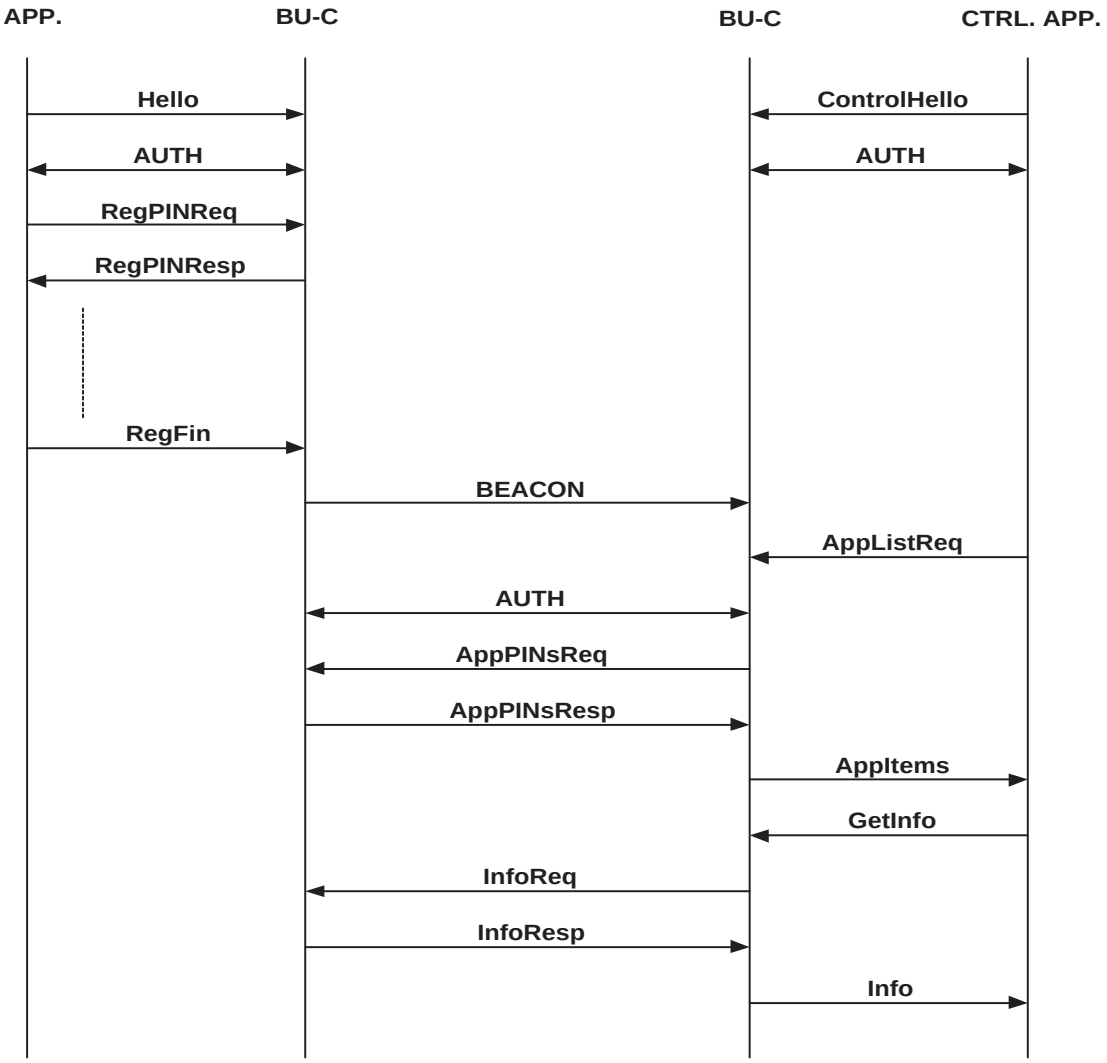
- az **adatáramlás iránya** (*I*nput/*O*utput), mely megadja, hogy az alkalmazás írni vagy olvasni fogja a lábat. Egy láb egyszerre csak egyik fajta lehet, duplex átvitelhez az alkalmazásnak két lábat kell használnia,
- a láb **szükségessége** (*O*ptional/*M*andatory), annak jelzésére, hogy a szolgáltatás igénybevételéhez feltétlenül csatlakoztatni kell-e erre a lábra egy másik szolgáltatást, vagy használata opcionális. A 3.1. fejezetben ismertett esettanulmányban például nem igazán lenne értelme elindítani egy saját kijelző nélküli Tetris játékot, ha képernyő kimenetére nem csatlakoztatunk semmit,
- az alkalmazás által írható lábak esetében a láb **fontossága** (*Priority*), melynek az adat küldésénél van szerepe. A BUMP-Network réteg ismertetésénél említett módon bizonyos lábakon keresztül küldött adatoknak elsőbbsége van másokkal szemben,
- a láb **kapacitása** (*Capacity*), annak nyilvántartására, hogy az adott lábhoz egyszerre hányan kapcsolódhatnak. A csatornák közül fókuszváltással lehet az adatáramlás szemszögéből éppen élő kapcsolatot kiválasztani,
- a láb által **használt szállítási rétegbeli modul típusa**, annak eldöntésére, hogy adatküldéskor az üzenetet a BUMP-Network melyik modulnak adja tovább,
- a láb – rendszerben használt – **belső azonosítója** (*TAP-ID*), mely megadja, hogy a láb az előbb kijelölt moduljának mely szállítási elérési pontjához (*TAP*) csatlakozott,
- a láb **felhasználó által adott azonosítója** (*MyPINName*), hogy egy új összeköttetés felépítésekor a vezérlő több azonos típusú láb esetén egyértelműen kiválaszthassa az egyiket. Ismét a 3.1. fejezetben ismertett esettanulmányt vizsgálva, a két billentyűzettel rendelkező Tetris játék klaviatúrákkal való összekötésekor egyértelműen azonosítani kell a két bemenetet, nehogy ugyanahhoz próbáljuk meg csatolni mind a kettőt. Ennek a névnek a használata felhasználói szemszögből sokkal egyszerűbb, mint egy számazonosító kezelése.

A BUMPC a fenti paraméterekből álló bejegyzéseket egy szolgáltatás-nyilvántartó táblázatba helyezi.

Az alkalmazás-regisztráció folyamatát (10. ábra) leginkább az adatbáziskezelőknél használt tranzakciókhoz lehet hasonlítani. Első lépésben az applikáció elküld a BUMPC-nek egy *Hello* üzenetet, mely a szolgáltatás legfontosabb tulajdonságait tartalmazza: a nevét, a leírását és

[3] erről bővebben a 4.5.3. fejezetben lesz szó

az azonosítóját. Ezután az alkalmazás sorban elküldi lábregisztrációs üzeneteit (*RegPINReq*), melyben egy-egy lábának bejegyzését kéri. Itt adja meg, hogy lábai milyen tulajdonságokkkal rendelkeznek. Ezek mindegyikére egy *RegPINResp* válaszüzenetet kap, amelyben a BUMPC elküldi a lábhoz rendelt szállítási réteg elérési pont-azonosítót (*TAP-ID*). Miután az applikáció az összes lábára kiadott egy bejegyzési kérelmet, egy *RegFIN* üzenettel zárja le a regisztrációját. A BUMPC az új szolgáltatással és annak paramétereivel kiegészíti az általa nyújtott szolgáltatásokat nyilvántartó táblázatot, mely alapján a szolgáltatás periodikusan a hálózatra küldött *beacon* üzenetekkel lesz meghirdetve. Ezek az alkalmazás eszközének címét, a szolgáltatás nevét és azonosítóját tartalmazzák.



10. ábra. Applikáció regisztrálása, reklámozása és lekérdezése

A többi eszköz BUMPC-je folyamatosan figyeli a hálózatot, hogy ki milyen alkalmazást ajánl fel. Ezen ismereteket lekérdezés esetén továbbadja egy vezérlő alkalmazásnak, mely-

nek segítségével a felhasználó – az előbbi információk birtokában – lábakat tud összekötni. Amennyiben egy eszköz BUMPC-je egy új szolgáltatás *beacon* üzenetét veszi, a reklámozó félnek küld egy *appPINsReq* üzenetet, melyben megkérdezi, hogy az applikáció milyen lábakkal rendelkezik. Válaszul egy *appsPINsResp* üzenetet kap, melynek tartalmával frissíti az igénybevehető szolgáltatásokat nyilvántartó táblázatát. Azáltal, hogy a BUMPC egy új szolgáltatásról két lépcsőben értesül (új *beacon*, majd *appsPINsResp*) jelentős sávszélességet takarítunk meg, ugyanis a sűrűn kiküldött reklámozó üzenetek csak a legszükségesebb információt fogják magukban hordozni.

