Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Nagy Ákos

SZOFTVER-HASZNÁLHATÓSÁG ÉS FELHASZNÁLÓI ÉLMÉNY MÉRÉSE MOBIL ESZKÖZÖKÖN SZENZORADATOK ANALÍZISÉVEL

KONZULENS

Kővári Bence

BUDAPEST, 2012

Tartalomjegyzék

Összefoglaló

Ahogyan a számítógépek a mindennapi élet részévé váltak, úgy jelentek meg újabb és újabb kihívások a szoftverfejlesztésben is. Ma egy alkalmazás készítésekor nem csak arra figyelünk oda, hogy funkcionálisan megfeleljen a specifikációnak, hanem arra is, hogy a felhasználók a lehető legrövidebb tanulási szakasz után a lehető leghatékonyabban használhassák az alkalmazást.

A mobil eszközök fejlődése további változásokat hozott a szoftverfejlesztésbe. Az ilyen eszközökön (akár mobiltelefonokról, PDA-król, vagy tabletekről van szó) a szokásos beviteli módok mellett megjelennek olyan interakciós lehetőségek, mint az érintőképernyő vagy különböző mozgásszenzorok. A felhasználóknak meg kell küzdeniük ezekkel az új eszközökkel, meg kell tanulniuk használni őket. Ahhoz, hogy ez minél hatékonyabban történhessen, a szoftverek fejlesztőinek meg kell ismerniük a felhasználók eszköz- és alkalmazáshasználati szokásait.

Az ilyen adatok begyűjtése és elemezése időigényes feladat, viszont adott alkalmazás üzleti sikeréhez elengedhetetlen. A folyamat automatizálása komoly kihívás, viszont hatékony megvalósítás esetén nagyban hozzájárulhat az elkészült alkalmazás sikerességéhez.

Dolgozatomban egy Windows 8 platformra készült referencia-alkalmazáson keresztül térképeztem fel, hogy milyen adatokat érdemes gyűjteni a felhasználói interakciókról és a szenzorokból. Külön hangsúlyt fektettem arra, hogy a folyamat automatizált legyen és alkalmazásához a lehető legkevesebb módosítást kelljen végezni a szoftver forráskódjában. Keretrendszerem működését egy szoftver bevezetése kapcsán készült esettanulmányban vizsgáltam. A gyűjtött adatokat különböző statisztikai módszerekkel és vizualizációs eszközökkel feldolgoztam, értelmeztem és következtetéseket vontam le az általános felhasználói magatartással kapcsolatban.

Az elkészült komponens könnyen beilleszthető modulként már kész szoftverekbe, valamint a mérési keretrendszer könnyen felhasználható más mobil Windows platformok vizsgálatára is, így az eredmények könnyen általánosíthatóak egyéb mobil környezetekre is. A levont következtetések egyrészt segíthetnek megérteni a felhasználók igényeit, másrészt pedig olyan hibákra mutathatnak rá az alkalmazásban, amik a funkcionális tesztelési módszerek előtt rejtve maradnak.

1 Bevezetés

Az alkalmazások fejlesztésekor szoftverfejlesztőként olyan követelményeket is figyelembe kell vennünk, amik nincsenek megadva a specifikációkban. A megfelelően implementált funkcionalitás mellett cél az is, hogy a felhasználók a lehető legrövidebb tanulási szakasz után a lehető leghatékonyabban használhassák az alkalmazást. A fejlesztésbe gyakran a programozók mellett olyan szakemberek is bekapcsolódnak, akiknek a szakterületük nem tervezési minták hatékony megvalósítása vagy egy szolgáltatás-orientált architektúra elemeinek a kidolgozása, hanem a szoftver-ergonómia, a felhasználói élmény vagy a szoftver-használhatóság.

A mobil eszközök fejlődése tovább emelte az előzőekben említett témakörök fontosságát. Az érintőképernyő és a mozgásszenzorok, valamint maga a mobilitás ténye, ha megfelelően vannak kihasználva az alkalmazásban, nagyban hozzájárulhatnak a felhasználóktól érkező pozitív visszajelzésekhez.

A felhasználók ma már nem csak egyszerű berregő gépezetként tekintenek a számítástechnikai eszközökre – elvárják, hogy kényelmesen, gyorsan használható alkalmazások készüljenek, amik a lehető legjobban alkalmazkodnak az ő szokásaikhoz. Ahhoz, hogy ilyen alkalmazásokat fejlesszünk, meg kell ismerni és meg kell érteni a felhasználók szokásait, viselkedését.

Munkám során egy olyan eszköz elkészítését tűztem ki célul, ami segít ebben a szoftverfejlesztőknek. Az elkészített modul (a továbbiakban: keretrendszer) különböző adatokat gyűjt a felhasználók szokásairól, hogy később ezt analizálni lehessen. Ezek között az adatok között vannak olyanok, amelyek felhasználói interakciókból származnak és olyanok, amelyek az eszközök beépített szenzoraiból származnak. Fontos további célkitűzésem volt, hogy a komponens a lehető leggyorsabban, a lehető legkevesebb munkával beépíthető legyen már meglévő alkalmazásokba és ezután az adatgyűjtés automatikusan működjön.

A dolgozatomban elsőként a felhasználói élmény és a használhatóság jelentőségét tárgyalom. Ezután bemutatom a mobil eszközök terjedésének gyors ütemét, kiemelve fontos és erős tendenciákat és utalok ezek hatására a szoftverfejlesztésben.