A BUMPC ellenőrzi, hogy minden számára felkínált alkalmazástól folyamatosan kap-e reklámozó üzenetet. Amennyiben nem (lejár a *beacon*-höz tartozó óra, amit minden üzenet fogadásakor felhúz), a szolgáltatást törli az igénybevehető alkalmazások táblázatából. A *beacon* jelek abbamaradása két okból lehetséges: ha az alkalmazás visszavonta felajánlását, vagy ha az eszközök túlságosan eltávolodtak egymástól és már nem veszik egymás rádiós jeleit. Egy felajánlás visszavonása jelentheti azt, hogy az applikáció befejezte működését, és azt is, hogy lábaihoz több csatorna már nem rendelhető, mivel összeköttetéseinek száma elérte a maximálisan megengedettet.

4.5.3. Kapcsolatrendszer kiépítése

Egy kapcsolatrendszert egy eszközön futó speciális vezérlő vagy egy ilyen képeségekkel rendelkező alkalmazás tud kiépíteni. A 3.1. fejezetben bemutatott esettanulmányban ilyen szerepe van a PDA-nak, aki felépít egy kapcsolatrendszert képernyője, egy Tetris játék és egy billentyűzet között. A vezérlő egységnek mindenekelőtt az autentikációs egységen keresztül egy *ControlHello* üzenettel és az alkalmazott azonosítási eljárás speciális üzeneteivel be kell jelentkeznie a BUMPC-hez.

A vezérlő a BUMPC-től *AppListReq* üzenetekkel tudja lekérdezni a hálózatban reklámozott szolgáltatásokat (10. ábra). Egy ilyen üzenetre válaszul *AppItems* üzenetet kap, mely egyedül a *beacon*-ökben küldött információkat tartalmazza. Ha a vezérlő további információt szeretne megtudni az adott szolgáltatásról, akkor a BUMPC rétegnek *GetInfo* üzenetet küld. A BUMPC ennek hatására a túloldal felé egy *infoReq* üzenetet küld, melyre a megcímzett BUMPC egy *infoResp* üzenetben válaszol, amely már a kért *AppDescr* leírót tartalmazza. Ezután a vezérlő BUMPC-je a kívánt információt az applikációnak *Info* üzenet formájában juttatja el. A szolgáltatás igénybevételéhez szükséges követelményeket (lábak száma és azok tulajdonságai) a BUMPC rétegnek minden őt érdeklő applikáció esetében külön *PINListReq* üzenetekkel, egyesével kell lekérdeznie. A vezérlő az ezekre feleletként kapott *PINItems* üzenetek alapján tudja magában összeállítani, hogy melyik lábat mely másikkal fogja majd összekötni.

A kapcsolatrendszer kiépítése – egy applikáció lábainak regisztrációjához hasonlóan – tran-

zakciós jelleggel történik. A vezérlő az üzenetek elküldése előtt kisorsol egy ún. tranzakciós azonosítót (*transactionID*), mellyel később egyértelműen tud hivatkozni a kiépített vagy kiépítés alatt álló összeköttetés-halmazokra. Ezután küldi csak el az első üzenetet, a *TStart*-ot, a tranzakció kezdetének jelzésére. Ez tartalmazza az előbb említett azonosítót, illetve a vezérlő saját belső azonosítóját (*controlAppID*). Az átküldött azonosítópár a BUMPC-ben egy összeköttetés-azonosítót (*sessionID*) határoz meg. Ez a későbbiekben az eszköz címével kiegészítve a hálózatban egyedi azonosítóvá válik, és egyértelműen jelölni fogja az egy kapcsolatrendszerhez tartozó csatornák csoportját.

A *TStart* üzenetet követően a vezérlő alkalmazás sorra elküldi a pont-pont kapcsolatok felépítésére vonatkozó kéréseit egy-egy *ConnPINs* üzenet formájában. Az összes összekötettést kérő parancs kiadása után a vezérlő egy *TEnd* üzenettel jelzi, hogy a tranzakciót befejezte, nincs több csatornára igénye. Amennyiben erre egy *TNOK* üzenetet kap válaszul, úgy magában elkönyveli, hogy a BUMPC nem tudta létrehozni számára a kért kapcsolatrendszert. Ha a BUMPC *TOK*-val jelez vissza, az a kiépítés sikerességét nyugtázza és emellett azt is tartalmazza, hogy mely opcionális láb – opcionális láb összeköttetéskérések teljesültek. Erre azért van szükség, mert bár előfordulhat, hogy a BUMPC nem tudta az összes kért lábat összekötni, az összes kötelező láb sikeres összekapcsolása esetén a kapcsolatrendszer szerinte mégis felépíthető. A BUMPC ebben az esetben meghagyja a felhasználónak a választás jogát: vagy használja a kapcsolatrendszert ilyen formában, vagy lebontja, és megpróbál helyette egy újat kiépíteni.

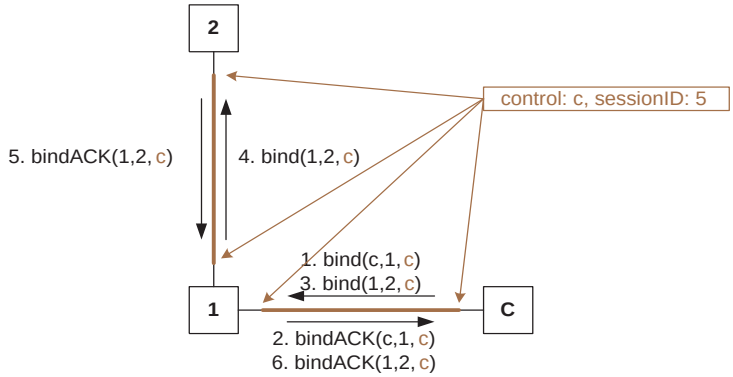
A BUMPC egy kapcsolatrendszert egy előbb említett *TStart* üzenet hatására kezd el felépíteni. Első lépésben minden pont-pont összeköttetés-kérelemhez leellenőrzi, hogy értelmes-e: azonos *PIN-Type*-pal rendelkező be- és kimenetet akar-e egymással összekötni. Ha igen, akkor ezután sorban minden összeköttetéshez kiküld pontosan egy *bind* üzenetet, amely törzsében tartalmazza a kapcsolatrendszer kiépítését kezdeményező eszköz címét. Ellenkező esetben a kapcsolatrendszert nem építi fel, hibát jelez a vezérlőnek.

Értelmezés és feldolgozás szempontjából két fajta *bind* üzenetet különböztethetünk meg. Az első típusúban az üzenet küldője megegyezik a kért összeköttetés egyik eszközével, a másodikban a küldő nem lesz része az összeköttetésnek (vezérlő alkalmazás). Az üzenet címzettje a kért csatorna egyik végpontja:

- ha a vezérlő és egy másik fél között kell kiépíteni a csatornát, akkor értelemszerűen a másik eszköz,
- míg ha a vezérlő nem része a csatornának, akkor a két csatornavégpont közül az egyik.

Amennyiben a *bind* küldője lesz a csatorna egyik végpontja, és a vevőnél a kívánt láb létezik, valamint van szabad kapacitása, úgy a címzett *bindACK*-val válaszol, míg elutasítás esetén

bindNACK-ot küld vissza. Pozitív nyugta küldésekor a címzett egyúttal frissíti az összeköttetéskéréseket nyilvántartó táblázatát, mely szerepét és használatát a következő bekezdésben ismertetjük. Ha azonban a *bind* küldője nem végpontja a kialakítandó csatornának, úgy a címzettnek a *bind*-ot tovább kell küldenie, hogy a harmadik fél is értesüljön kérelemről. Ez az eszköz már a fent leírtak szerint fog cselekedni, hiszen a *bind* üzenetet a kért csatorna másik végpontjától kapta. Ha a küldő a harmadik féltől egy *bindACK* üzenetet kap, akkor frissíti táblázatát és pozitív visszajelzést küld, míg *bindNACK* hatására negatív nyugtával jelez vissza a vezérlőnek. (11. ábra)



11. ábra. Kapcsolatrendszer kiépítése *c* kezdeményezésére és felügyelete alatt

A BUMPC a hozzá érkező összeköttéskéréseket egy táblázatban tartja nyilván, mindegyikhez egy státuszt rendelve. Ez a státusz lehet „függő” vagy „véglegesítendő”. Amikor a BUMPC egy új, eddig ismeretlen *bind* üzenetet vesz és a benne megjelölt láb szabad, akkor a kérés „függő” állapottal bejegyzésre kerül. Ez a státusz mindaddig változatlan marad, amíg a lábhoz tartozó alkalmazás összes kötelezően igénybeveendő lábára nem érkezik egy olyan kérés, mely ugyanazon BUMPCtől származik és ugyanazon összeköttetés azonosítóval rendelkezik (*sessionID*). Ha ez teljesül, akkor ezen összeköttéskérések mind „véglegesítendő” állapotba kerülnek. Az ezután érkező, ugyanezen vezérlő által kezdeményezett, és ugyanezen összeköttetés azonosítóval rendelkező összeköttetéskérelmek (amik már csak opcionális lábra vonatkozhatnak) már „véglegesítendő” állapotban kerülnek bejegyzésre.