A keretrendszer referencia-implementációját Windows 8 platformra készítettem el. Ez az operációs rendszer még nagyon fiatal (a dolgozat írásának pillanatában még a boltok polcain meg sem található) – ezért röviden bemutatom az újdonságait, majd külön kitérek a munkám szempontjából fontos tulajdonságaira.

Ezt követően bemutatom a kész modult, amin keresztül pontos leírását adom meg a keretrendszer koncepcionális működésének is. Kitérek a gyűjtött adatokra (események és szenzoradatok), az adatgyűjtéshez használt adatmodellre és részleteiben leírom az automatizált folyamatot lépésenként: a keretrendszer inicializálása, adatok gyűjtése, tárolása, küldése a szerverre.

Miután a keretrendszert bemutattam, megmutatom milyen következtetéseket lehet levonni a gyűjtött adatokból – ehhez egy népszerű magyar aukciós oldalnak fejlesztett Windows 8 alkalmazásba építettem bele a keretrendszeremet. A levont következtetések hasznosak egyrészt az alkalmazás értékelésére, másrészt a keretrendszer koncepcionális helyességének bizonyítására.

Az adatelemzést követően pedig összehasonlítom a munkámat egy már létező, hasonló keretrendszerrel, legutoljára pedig rámutatok néhány továbbfejlesztési lehetőségre.

1.1 Felhasználói élmény és jelentősége

A felhasználói élmény (user experience, UX) az újgenerációs szoftverek készítésének egyik központi kérdése, meghatározó szerepe van a szoftverfejlesztés minden szakaszában. Pontos definíciót nehéz adni – Jesse James Garrett, UX-guru a következőként írja le [1]:

„Az élmény, amit a termék kivált a felhasználóban, amikor valós körülmények között használja azt.”

Érdemes megfigyelni, hogy a definíció – igaz semmilyen területen nem túl specifikus – nem tartalmaz közvetlen utalást szoftverekre, egyszerűen csak „termék”-ként hivatkozik arra a valamire, amivel kapcsolatban a felhasználói élményt tárgyalni érdemes.

A fenti definícióból kiindulva tehát érdemes egy újabb meghatározást megvizsgálni: mi is a az a termék?

“Emberi munkának valamely szükséglet kielégítésére alkalmas eredménye.”

Az előző definíció a Magyar Értelmező Kéziszótárból [2] származik. Bár ez a definíció sem túl specifikus, látható, hogy a szoftver – a jól megírt szoftver legalábbis mindenképpen – valóban termék, azaz igenis értelmezhető rá a felhasználói élmény fogalma.

Ha pedig értelmezhető a felhasználói élmény fogalma, akkor teljes nyugodtsággal adhatunk hitelt azoknak a tudományos igényű vizsgálatoknak, amelyek rámutatnak az UX szerepére a termékek piaci szereplésében [3].

Látható tehát, hogy a felhasználói élménnyel igenis foglalkozni kell, érdemes az időt és az energiát belefektetni.

1.2 Szoftver-használhatóság

A szoftver-használhatóság szintén központi kérdés a szoftverfejlesztésben. Mi sem bizonyítja ezt jobban, mint hogy komplett tesztelési módszertanok léteznek arra, hogy a használhatóságot vizsgáljuk. Az International Software Testing Qualifications Board különböző tesztelői minősítéseinek megszerzéséhez például szükséges rendelkezni ismeretekkel a használhatósági tesztekről is. Léteznek különböző kérdőívek, amelyekkel mérhető a használhatóság – ilyen például a Software Usability Measurement Inventory [4]. Az International Organization for Standardization 9126 számú szabványában külön foglalkozik a használhatósággal, ahol következők vannak lefektetve ezzel kapcsolatban (fordítás, az eredeti szöveget lásd [5]):

"*Olyan tulajdonságok csoportja, amik hatással vannak a használathoz szükséges erőfeszítésekre és az ilyen használat egyéni megítélésére felhasználók egy meghatározott vagy vélelmezett csoportjának szempontjából.*"

A definícióban említett tulajdonságokat a szabvány fel is sorolja:

- Érthetőség
- Tanulhatóság
- Működtethetőség
- Vonzóság
- Használhatósági megfelelés

Jól láthatóan a felhasználói élmény egyik fontos alkotóeleméről van szó – ami jól használható a rendeltetés szerinti céljára valós környezetben, az **lehet**, hogy pozitív élményt vált majd ki a felhasználóból. Ha viszont a felhasználó nem képes a feladatait elvégezni a szoftver segítségével, akkor **biztosan** negatív tapasztalatai lesznek vele kapcsolatban.

1.3 Mobil eszközök jelentősége

A mobil eszközök jelentősége, hatása megkérdőjelezhetetlen. Ezt jól példázzák a közelmúltból gyűjtött adatok és előrejelzések:

- Az ITU adatai alapján 2011-ben 5,9 milliárd regisztrált mobil-előfizetés volt a világon. Ez azt jelenti, hogy a világ teljes akkori népességének 87%-a rendelkezett mobil előfizetéssel. Összehasonlításképpen 2007-ben ugyanez a szám 3,5 milliárd körüli értéken volt – azaz 4 év alatt közel 45%-kal nőtt az előfizetők száma. További fontos adat, hogy a fejlődő

országokban is már a népesség 79%-a rendelkezik valamilyen mobil-előfizetéssel [6].