A vezérlő oldalán levő BUMPC az összes kiküldött *bind* üzenetére választ vár. Amennyiben egy nyugta adott időn belül nem érkezik meg, a BUMPC többször újraküldi az eredeti üzenetet. Ha egy bizonyos számú próbálkozás után sem érkezik rá válasz, és ezen kérések nem opcionális lábakat szerettek volna egymással összekötni, akkor a BUMPC felhagy a próbálkozással, és a sikertelenséget jelzi az összeköttetést kezdeményező alkalmazás felé.

Amint a BUMPC-hez megérkezett az utolsó olyan összeköttetéskérésre adott válasz, amelyben legalább az egyik láb kötelezően igénybeveendő, a BUMPC egy üzenetszórással kiküldött

run üzenettel belekezd a kapcsolatrendszer véglegesítésébe. A két lépcsős kapcsolatfelépítés előnye, hogy segíti elkerülni az erőforrások felesleges lefoglalását. A rendszer egy alkalmazásnak csak akkor jelzi vissza, hogy igénybevették a lábát, mikor az általa és párja által alkotott csatorna kapcsolatrendszere teljesen kiépült.

A vezérlő egység által kiküldött *run* üzeneteket minden BUMPC veszi, és megnézi, hogy az előbbiekben tárgyalt táblázatában van-e olyan bejegyzés, amely az üzenetet küldő vezérlőhöz tartozik, az összeköttetés azonosítója (*sessionID*) megegyezik a *run* üzenet törzsében levővel, és a bejegyzés véglegesítendő állapotú. Ha van, akkor a bejegyzéseket átadja a BUMP-Network által nyilvántartott összeköttetéstáblázatnak és ha a lábra ez volt az első csatlakozás, akkor a fókuszt is ehhez az összeköttetéshez rendeli. Ezzel egyidőben minden ilyen bejegyzéshez tartozó alkalmazás után – annak azonosítójával – visszaküld a vezérlőnek egy-egy *runACK* üzenetet.

A vezérlő BUMPC-je annyi *runACK* üzenetet vár, ahány applikáció-lábat össze szeretne kötni. Ha ezek mind megérkeztek, akkor egy, már korábban említett *TOK* üzenettel visszajelez a vezérlő applikációnak, hogy a kívánt kapcsolatai élnek. Amennyiben egy előre megadott időn belül nem érkezik meg az összes *runACK*, a vezérlő BUMPC modulja párszor újra elküldi a *run* üzenetet. Ha ezekre sem érkezik meg a kellő számú nyugta, akkor a vezérlő kezdeményezi a már részben felépített kapcsolatrendszer bontását.

4.5.4. Kapcsolatrendszer lebontása

Egy kapcsolatrendszer lebontását kezdeményezheti:

- egy alkalmazás, melynek lába szerepel egy adott összeköttetésrendszerben,
- egy szállítási rétegbeli modul,
- az a vezérlő, amelyik a kapcsolatrendszert létrehozta.

A parancs kiadására azonban csak a kapcsolatokat felépítő BUMPC-nek van joga, tehát a fenti három entitásnak a kapcsolatlebontási kérelmeket előbb hozzá kell eljuttatniuk.

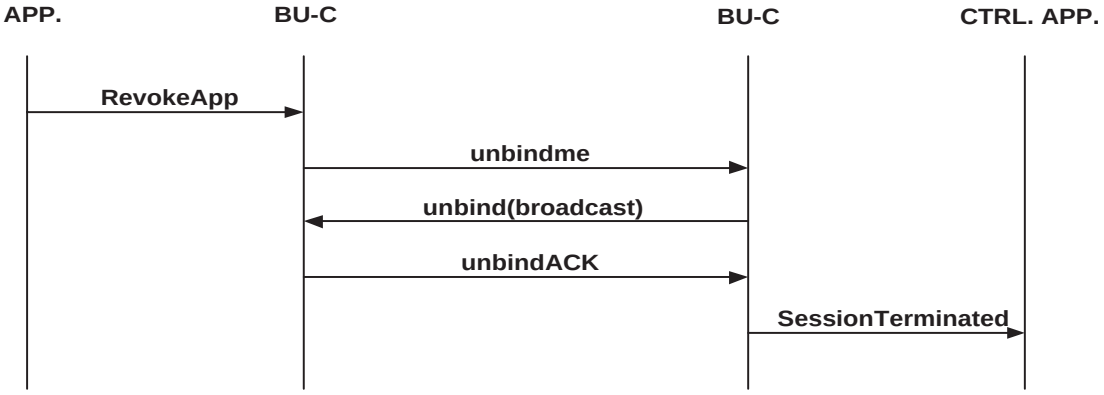
Ha egy applikáció vissza szeretné vonni a szolgáltatását, egy *RevokeApp* üzenetet küld a saját BUMPC-jének, melyben átadja belső azonosítóját (*AppID*). A BUMPC ennek hatására egy *unbindme* üzenetetet küld azon vezérlő eszköz BUMPC-jének, aki a „visszavonuló” applikáció lábát vagy lábait valamely más lábbal vagy lábakkal összeköttette.

A vezérlő eszköz BUMPC-je az előbb elküldött üzenet vétele után egy üzenetszórással továbbított *unbind* üzenetettel válaszol. A BUMPC szempontjából nincs ugyanis értelme fenntartani az összeköttetésrendszert, ha annak az egyik láncszeme meghibásodott.

A hálózatban jelenlevő összes BUMPC megkapja az előbb szétküldött üzenetet és megnézi, van-e olyan bejegyzett lába, melyhez az üzenet törzsében szereplő összeköttetés-azonosítóval jelzett csatorna vagy csatornakérelem tartozik. Amennyiben igen, úgy törli a BUMP-Network és BUMPC által kezelt táblázatokból az ezen lábhoz tartozó bejegyzéseket, majd visszaküld egy *unbindACK* üzenetet a vezérlő BUMPC-nek.

Az *unbind*-ot küldő BUMPC annyi nyugtát vár üzenetére, ahány eszköz az adott kapcsolatrendszerben részt vett. Ha adott időn belül nem kapja meg az összes visszajelzést, akkor párszor újra megpróbálkozik a kapcsolatrendszer teljes bontásával. Ha ezek után sem kap választ, az egész kapcsolatrendszert lebontottnak tekinti és küld a kapcsolatrendszert korábban felépítő vezérlőalkalmazásnak egy *SessionTerminated* üzenetet. Szintén elküldi ezen üzenetet sikeres kapcsolatrendszer-bontás esetén, azaz mikor mindenki nyugtázta *unbind* üzenetét. Azok, akik nem küldtek nyugtát, automatikusan bontják a kapcsolatot, észlelvén, hogy a kapcsolatrendszer többi tagja már nem elérhető.

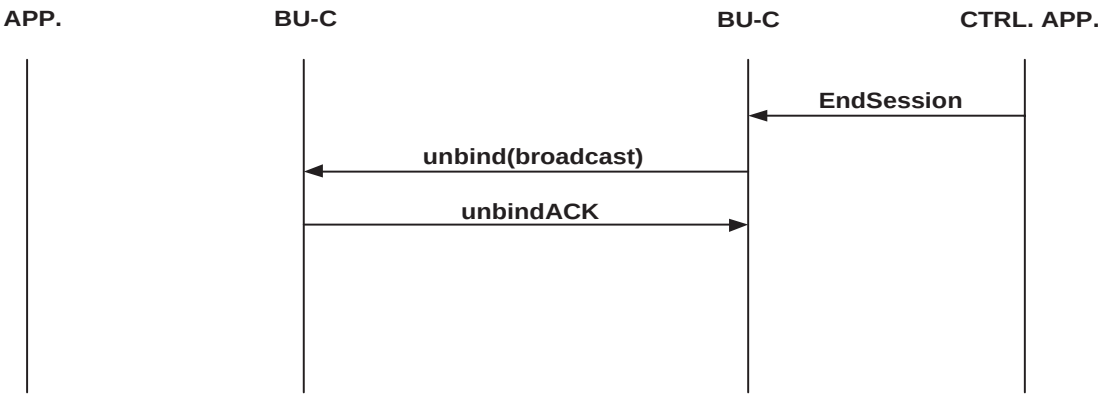
A legutolsó lépésben az *unbindme* üzenetet küldő eszköz BUMPC egysége – miután az összes fenti kommunikáció lezajlott – töröl minden a megszüntetendő applikációval kapcsolatos bejegyzést. Ezzel a kapcsolatok leépítése megtörtént (12. ábra).



12. ábra. Szolgáltatás visszavonása

Az összeköttetéseket az azt létrehozó vezérlőalkalmazásnak is joga van lebontani (13. ábra). Ehhez egy *SessionEnd* üzenetet kell küldenie a BUMPC-nek megadván benne a saját belső azonosítóját (*controlAppID*) és az általa az összeköttetés kérésekor kisorsolt tranzakciós számot (*transactionID*). Ezután a kapcsolat lebontása úgyanúgy történik, mint amikor a BUMPC egy *unbindme* üzenetet kapott.

Szállítási rétegbeli modul akkor bonthat egy kapcsolatot, ha azt hosszabb ideje nem tudja már használni, például adatcsomagjaira sokszori próbálkozás után sem kap nyugtát a vevőtől. Ilyenkor egy belső üzenettel kéri a BUMPC-től, hogy bontsa a kapcsolatot, ami a hálózatra kül-



13. ábra. Kapcsolatrendszer lebontása

dött *unbindme* üzenettel jelzi ezt a vezérlő BUMPC-jének. Innentől kezdve a kapcsolatrendszer az előbbiekben ismertett módon bontódik le.

4.5.5. Alive üzenetek

A BUMP controller feladata, hogy az eszközök között fennálló kapcsolatokat ellenőrizze: folyamatosan vizsgálja, hogy használhatók-e a csatornák, illetve hogy a kapcsolat fennállása alatt nem tűnt-e el az adatot küldő alkalmazás. Megeshet ugyanis, hogy egy küldő fél a mobil eszközök mozgása következtében kívül kerül a vevő hatósugarán, és az hiába vár adatot. Meg kell tehát tudnunk különböztetni két esetet: azt, amikor az adó azért nem küld adatot, mert nem akar, és azt, amikor küld, de a vevő már nem tudja venni. Utóbbi esetben a kapcsolatot és ezzel együtt a kapcsolatrendszert is bontani kell.

A két állapot megkülönböztetéséhez speciális *alive* üzeneteket használunk, amit a kimeneti lábbal rendelkező alkalmazás BUMPC-je akkor küld ki, amikor az alkalmazás egy másik szolgáltatásnak már hosszabb ideje nem küldött adatot. A BUMPC minden szolgáltatáshoz annyi órát tart nyilván, ahány applikáció felé rendelkezik olyan csatornával, amelyhez lokálisan kimeneti láb csatlakozik. Ha a szolgáltatás egy irányba adatot küld, akkor mindig felhúzza az irányhoz tartozó órát. Amennyiben ez lejár, a BUMPC a csatorna másik végén levő eszköznek *alive* üzenetet küld, melyben jelzi, hogy csak azért nem ment a két applikáció közötti csatornákon adat, mert nem volt mit küldeni.

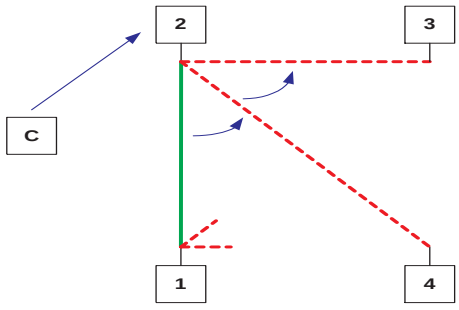
A BUMPC minden olyan applikációhoz is órát tart nyilván, amelytől adatot fogad. Ha a BUMPC érzékeli, hogy egy applikáció felől adat, vagy *alive* üzenet érkezik, akkor felhúzza az adott alkalmazáshoz tartozó órát. Amennyiben ez lejár, úgy kezdeményezi az érintett applikációhoz tartozó csatornák és ezzel együtt az egész kapcsolatrendszer bontását. A BUMPC abból indul ki ugyanis, hogy a küldő fél eltűnt, és ezáltal a köztük levő csatornák megszakadtak.

4.5.6. Fókuszváltás

Egy applikáció egy lábára egyszerre több csatorna is be lehet kötve, azonban ezek közül mindig csak az egyiken továbbítunk adatot, azaz mindig csak az egyiken van a fókusz. Fókuszváltásnak azt az eseményt nevezzük, amikor egy vezérlőalkalmazás egy láb csatornái közül a fókuszt egy másik csatornának adja át. Egy láb csatornái sorba vannak rendezve, egy fókuszváltással mindig a soron következő csatornára tudunk átváltani.

Fókuszváltást a kapcsolatrendszert felépítő, illetve egy lábait felajánló alkalmazás eszközén futó vezérlő kezdeményezhet. Előbbinek joga van az általa összekötött összes lábhoz tartozó fókusz megváltoztatására, míg az utóbbi csak a saját eszközén található alkalmazások lábait irányíthatja.

Egy vezérlő alkalmazás egy saját eszközén történő fókuszváltoztatást a BUMPC-nek küldött *ChangeLocalFocus* üzenettel tud elindítani. Az üzenet törzse tartalmazza, hogy a váltás mely applikáció (*AppID*) melyik lábára vonatkozik (*MyPINName*). A BUMPC először megnézi, hogy több csatorna is csatlakozik-e az adott lábhoz. Amennyiben igen, a fókuszt tovább kell adni az adott lábra csatlakozó következő csatornának. Ha a távoli láb kimeneti típusú, akkor a fókuszt váltó eszköz először elküld egy *lockyourPIN* üzenetet a fókuszban levő csatorna túlsó végére, amellyel a partner alkalmazásnak jelzi, hogy ne küldjön több adatot a csatornán, tiltsa le a hozzá tartozó lábat. Ezután megkeresi a BUMP-Network által kezelt táblázatban a jelenleg használt csatorna bejegyzését, és átadja a fókuszt a soron következő csatornának. Amennyiben az újonnan fókuszt kapó csatorna túlsó végén kimeneti típusú láb található, a nyilvántartó eszköz BUMPC-jének küld egy *unlockyourPIN* üzenetet felszólítván őt, hogy a lábra küldött adatokat ezentúl a csatornán továbbítsa. (lásd a 14. ábrán: a 2-es számú alkalmazás átvált az 1-essel közös csatornáról a 4-essel közösre, majd arról a 3-assal közösre)



14. ábra. Fókuszváltások

A vezérlő alkalmazások más eszköz lábán is tudnak fókuszt váltani, de csak az általuk csatlakoztatott csatornákra vonatkozóan. Ha a BUMPC egy vezérlőtől *ChangeRemoteFocus* üzenetet kap, akkor az üzenet törzsében lévő címre továbbküld egy *chfocus* üzenetet. Amennyiben a

törzsben megjelölt lábon a fókusz egy általa felépített csatornára mutat, akkor a távoli BUMPC az előbb ismeretett módon fókuszt vált, majd visszaküld egy *changefocusACK* üzenetet. Ha a nyugta adott időn belül, valamint többszöri újrapróbálkozás után sem érkezik meg, akkor a BUMPC a csatornát hibásnak nyilvánítja és lebontja. (lásd a 14. ábrát: a vezérlő a 2-es számú alkalmazás fókuszát átváltja az 1-essel közös csatornáról a 4-essel közösre)

4.6. Programozói felület

Protokollunkhoz terveztünk egy programozói interfészt (Application Programing Interface – API), mely megkönnyíti az alkalmazások fejlesztőinek a BUMP használatát. Ebben függvényeket definiáltunk, melyekkel a rendszer szolgáltatásai könnyebben elérhetők és ezeket egy könyvtárba gyűjtöttük össze.

4.6.1. Felhasználói függvények

Felhasználói függvények közé soroljuk mindazon API-hívásokat, amelyeket egy szolgáltatást felajánló alkalmazás használhat. Ezek a következők lehetnek:

- `bump_RegisterApplication`: egy alkalmazás bejelentkezését szolgálja. Paraméterül az applikáció nevét, leírását, lábainak számát és lábainak tulajdonságait kell megadni. Egy láb tulajdonságánál fontos megadni annak nevét, ugyanis a lábra való hivatkozás a későbbiekben ez alapján fog történni. A függvény visszatérési értékében jelzi, hogy sikerült-e az alkalmazás regisztrálása.