- A PortioResearch adatai alapján 2016-ra a 8 milliárdot is át fogja lépni a mobil-előfizetők száma [7]. Az ENSZ becslései alapján akkor a világ népessége 7,3 milliárd körül fog mozogni [8] – ez tehát azt jelenti, hogy a világ minden emberére több mint 1 egész mobil-előfizetés jut majd. Hasonló tendenciákról számol be a Reuters is egyik cikkében [9].
- Szintén az ITU 2011-es adatai alapján több mint kétszer annyi mobil-előfizetés van érvényben a világon, mint hagyományos vonalas előfizetés [6].
- A PortioResearch adatai alapján 2011-ben több mint 420 millió okostelefont szállítottak le. Ez 2010-hez képest több mint 40% növekedés, 2016-ban pedig ez a szám át fogja lépni az 1 milliárdot [7].
- Az IDC adatai alapján 2010-ben 19,4 millió tabletet adtak el. 2011-ben ez a szám már 68,7 millió volt, 2012-re több mint 106 millió, 2016-ra pedig több mint 198,2 millió eladott eszközt jósolnak [10].
- A PortioResearch adatai alapján 2011-ben a mobilalkalmazások felhasználói bázisa 122,1 millió fő volt, 2016-ra pedig már 565,4 milliót jósol ez a tanulmány [7].

Érdemes kicsit jobban átgondolni, hogy mit jelent pontosan ez a hatás.

Az ilyen eszközök egy új interakciós modellt mutattak be a felhasználóknak. Eddig rendelkezésükre állt egy rejtélyes – és rejtélyes módon egérnek nevezett – eszköz, amit az asztalon mozgatva a képernyőkön ki tudtak jelölni dolgokat, elindítani programokat. Ez az eszköz volt a „mediátor” a szándékaik és cselekvéseik között.

Az új modellből azonban többnyire hiányzik ez a mediátor. Az érintőképernyő lehetővé teszi, hogy egyszerűen az ujjaikkal – a számukra természetes eszközökkel – jelöljenek ki, nagyítsanak, lapozzanak. Bár ez az új modell sokkal természetesebbnek hat, mint az előző, mégis meg kell tanulniuk, meg kell szokniuk a használatát – különösen annak a régebbi generációnak, akik már hozzászoktak az egérhez.

Ezek az új eszközök rendelkeznek még egy evidens, de annál hasznosabb tulajdonsággal: hordozhatóak. Ez egyrészt azt jelenti, hogy egyes használati helyek között könnyen mozgathatóak, másrészt pedig azt, hogy akár használat közben sem kell egy helyben maradniuk – ebben pedig rengeteg lehetőség van (például remek autó-szimulátorokat lehet megvalósítani így, ahol az eszköz maga a kormány, lásd [11]). Ezeknek a lehetőségeknek a kihasználását segítik a szenzorok: olyan eszközök, amelyek

fizikai hatásokat (mágneses erőtér, gyorsulás) mérnek, és ebből következtetnek az eszköz aktuális pozíciójára, környezetére.

A szenzorokat azonban nem csak aktívan használhatjuk fel alkalmazásainkban (mint például az előző szimulátor példában), hanem passzívan is. Ennek egyik formája az, amikor a szenzorok mérési eredményeit folyamatosan figyeljük és a környezet változásaihoz alkalmazkodik az alkalmazás (a fényszenzor mérései alapján például a kijelző fényerejét módosíthatjuk a környezetnek megfelelően).

Másik passzív felhasználási lehetőség az adatgyűjtés. A gyűjtött adatok vizsgálatával pedig – ahogyan a dolgozat későbbi fejezetiben bemutatom – olyan kérdésekre adhatunk választ, hogy a felhasználó világosban vagy sötétben használja-e inkább az alkalmazást, séta közben vagy álló helyzetben, alaporientációban vagy elforgatva.

2 Technológiai háttér: Windows 8 platform és Windows Runtime API

A Windows 8 a Microsoft operációs rendszerének legújabb kiadása, amely sokban eltér az eddigi Windows verzióktól. Érdemes néhány különbséget áttekinteni, hogy jobban átérezzük a dolgozat motivációit és céljait.

2.1 Windows 8 platform

Az új platform hibrid platform – mind asztali, mind mobil környezetben elérhetőek ugyanazok a szolgáltatások megfelelő hardveres környezet biztosítása esetén. Ha az erőforrások ezt nem teszik lehetővé, akkor az asztali gépekbe szánt verzió képességeinek egy részével rendelkező operációs rendszer futtatható a gyengébb eszközön – ez azonban valóban az eredeti egy részhalmaza, nem pedig módosított vagy újraimplementált változata.

Maga hibrid megoldás önmagában még nem jelent igazi előrelépést a felhasználói élmény szempontjából, hiszen az eddigi alkalmazások nem az 1.1 részben bemutatott új interakciós modellt figyelembe véve készültek (gondoljunk csak például arra, hogy milyen nehézkes egy menüelemet ujjal kiválasztani). Ezért az új operációs rendszerrel együtt megjelent egy új alkalmazásfejlesztési stílus és egy új alkalmazáscsoport: a „Microsoft design principles” adja az útmutatást, a „Microsoft design language” pedig az eszközöket az új, „Windows Store” (korábban Metro stílusú) alkalmazások fejlesztéséhez.