- `bump_GetInfo`: a paraméterül kapott applikációnak egy e célra létrehozott struktúrában visszaadja a leírását.

- `bump_GetApplicationsReset`: a függvény segítségével közöljük a BUMPC-el, hogy szeretnénk megtudni a hálózatban felajánlott szolgáltatásokat. Visszatérési érték nincs, a szolgáltatásokat a következő függvénnyel tudjuk kinyerni.

- `bump_GetApplications`: egyesével meghívva visszaadja a felajánlott szolgáltatások nevét, címét és azonosítóját. Ha már nincs több applikáció, mert az összeset felsorolta, akkor nincs visszatérési érték. Ez a függvény csak akkor hívható meg, ha a hívást egy `bump_getapplicationsreset` előzte meg.

- `bump_GetPINsReset`: paraméterül meg kell adni az applikáció címét és azonosítóját, amely lábainak tulajdonságát szeretnénk megtudni. Visszatérési érték nincs, a lábak nevét és tulajdonságát a következő függvénnyel tudjuk kinyerni.

- `bump_GetPINs`: egyesével meghívva visszaadja a kívánt szolgáltatás lábainak tulajdonságát. Ha már nincs több láb, mert az összeset felsorolta, nincs visszatérési érték. Csak akkor hívható meg, ha a hívást `bump_getPINsreset` előzte meg.

- `bump_SendData`: paraméterül meg kell adni a láb nevét, a küldendő adat méretét és magát az adatot. Visszatérési értekei a következők lehetnek:

  – -1, ha a kapcsolat már nem él,

  – 0, ha a láb tiltva van (nincs fókusz a hozzá kötött csatornán),

  – 1, ha az adatküldés sikerült

- `bump_ReadData`: az adat fogadására szolgál. Paraméterül meg kell adni a láb nevét, amin az adatot várja, az adat számára lefoglalt memóriaterület méretét és egy erre a területre mutató mutatót. A függvény visszatérési értékben a beolvasott adat méretét adja meg.

- `bump_RevokeService`: segítségével az applikáció lábainak felajánlását lehet visszavonni, tipikusan az alkalmazás leállításakor használható.

4.6.2. Vezérlő függvények

Vezérlő függvénynek nevezünk minden olyan API-hívást, mellyel egy kapcsolatrendszert lehet kezelni.

- `bump_RegisterControlEntity`: egy vezérlőalkalmazás e függvény segítségével tudja magát a BUMPC-nél regisztrálni. A visszatérési érték mutatja, hogy a bejelentkezés sikeres volt-e vagy sem.

- `bump_CreateSession`: egy vezérlőalkalmazás e függvény segítségével tud egy összeköttetésrendszert felépíteni. Az összeköttetendő lábakat egy speciális struktúratömbben kell megadni. A visszatérési érték mutatja, hogy a kapcsolatfelvétel sikerült-e (0: nem – nagyobb, mint nulla érték: igen, az érték továbbiakban a vezérlőapplikáción belül egyértelműen azonosítani fogja a kapcsolatrendszert).

- `bump_EndSession`: meghívásával bontani lehet a paraméterben megadott kapcsolatrendszert.

- `bump_ChangeLocalFocus`: a paraméterként átadott helyi lábon levő csatornák közötti fókuszváltásra szolgál. A visszatérési érték mutatja, hogy a váltás sikerült-e.

- `bump_ChangeRemoteFocus`: ugyanazt teszi, mint az előbbi függvény, csak egy másik eszköz lábán próbál csatornát váltani.

5. Implementáció

5.1. Előtanulmány a *Blown-up* rendszerhez

Munkánkat kezdetben a PDA-val történő munkavégzés kényelmesebbé tétele és felhasználási körének kibővítése motiválta. Ezért készítettünk egy megvalósíthatósági tanulmányt, melynek célja a jövőbeli rendszer életképességének feltérképezése volt. E tanulmány keretében megvizsgáltuk azon jelenleg rendelkezésre álló technikák körét, melyek segítségünkre lehetnek. Az átfogó megoldások mellett olyan programrendszereket is vizsgáltunk, amelyek a megszokott számítógépes architektúrákra íródtak, de képesek a *Blown-up* rendszer igényeinek kielégítésére is. A programokat kisebb nagyobb módosításokkal egy olyan alkalmazásba integráltuk, amely képes demonstrálni rendszerünk tervezett funkcióinak egy részét.

Az elkészített alkalmazás egy grafikus felhasználói felület (Graphical User Interface – GUI), mely az általunk írt C programok, shell szkriptek és egyéb alkalmazások összehangolásával lehetőséget ad, hogy bemutathassuk a BUMP protokoll képességeinek egy szűk, de felhasználói szemszögből látványos halmazát. A GUI funkcióját tekintve két elkülönülő modulra bontható, azonban használhatósági megfontolásokból egységesen implementáltuk. A kiszolgáló (szerver) modul hirdeti meg a felhasználó által felajánlott szolgáltatásokat, valamint fogadja a kliens parancsokat, míg az ügyfél (kliens) modul észleli a hálózaton jelenlévő szolgáltatásokat, és a felhasználó kérését továbbítja a kiszolgáló egység(ek) felé.

Mivel ez az implementáció egy előtanulmány, ezért csak a megoldandó probléma közös közte és a *Blown-up* rendszer között, nem követi a kidolgozott rendszer struktúráját, üzeneteit, megoldásait. Ez az implementáció csak arra fókuszál, hogy megoldást nyújtson egy PDA felhasználója számára, hogy a környezetében lévő eszközök által felajánlott billentyűzet, egér, képernyő és mp3 lejátszó szolgáltatásokat igénybe vehesse.

Felhasznált alkalmazások

Az előtanulmány során készített programcsoport négy különböző szolgáltatást képes megvalósítani. A szolgáltatások külön alkalmazások segítségével vehetők igénybe:

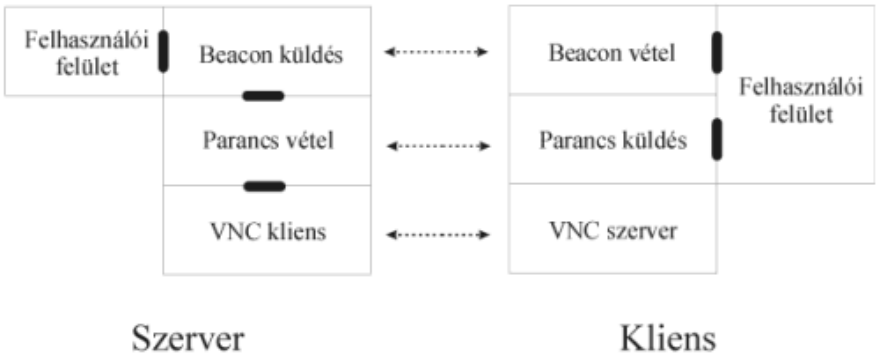
- AT&T által készített Virtual Network Computing (VNC) [VNC] lehetővé teszi a kliens munkakörnyezetének (desktop) megjelenítését egy kijelző szolgáltatást felajánló eszközön.

- Egy X hívásokkal működő *xremote* nevű program általunk készített módosításával egy számítógépen egy másik számítógép pozicionáló eszközét tudjuk használni.

- Szintén a módosított *xremote* programmal egy „távoli” billentyűzetet is igénybe vehetünk.

- Egy mp3 lejátszó programot futtató eszköz által felajánlott mp3 szolgáltatáson keresztül akár mp3-at is hallgathatunk.

E három program felhasználásával nem kellett foglakoznunk a szolgáltatás technikai megvalósításának részleteivel, mindössze arra kellett koncentrálnunk, hogy a felhasználói felületet és a programok által nyújtott lehetőségeket összehangoljuk.



15. ábra. Az előtanulmány architektúrája

5.2. A grafikus felhasználói felület

A grafikus felület megvalósításához a TCL/TK (Tool Command Language / ToolKit) [TCL] szkriptnyelvet használtuk, mely három, a cél szempontjánól különösen hasznos tulajdonsággal rendelkezik. Egyrészt platformfüggetlen, másrészt C programokkal és shell szkriptekkel egyszerűen összehangolható, harmadrészt a TCL programnyelv TK kiegészítése hatékony megoldásokat nyújt ablakos alkalmazások elkészítéséhez.

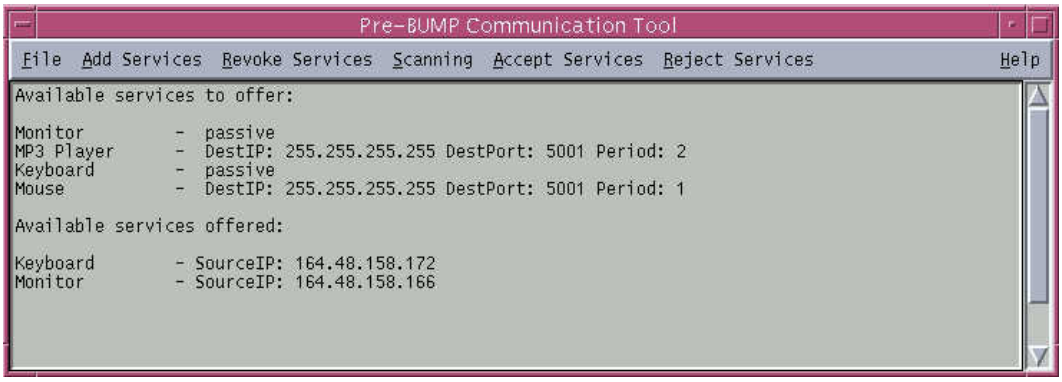
Az elkészült grafikus felhasználói felület a következő menüpontokat tartalmazza:

- **Szerver funkciók:**
  - *Add Services*: új szolgáltatás felajánlása.
  - *Revoke Services*: szolgáltatás megszüntetése.
- **Kliens funkciók:**
  - *Scanning*: szolgáltatás felderítő üzemmód bekapcsolása.
  - *Accept Service*: szolgáltatás elfogadása.

- *Reject Service*: szolgáltatás lemondása.

Amennyiben a felhasználó számára rendelkezésre áll egy igénybevehető szolgáltatás, úgy használatának kezdeményezését, majd befejezését a fent felsorolt funkciókat megvalósító menüpontokkal tudja végrehajtani.

Egy adott eszköz felhasználója számára a GUI visszajelzi, hogy milyen szolgáltatásokat ajánlott fel, illetve típus és eszközcím szerint megjeleníti, hogy a hálózatban milyen szolgáltatásokat reklámoznak. A 16. ábrán látható, hogy az adott eszköz egy mp3 és egy pozícionáló szolgáltatást ajánl fel, míg számára nincs igénybevehető szolgáltatás.



16. ábra. A grafikus felület

5.3. A jelenlegi rendszer

Rendszerünk megvalósítása jelenleg még csak a tervezési fázison van túl, ezért konkrét implementáció még nem készült. Mint azonban látható, a protokollrétegben szereplő entitások között váltott üzenetek, az üzenetváltás menete illetve az API részletesen ki van dolgozva. Az üzenettípusok és azok paramétereinek meghatározása mellett fontosnak tartottuk, hogy valamilyen protokolleíró nyelvvel az entitások üzenetekre válaszul adott viselkedését, állapotváltozásait is definiáljuk. Erre a célra egy modellező nyelvet, az SDL-t (Specification Description Language) választottuk, amely a processzeket „kommunikáló kiterjesztett véges automataként” (Communicating Extended Finite State Machine – CEFSM) ábrázolja. A függelékben a protokoll SDL leírásából néhány érdekesebbet részt mutatunk be.

6. Összefoglaló

Napjainkban már léteznek meglehetősen nagy teljesítményű és sokféle célra használható hordozható számítógépek, azonban ezek számos esetben túl nagy méretűnek bizonyulnak. Éppen ezért egyre jobban terjednek a digitális személyi asszisztensek (Personal Digital Assistant – PDA), melyek kis méretük miatt kényelmesen hordozhatók, viszont méretükből adódóan hosszabb időn át történő használatuk kényelmetlenné válhat. Ugyanakkor teljesítményük is kisebb, valamint a futtatható alkalmazások köre is szűkebb.

Mi egy olyan rendszert terveztünk, amely a felhasználó környezetében lévő ad hoc hálózatban résztvevő eszközök által nyújtott szolgáltatásokat egy személyi hálózattá (Personal Area Network – PAN) szervezi és úgy jeleníti meg a különböző szolgáltatásokat, mintha azok ugyanazon a számítógépen állnának rendelkezésre. A *Blown-Up Micronet Protocol* (BUMP) azon kívül, hogy felelős az eszközök közötti kommunikáció menedzseléséért, middleware-ként – az alkalmazások és a hálózati réteg közé illeszkedve - programozási felületet (API-t) is biztosít az alkalmazásoknak. A rendszeren keresztül gyakorlatilag bármilyen szolgáltatást össze lehet kapcsolni bármilyen másik szolgáltatással, továbbá a rendszer lehetőséget nyújt egy PDA számára külső eszközök (például egér, billentyűzet) elérésére, távoli alkalmazások (például MP3 lejátszó) használatára, ezáltal – szolgáltatástól függően – tehermentesíti a PDA processzorát, kényelmesebbé teszi a kézi számítógéppel való munkát, illetve új alkalmazások használatára ad lehetőséget.

Mivel rendszerünknek csak egy kis részét implementáltuk, ezért elsődleges jövőbeni célunk, hogy a teljes rendszert megvalósítsuk. Szeretnénk minél több alkalmazást a protokollunkkal kompatibilissé tenni és ezáltal megmutatni, hogy rendszerünk mennyire megkönnyíti mind a programozó mind a felhasználó munkáját. A későbbiekben szeretnénk megvizsgálni, hogy milyen irányban érdemes a protokollt továbbfejleszteni, milyen további modulokkal célszerű kiegészíteni.

A. Függelék

A.1. Üzenetek

Honnan	Hová	Üzenet(Paraméterek)
A	BU-C	Hello(AppID, AppName, AppDescr)
A	BU-C	RegPINReq(AppID, TP-Type, PIN-Type, I/O, M/O, Priority, Capacity, MyPINName)
A	BU-C	RegFin(AppID)
A	BU-C	RevokeApp(AppID)
CA	BU-C	ControlHello(controlAppID)
CA	BU-C	AppListReq(controlAppID)
CA	BU-C	GetInfo(serviceAddr, serviceAppID, controlAppID)
CA	BU-C	TStart(controlAppID, transactionID)
CA	BU-C	ConnPINs(linkAddr1, AppID1, MyPINName1, linkAddr2, AppID2, MyPINName2, transactionID)
CA	BU-C	TEnd(transactionID)
CA	BU-C	PINListReq(controlAppID, serviceAddr, serviceAppID)
CA	BU-C	SessionEND(controlAppID, transactionID)
CA	BU-C	ChangeRemoteFocus(serviceAddr, AppID, MyPINName)
CA	BU-C	ChangeLocalFocus(AppID, MyPINName)
BU-C	A	RegPINResp(AppID, PIN-Type, TAP-ID)
BU-C	CA	AppItems(controlAppID, NumofApps, (serviceAddr, serviceAppID)[])
BU-C	CA	Info(serviceAddr, serviceAppID, AppDescr)
BU-C	CA	TOK(transactionID, (optLinkAddr1, optAppID1, optMyPINName1, optLinkAddr2, optAppID2, optMyPINName2) [])
BU-C	CA	TNOK(transactionID)
BU-C	CA	PINItems(serviceAddr, serviceAppID, NumOfPINs, (PIN-Type, TP-Type, I/O, M/O, Capacity, TAP-ID, MyPINName)[])
BU-C	CA	SessionTerminated(bindID)

Honnan	Hová	Üzenet(Paraméterek)
BU-N	TL	DATA(TP-Type, TAP-ID, frameNum, userdata)
TL	BU-N	
BU-N	TL	DATAACK(TP-Type, TAP-ID, frameNum[])
TL	BU-N	
BU-N	BU-N	data(destAddr, TR-Type, destTAP-ID, senderAddr, userdata)
A	TL	appData(userdata)
TL	A	
BU-C	BU-C	beacon(senderAddr, AppID, AppName)
BU-C	BU-C	appPINsReq(destAddr, senderAddr, destAppID, queryID)
BU-C	BU-C	appsPINsResp(destAddr, senderAddr, sourceAppID, queryID, NumofPINs, (PIN-Type, TP-Type, I/O, M/O, Capacity, TAP-ID, MyPINName)[])
BU-C	BU-C	infoReq(destAddr, senderAddr, destAppID, queryID)
BU-C	BU-C	infoResp(destAddr, senderAddr, queryID, AppDescr)
BU-C	BU-C	bind(linkAddr1, TAP-ID1, linkAddr2, TAP-ID2, senderAddr, controlAddr, sessionID)
BU-C	BU-C	bindACK(destAddr, linkAddr1, TAP-ID1, linkAddr2, TAP-ID2, senderAddr, sessionID)
BU-C	BU-C	bindNAK(destAddr, linkAddr1, TAP-ID1, linkAddr2, TAP-ID2, senderAddr, sessionID)
BU-C	BU-C	run(BROADCAST, controlAddr, sessionID)
BU-C	BU-C	runACK(destAddr, senderAddr, sessionID)
BU-C	BU-C	unbind(BROADCAST, controlAddr, sessionID)
BU-C	BU-C	unbindme(controlAddr, senderAddr, senderTAP-ID, sessionID)
BU-C	BU-C	unbindACK(controlAddr, senderAddr, senderTAP-ID, sessionID)
BU-C	BU-C	alive(destAddr, senderAddr, senderAppID, sessionID)
BU-C	BU-C	lockyourPIN(destaddr, senderAddr, TAP-ID)
BU-C	BU-C	unlockyourPIN(destaddr, senderAddr, TAP-ID)
BU-C	BU-C	chfocus(destAddr, senderAddr, TAP-ID)
BU-C	BU-C	chfocusACK(destAddr, senderAddr, TAP-ID)