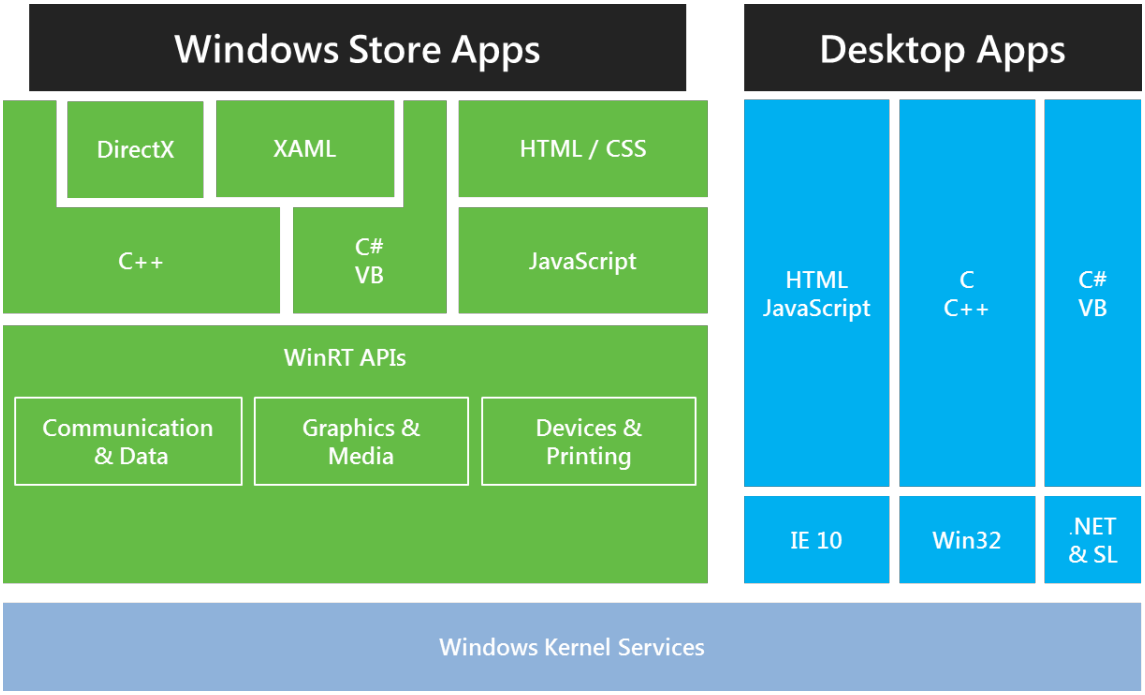
Az új elvek között szerepel az, hogy az alkalmazások futása gyors és folyamatos („fast and fluid”). Ez lényegében azt hangsúlyozza, amire már az előző fejezetekben is kitértem (lásd az 1.1 részben): a felhasználók gyorsabban tudnak interakcióba lépni az eszközzel, így az eredményeket is rögtön és gyorsan várják.

Külön hangsúlyt kap az, hogy érintésre tervezzünk – azaz ne az asztali alkalmazásokat kelljen ujjal nyomogatni, hanem a felület kifejezetten erre legyen optimalizálva.

Egy másik hangsúlyos újdonság a felület építőköveként megjelenő, „csempe” néven emlegetett elem. A csempe feladata nem csak annyi, hogy leváltsa az eddig megszokott ikonokat, hanem önmagában is tartalmat szolgáltasson. Az ikon szerepe mindössze annyi volt, hogy segített a felhasználónak azonosítani azt a funkciót vagy tartalmat, amit az ikonra kattintva elért. A csempe viszont önmagában is rendelkezik

tartalommal, ami folyamatosan változik („élő csempe”) – és mellesleg mögötte még több tartalom rejlik. A tartalom a kulcsszó – ennek hatékony és kellemes prezentációja válik elsődlegessé a platform alkalmazásaiban.

Ezeket az új eszközöket a Windows Runtime API szolgáltatásain keresztül vehetjük igénybe. Ez az API tartalmazza azokat a vezérlőket, amelyeket elhelyezhetünk a felhasználói felületen és definiálja azokat a szolgáltatásokat, amelyekkel a felület élőnek hat. Az API egyik fontos részét képzik az aszinkron metódusok, amelyek segítenek elérni – sőt, szinte kierőszakolják –, hogy a felhasználói felület hosszabb metódushívások esetén se blokkolódjon. Az új API az eddig is létező Win32 API mellé került bevezetésre teljesen egyenjogú programozási felületként, kibővítve a programozók eszköztárát olyan lehetőségekkel, mint a JavaScript alkalmazások fejlesztése vagy a HTML nyelven leírt felhasználó felületek létrehozása:



1. ábra - A Windows 8 platform [28]

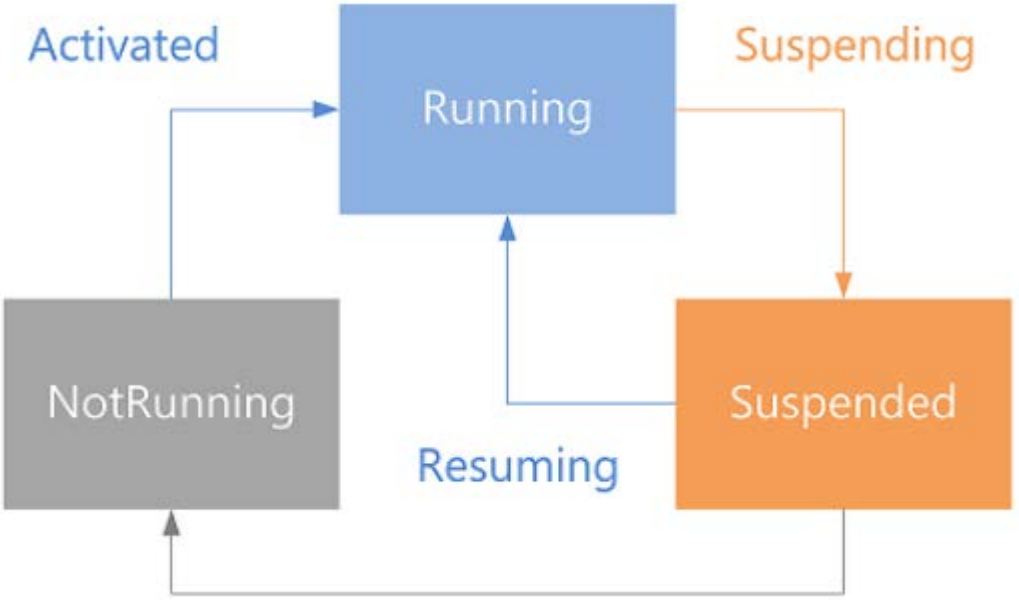
Az 1. ábra jól szemlélteti az új lehetőségeket és elhelyezkedésüket a már meglévő architektúrában. Ezek teszik lehetővé annak a teljesen új fejlesztési és alkalmazásfuttatási-modellnek a működését, ami figyelembe veszi a fentebb említett alapelveket és alkalmazkodik a mobil eszközök hardveres korlátaihoz, különös figyelemmel az akkumulátor kapacitására.