A.1.1. Küldő entitások

A	:	alkalmazások (Applications)
CA	:	vezérlő alkalmazás (Control Application)
TL	:	szállítási réteg (Transport Layer)
BU-C	:	Blown-up vezérlési réteg (Blown-up Control)
BU-N	:	Blown-up hálózati réteg (Blown-up Network)

A.1.2. Azonosítók magyarázata

AppID	:	alkalmazás azonosító
AppName	:	alkalmazás neve
AppDescr	:	alkalmazás leírása
BROADCAST	:	hálózati broadcast cím
Capacity	:	adott láb hány kapcsolatot képes kezelni
controlAppID	:	vezérlő alkalmazás azonosítója
controlAddr	:	a vezérlő eszköz címe
destAddr	:	az üzenet címzettje
destAppID	:	a célalkalmazás azonosítója
destTAP-ID	:	a fogadó fél szállítási elérési pontjának azonosítója
frameNum	:	keretszorszám
I/O	:	be-, kiviteli láb
linkAddr1	:	a láb összekötést kérő eszköz címe
linkAddr2	:	a láb összekötést kiszolgáló eszköz címe
M/O	:	kötelező, vagy választható láb
MyPINName	:	láb neve (felhasználó által definiált)
NumofApps	:	alkalmazások száma
NumofPINs	:	lábak száma

Priority	:	láb prioritása
PIN-Type	:	szolgáltatási típus (előre definiált értékek)
queryID	:	a lekérdezés azonosítója
senderAddr	:	az üzenetet küldő eszköz címe
senderAppID	:	a küldő alkalmazás azonosítója
senderTAP-ID	:	a küldő fél szállítási elérési pontjának azonosítója
serviceAddr	:	szolgáltatást nyújtó eszköz címe
serviceAppID	:	szolgáltatást nyújtó alkalmazás azonosítója
sessionID	:	a kapcsolatcsoport azonosítója
TAP-ID	:	a szállítási réteg elérési pontjának azonosítója
TP-Type	:	használt szállítási típus
transactionID	:	alkalmazás által kezdeményezett kapcsolatcsoport kiépítése során keletkezett azonosító később ezzel tud hivatkozni a kapcsolatcsoportra
userdata	:	alkalmazások által küldött adat

A.2. API függvények és leírásaik

API függvények	Leírás
függvény neve:	`bump_RegisterApplication`
paraméterei:	`string AppName, string AppDescr, int PINnumber, struct PIN[]`
visszatérési értéke:	`success/failure`
függvény neve:	`bump_RegisterControlEntity`
paraméterei:	–
visszatérési értéke:	`success/failure`
függvény neve:	`bump_GetInfo`
paraméterei:	`destAddr, destAppID, string AppDescr`
visszatérési értéke:	`struct info`
függvény neve:	`bump_GetApplicationsReset`
paraméterei:	–
visszatérési értéke:	`void`
függvény neve:	`bump_GetApplications`
paraméterei:	–
visszatérési értéke:	`struct AppItem/NULL`
függvény neve:	`bump_PINsreset`
paraméterei:	`struct AppItem`
visszatérési értéke:	`void`
függvény neve:	`bump_GetPINs`
paraméterei:	–
visszatérési értéke:	`struct UserPIN/NULL`
függvény neve:	`bump_CreateSession`
paraméterei:	`struct PINPairs[], int NumofPairs`
visszatérési értéke:	`-1 Hiba / bindID`
függvény neve:	`bump_EndSession`
paraméterei:	`bindID`
visszatérési értéke:	`void`

API függvények	Leírás
függvény neve:	`bump_RevokeService`
paraméterei:	–
visszatérési értéke:	`void`
függvény neve:	`bump_SendData`
paraméterei:	`string MyPINName, int length, void* data`

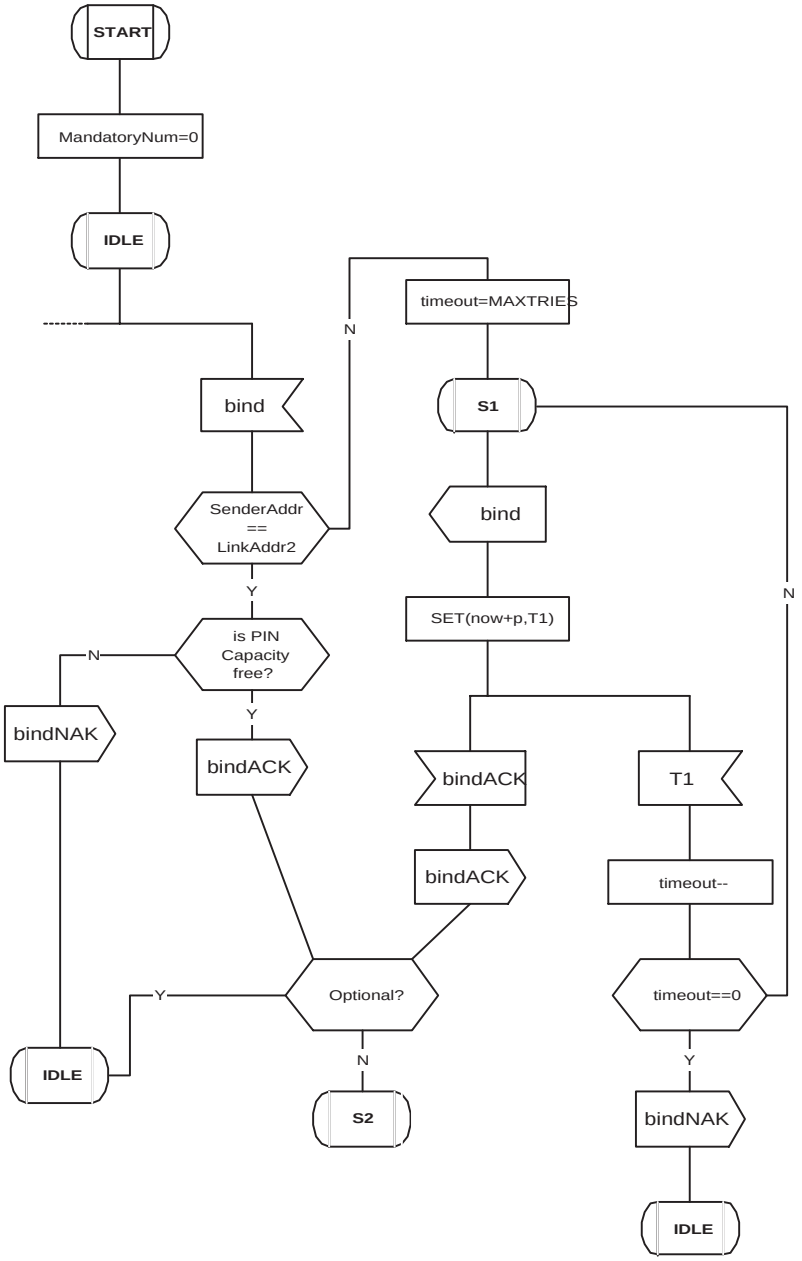
visszatérési értéke:	`-1, ha a nincs csatorna kapcsolódva`
	` 0, ha le van tiltva a láb`
	`>0, elküldött byte-ok száma`

függvény neve:	`bump_ReadData`
paraméterei:	`string MyPINName, int maxbuffersize, void* buffer`
visszatérési értéke:	`-1, ha nincs csatorna kapcsolódva` `>0, fogadott byte-ok száma`
függvény neve:	`bump_ChangeRemoteFocus`
paraméterei:	`serviceAddr, int serviceAppID, string MyPINName`
visszatérési értéke:	`void`
függvény neve:	`bump_ChangeLocalFocus`
paraméterei:	`string MyPINName`
visszatérési értéke:	`void`