A dolgozat szempontjából különösen fontos áttekinteni az Windows Runtime API-nak azokat a részeit, amelyek az alkalmazások életciklusát és navigációs modelljét támogatják, amelyek az eseménykezelést valósítják meg, valamint amelyek az érintőképernyő és a további szenzorok szolgáltatásainak elérését és felhasználását

lehetővé teszik. Ezekre a technológiai részleteknél folyamatosan vissza fogok utalni az adatgyűjtés megvalósításáról szóló fejezetekben.

2.1.1 Az alkalmazások életciklusa

A Windows Store alkalmazások életciklusa eltér az asztali Windows platformon eddig megszokottaktól, viszont erősen emlékeztet például a Windows Phone platform alkalmazásainak életciklusára:



2. ábra - A Windows Store alkalmazások életciklusa [12]

A 2. ábra állapotai közül a Running felel meg annak, hogy az alkalmazás fut. A platformon – a korlátozott erőforrások jobb kihasználásának érdekében – legfeljebb két alkalmazás futhat egyszerre, ha ennél többet indítunk, akkor egyesek átkerülnek Suspended állapotba. Ilyenkor az alkalmazások „alvó” állapotban vannak, amiből természetesen újra átkerülhetnek Running állapotba, ha a felhasználó újra előtérbe hozza őket. Ezen kívül van még egy NotRunning állapot – az alkalmazás ide kerül rögtön telepítés után, amikor még egyszer sem volt futtatva, vagy Suspended állapotból például valamilyen kezeletlen kivétel vagy erőszakos felhasználói megszakítás után.

Amit fontos észrevenni, hogy nincs közvetlen állapot, ami az alkalmazás bezárása vonatkozna. Ezzel összhangban a programozási modellben sincs erre semmilyen eszköz – nincs Closed esemény és a felhasználói felületen is **tilos** olyan opciót biztosítani a felhasználónak, amivel bezárhatja az alkalmazást (különben nem publikálható a Windows Áruházba az alkalmazás). Természetesen ettől függetlenül bezárhatóak a programok, de általános javaslat, hogy az életciklus végének kezelését bízza a felhasználó az operációs rendszerre [12].

Az események naplózásakor fontos szerepe van az életciklus-eseményeknek, elemzéskor értékes információkkal szolgálhatnak, másrészt az életciklus ismerete segít a munkamenet fogalmának definiálásában.

2.1.2 Az alkalmazás navigációs mechanizmusa

A navigációs mechanizmus nagyon hasonlít a webes modellhez, valamint – az életciklushoz hasonlóan – erősen emlékeztet a Windows Phone platform megoldására.

Az alkalmazás lapokból épül fel. A lapok valamilyen tartalmat jelenítenek meg, a tartalom egyes darabjaira kattintva – a linkekhez hasonlóan – váltogathatunk a lapok között.

Az alkalmazáshoz tartozik egy Frame – ez tartalmazza mindig az aktuális lapot és ez biztosítja a navigáláshoz szükséges keretrendszert is (metódusok a navigáláshoz, események, paraméterek, navigációs verem). Navigáció során meg kell adnunk, melyik lapra szeretnénk navigálni, majd egy ilyen lap példányosodik és kerül megjelenítésre a Frame-ben. Lapváltás előtt váltódik ki a Navigating, lapváltás után a Navigated esemény.

Ezekről szintén érdemes tudni, hiszen a keretrendszer egyik alapvető feladata, hogy naplózza az alkalmazásban történt lapváltásokat, amelyekből nagyon hasznos adatokat nyerhetünk és kifejezetten értékesek a következtetések, amelyek levonhatóak belőlük. Mindemellett az egyéb események naplózásának mechanizmusában is fontos szerepet fog játszani a navigáció.

Látható tehát, hogy a Windows Store alkalmazások (a klasszikus játékok kivételével) valóban leginkább egy webhelyre hasonlítanak: a lapok, a lapok közötti váltás analóg a webes környezetben tapasztalt működéssel. Ez később, az adatok elemzésénél lesz hasznos, mert erre a felismerésre alapozva referenciaként az eddigi webes statisztikákat is felhasználhatjuk.

2.1.3 Eseménykezelés

A .NET platform az események kezelésére kész megoldást kínál. Először bevezeti a delegate fogalmát: ez egy típusos metódusreferencia. Önálló típusokat hozhatunk létre – tehát felügyelt marad a kezelése – ezeket példányosíthatjuk és a típusnak megfelelő metódusokat (akár többet is), felírhatunk a delegate hívási listájára (invocation list). A delegate meghívásakor ezek a metódusok futnak le.

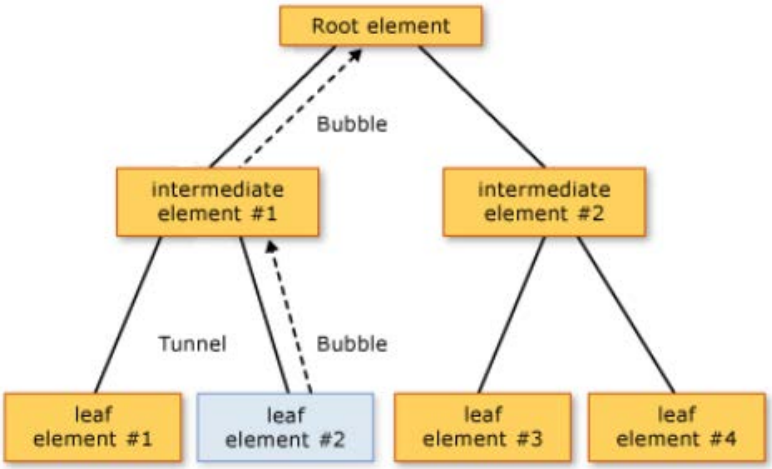
A delegate bevezetése után az esemény (event) bevezetése egy egyszerű lépés: az esemény olyan delegate, amit újrapéldányosítani csak a tulajdonos tud, illetve a fel- és a leiratkozás, valamint az esemény kiváltása szintén a tulajdonos hatásköre. Maga a

működési mechanizmus azonban azonos a delegate működésével – egy híváslistára felírt metódusok hajtódnak végre.

A Windows Runtime API azonban egy teljesen új mechanizmust vezet be az események kezelésére. Bevezeti az EventRegistrationToken fogalmát – ez egy olyan egyszerű struktúra, amit akkor kapunk vissza, amikor egy Windows Runtime eseményre feliratkozunk egy metódussal. Később ezzel a tokennel lehet azonosítani ezt a feliratkozást és lehet leiratkozni. Ezek a tokenek egy EventRegistrationTokenTable típusú objektumban vannak eltárolva, az egész mechanizmust (fel-és leiratkozás automatizálása, események meghívása) pedig a WindowsRuntimeMarshall statikus osztály kezeli.

Fontos látni különbséget a két megoldás között. .NET környezetben dinamikusan feliratkozni egy eseményre – bár kétség kívül nem kezdőknek szánt feladat – viszonylag egyszerűen megoldható, csupán ismerni kell a működési mechanizmust és az ilyen feladatokra a keretrendszerben biztosított reflexiós (Reflection) API-t. A Windows Runtime platformon azonban ez a feladat jóval körülményesebb – egyrészt az eltérő eseménykezelés miatt, másrészt pedig a .NET Reflection API teljes hiánya miatt. A részleteket egy későbbi fejezetben ismertetem.

Az eseménykezelés egy másik fontos aspektusa a buborékozás (bubbling). Magát a technikai megoldást a Microsoft már korábbi keretrendszereiben is bevezette (WPF, Silverlight) és a Windows 8 platformon is meghagyta, de szinte minden, felhasználói felületek felépítésére szolgáló rendszer támogat valami hasonlót. Ilyenkor az adott vezérlőben kiváltódó eseményeket nem csak az adott vezérlő, hanem az adott vezérlőt tartalmazó szülő is megkapja (ha definiál ilyen eseményt) és kezelheti:



3. ábra - Eseménybuborékozás [13]

Ezzel egyrészt lehetőségünk van arra, hogy egy adott konténeren belül egységesen kezeljünk bizonyos eseményeket, másrészt bizonyos feladatokat sokkal

közelebb tudunk hozni a megfelelő vezérlőkhöz (pl. konténer háttérszínét gombnyomásra változtatni).

Az események buborékozása egy olyan jelenség, amit kezelni kell az események naplózásakor, hiszen egy esemény a fában felfelé utazva többször is kiváltódik, így akár többször is naplózásra kerülhet. Ez az elgondolásaim szerint nem kívánatos – ennek részletes indoklása megtalálható a későbbi fejezetekben.

2.1.4 Érintőképernyő

Az érintőképernyő feladata a mobil eszközökben kettős – egyrészt output eszköz, azaz visszajelzést szolgáltat a felhasználónak, másrészt input eszköz és kezeli a felhasználói interakciókat. Különböző megoldások léteznek az érintések érzékelésére (kapacitív, rezisztív felületek, infravörös vagy optikai érzékelés), a dolgozatomnak azonban nem célja ezek bemutatása (részletesebben lásd [14]).

Amiért külön fejezetet érdemel ez az eszköz, az a bonyolultsága, hiszen lényegében egy nagyon összetett és sok funkcióval rendelkező szenzorról van szó. Windows 8 platformon a következő feladatokat látja el:

- Érzékeli, hogy hol történt lenyomás (tipikusan egy X és Y koordináta). A platformon van beépített korrekciós algoritmus is, ami segít meghatározni azt, hogy egy mozgásban lévő, dinamikus felületen a felhasználó „hova szeretett volna” nyomni.
- Érzékeli, hogy milyen erősségű a lenyomás (nem minden eszköz támogatja).
- Érzékeli, hogy mennyi ideig tart a lenyomás – így tud megkülönböztetni olyan gesztusokat, mint a Tap (egyszeri lenyomás-felengedés), Press and hold (lenyomás és hosszú tartás).
- Érzékeli, ha a lenyomás és a felengedés pozíciója nem ugyanaz. Ez segít a gesztusok felismerésében vagy a Drag&Drop műveletek kezelésében.
- Érzékeli a többujjas lenyomást. Ez szintén a gesztusfelismerés szempontjából fontos. A platform (a már fent említett Tap és Press and hold statikus gesztusok mellett) 5 alapvető, ún. manipulációs gesztust támogat (részletesebben lásd [15]):
    - Slide: Egy vagy több ujj megérinti a képernyőt és ugyanabba az irányba mozog.
    - Swipe: Egy vagy több ujj megérinti a képernyőt és ugyanabba az irányba mozog egy rövid szakaszon.