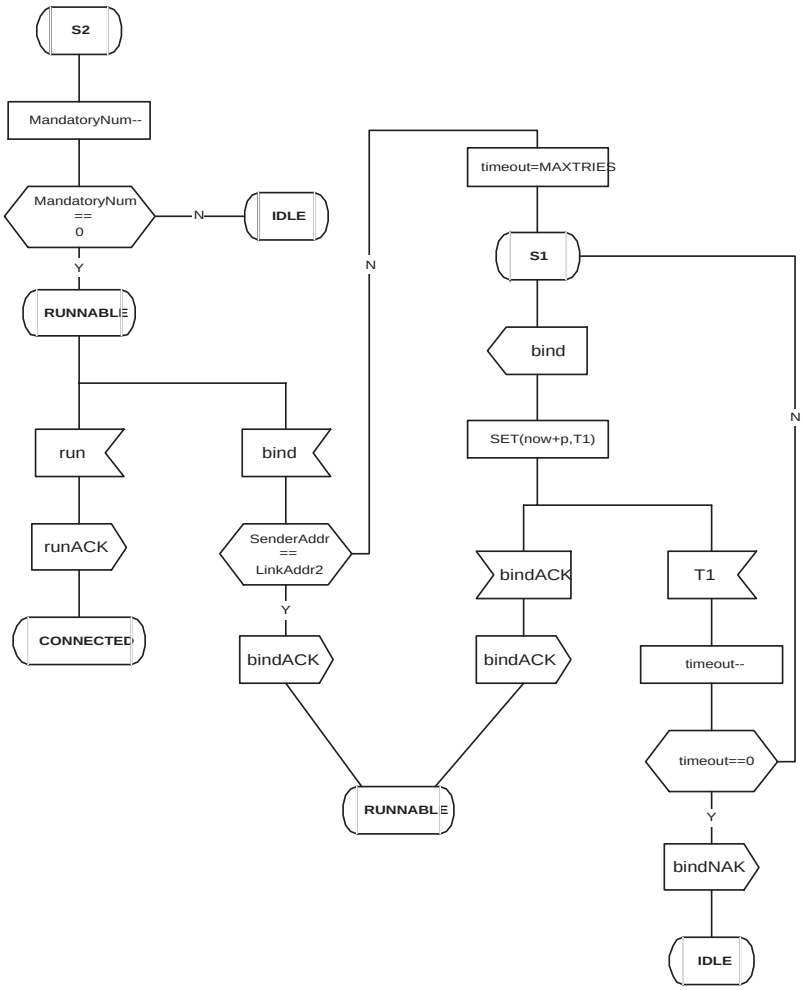
A.3. API struktúrák és leírásaik

API struktúrák	Leírás
struktúra neve:	`PIN`
elemei:	`enum TP-Type, int PINType, bool I/O, bool O/M, int Priority, int Capacity, string MyPINName`
struktúra neve:	`UserPIN`
elemei:	`enum TP-Type, int PINType, bool I/O, bool O/M, int Capacity, string MyPINName`
struktúra neve:	`info`
elemei:	`addr ServiceAddr, int ServiceAppID, string AppDescr`
struktúra neve:	`AppItem`
elemei:	`addr ServiceAddr, int ServiceAppID`
struktúra neve:	`PINPairs`
elemei:	`addr Link1Addr, addr Link2Addr, int Link1AppID addr Link2AppID, string Link1MyPINName, string Link2MyPINName`

A.4. SDL ábrák



17. ábra. Csatorna felépítése



18. ábra. Csatorna felépítése (folytatás)

Rövidítések

ACK – Acknowledgement
AODV – Ad hoc On-demand Distance Vector
API – Application Programming Interface
BU-C – BUMP controller
BU-N – BUMP-Network
BUMP – Blown-Up Micronet Protocol
BUMPC – BUMP controller
CEFSM – Communicating Extended Finite State Machine
DSR – Dynamic Source Routing
E21 – Enviro21
GUI – Graphical User Interface
H21 – Handy21
HTML – Hypertext Markup Language
HTTP – Hypertext Transfer Protocol
IC – Integrated Circuit
IEEE – Institute of Electrical and Electronics Engineers
IETF – The Internet Engineering Task Force
IP – Internet Protocol
IU – Information Utility
MAC – Medium Access Control
MACA – Multiple Access with Collision Avoidance
MANET – Mobile Ad Hoc Networks
MPEG – Motion Picture Experts Group
N21 – Network21
PAN – Personal Area Network
PDA – Personal Digital Assistant
QoS – Quality of Service
RDP – Resource Discovery Protocol
SDL – Specification Description Language
TAP – Transaction Access Point
TCL/TK – Tool Command Language/ToolKit
TCP – Transmission Control Protocol
TP – Transport
VNC – Virtual Network Computing
WLAN – Wireless Local Area Network

Hivatkozások

[Weiser91] Mark Weiser: *„The Computer for the 21st Century”*, Scientific American, September 1991.

[Weiser93] Mark Weiser: *„Some Computer Science Issues in Ubiquitous Computing”*, Communications of the ACM, July 1993.

[Satya01] M. Satyanarayanan: *„Pervasive Computing: Vision and Challenges”*, IEEE Personal Communications, August 2001.

[BPT96] P. Bhagwat, C. Perkins, S. Tripathi: *„Network Later Mobility: an Architecture and Survey”*, Personal Communications Magazine, Vol. 3, No. 3, June 1996.

[BMJHJ98] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, J. Jetcheva: *„A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols”*, MobiCom ’98.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, R. Katz: *„Improving TCP/IP Performance Over Wireless Networks”*, MobiCom ’95.

[Aura02] D. Garlan, D. P. Siewiorek, A. Smailagic, P. Steenkiste: *„Aura: Toward Distraction-Free Pervasive Computing”*, IEEE Pervasive Computing, 2002.

[Oxygen02] *„MIT Project Oxygen”*, Online Documentation, http://oxygen.lcs.mit.edu/publications/Oxygen.pdf

[Porto99] M. Esler, J. Hightower, T. Anderson, G. Borriello: *Next Century Challenges: Data-Centric Networking for Invisible Computing: The Portolano Project at the University of Washington*, Mobicom ’99.

[PortoWeb99] *Portolano/Workscape: Charting the new territory of invisible computing for knowledge work*, Online Documentation, http://portolano.cs.washington.edu/proposal/

[Endeav99] *The Endeavour Expedition: Charting the Fluid Information Utility*, Online Documentation, http://endeavour.cs.berkeley.edu/proposal/

[Speakeasy02] W. K. Edwards, M. W. Newman, J. Sedivy, T. Smith: *„Challenge: Recombinant Computing and the Speakeasy Approach”*, MobiCom’02, September 23-28, 2002, Atlanta, Georgia, USA.

[DPR00] S. Das, C. Perkins, E. Royer: *„Performance Comparison of Two On-demand Ad hoc Routing Algorithms”*, Proceedings of the IEEE Conference on Computer Communication, March 2000.

[TCL] Brent B. Welch: *„Practical Programming in TCL and TK”*, ISBN 0-13-182007-9

[JH98] Jaap Hartsen: *„BLUETOOTH – The universal radio interface for ad hoc, wireless connectivity”*, Ericsson Review No. 3, 1998

[PC97] Vincent D. Park and M. Scott Corson: *A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks*, Proceedings of IEEE INFOCOM ’97, Kobe, Japan (April 1997)

[PB94] C. Perkins, P. Bhagwat: *"Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers"*, SIGCOMM’94

[BBSpec] *Bluetooth Baseband Specification*, http://www.bluetooth.com

[WLAN99] *IEEE Std 802.11, 1999 Edition*, http://standards.ieee.org/catalog/olis/lanman.html

[HLAN2] *HiperLAN2 overview*, http://www.hiperlan2.com/WhyHiperlan2.asp

[Karn90] Phil Karn: *MACA - A New Channel Access Method for Packet Radio*, appeared in the proceedings of the 9th ARRL Computer Networking Conference, London, Ontario, Canada, 1990

[MANET] Mobile Ad hoc Networking (MANET): *Routing Protocol Performance Issues and Evaluation Considerations (RFC 2501)*, http://www.ietf.org/rfc/rfc2501.txt

[MPEG] MPEG homepage, http://mpeg.telecomitalialab.com/

[VNC] VNC homepage, http://www.uk.research.att.com/vnc

Ábrák jegyzéke