- Turn: Két vagy több ujj megérint a képernyőt és óramutató irányába vagy azzal ellentétesen, ívben mozog.
- Pinch: Két vagy több ujj megérinti a képernyőt és közelebb mozog egymás felé.
- Stretch: Két vagy több ujj megérinti a képernyőt és távolabb mozog egymástól.

A fenti alapvető gesztusok adják minden interaktív rendszer alapját, ezekkel valósíthatóak meg az olyan szokásos funkciók, mint például a nagyítás (tipikusan a pinch és stretch gesztusok megfelelő kezelésével) vagy a forgatás (a turn megfelelő kezelésével).

Az érintőképernyő által nyújtott szolgáltatások ismerete különösen fontos, hiszen ezek azok a szolgáltatások, amelyeket a felhasználók közvetlenül is megpróbálnak igénybe venni, amikor egy alkalmazást használnak – így ezek azok, amik kiváltják a naplózandó eseményeket és a gyűjtött adatok egyik felhasználási lehetősége például éppen az, hogy visszakövetkeztetünk adott interakciókra.

2.1.5 Sensor Fusion

Nem újdonság, hogy a mobil eszközökbe különböző szenzorokat szerelnek, amelyekkel az adott eszköz környezetének vagy helyzetének változását követhetjük. Ez a mobil eszközökön alapvető fontosságú (lásd az 1.3 részben tárgyaltakat).

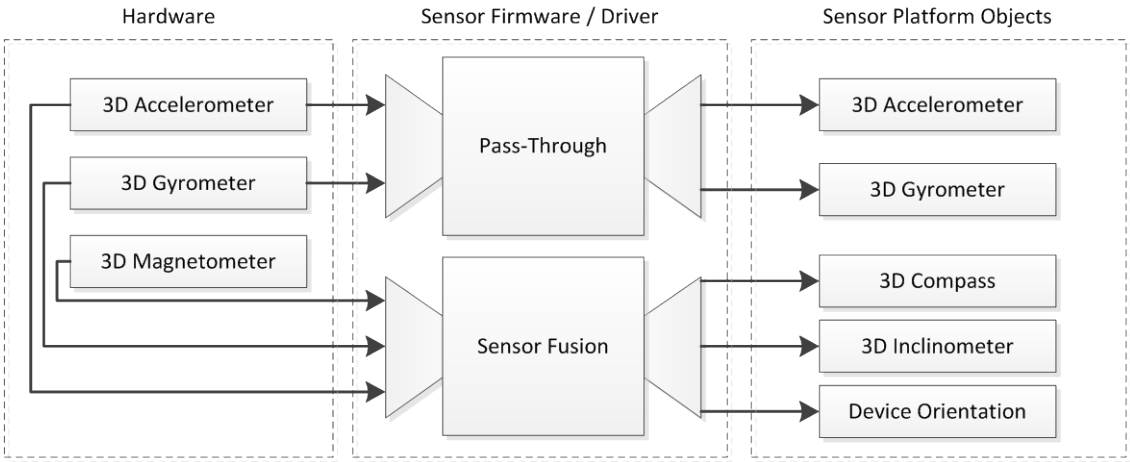
Mint minden hardver-szoftver együttműködési problémában, a szenzorok esetében is kulcsfontosságú egy olyan absztrakciós réteg bevezetése, amely elfedi a fizikai megvalósítás részleteit. Ez az egyik célja a Windows 8 Sensor Fusion platformjának.

Egy másik fontos célja ennek a megoldásnak, hogy a meglévő adatokat fel- vagy átdolgozza, esetleg hozzátegyen valamit a mérésekhez [16]. A következőkben bemutatom az egyszerű szenzorokat, majd azt, hogyan állítja elő a Sensor Fusion platform a végső eredményeket.

A legtöbb eszközben rendelkezésre áll egy **fényszenzor**, ami a külső fény erejét méri lux skálán. Ez a szenzor nem része a Sensor Fusion megoldásnak.

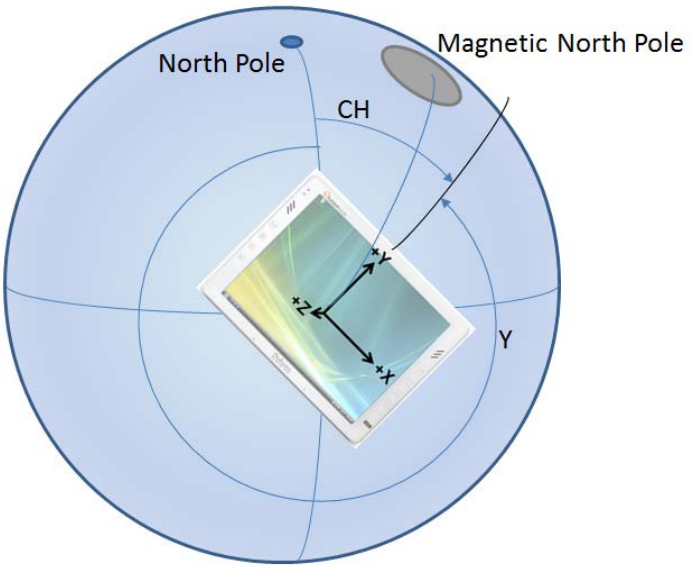
Három további szenzor a fizikai pozíciót vagy fizikai hatásokat méri. Az egyik ilyen szenzor egy 3D **gyorsulásmérő**, ami a tér három irányában méri a gravitációs gyorsulást. A másik szenzor egy 3D **giroszkóp**, ami a háromdimenziós tér három tengelye mentén méri a forgatások sebességét. A harmadik szenzor egy 3D **magnetométer**, ami a mágneses tér erősségét méri a tér három irányában.

Ennek a három szenzornak a mérései sok esetben kevésnek bizonyulnak, a magnetométer esetében pedig még nehezen értelmezhetőek is. Sok esetben egyetlen szenzor önmagában még akár hibás eredményeket is adhat (ilyen előfordulhat például a „gimbal lock” jelenség esetében, amikor a mérést végző 3 mérőfüggeszték közül 2 párhuzamos állásba kényszerül, ezzel pedig az egész szenzor elveszíti az egyik szabadsági fokát). A könnyebb felhasználhatóság, értelmezhetőség vagy éppen az utólagos korrekciók érdekében hozták létre a Sensor Fusion platformot, amit az alábbi ábra szemléltet:



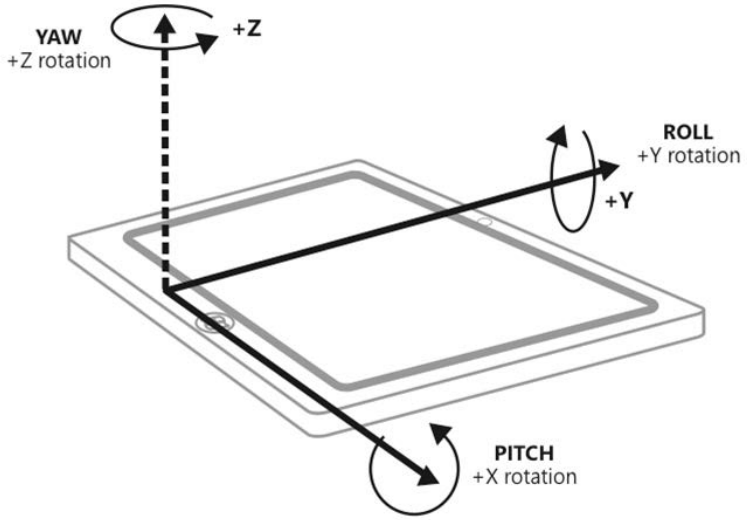
4. ábra - A Sensor Fusion platform [17]

A gyorsulásmérő és a giroszkóp mérései önmagukban is elérhetőek, de a magnetométer adataihoz már nincs hozzáférésünk. A három fizikai szenzorból lehetőségünk van olyan további szenzorokat „szintetizálni”, mint például egy háromdimenziós **iránytű**, amiben van döntés-kompenzáció és megmondja, merre van a valódi és a mágneses észak.



5. ábra - A döntéskompenzált iránytű működése [17]

Egy másik új szenzor az **inklinométer**, ami azt mondja meg, hogy az eszköz milyen mértékben van elfordulva a saját tengelyi körül:



6. ábra - Az inklinométer működése [17]

Egy harmadik ilyen szenzor egy komplett **orientációs szenzor**, amely a többi szenzor méréseiből képes helyesen megmondani, hogy az eszköz hogyan van elhelyezve a térben egy forgatási mátrix, vagy az ennek megfelelő kvaterniós reprezentáció segítségével. Ez például már alkalmas a fentebb említett gimbal lock jelenség megfelelő kezelésére is, hiszen minden fizikai szenzor adatait figyelembe veszi. Van még egy ennél jóval **egyszerűbb orientáció szenzor** is, amely csupán azt tudja megmondani, hogy hány fokkal van elforgatva a képernyő.

A fentiek alapján világos, hogy a megvalósított komplex szenzorrendszer több a részek összességénél; nem csak egy egyszerű absztrakciós réteg a hardver elfedésére és vezérlésére, hanem hozzátesz és értelmez, megkönnyítve ezzel a fejlesztők munkáját.

Az adatok kiolvasásának két módja van minden szenzor esetében: egyrészt lehetőségünk van explicit módon rákérdezni az aktuális értékekre, másrészt pedig feliratkozhatunk egy eseményre, amely akkor váltódik ki, ha a szenzor változást észlel a környezetben.

2.1.6 GeoLocation API

Windows 8 platformon a Geolocation API támogatja a pozícióadatok elérését. Fontos különbség a többi szenzorhoz képest, hogy ha ezt az API-t használni szeretnénk programunkban, akkor ezt az alkalmazás leírásában (manifest) közzé kell tennünk, de a felhasználóknak persze még ekkor is van joguk megtagadni az ilyen hívásokat.

A logikai szenzorból nyerhetünk adatokat a tényleges pozícióval kapcsolatban. Ez az adatokat két helyről kaphatja: a Windows Location Provider komponenstől vagy valódi GPS (Global Positioning System) szenzortól.

A Windows Location Provider WiFi hálózatok, access pointok és IP-cím alapján próbál meg pozícióadatokat szolgálni. Ez a megoldás kétségkívül egyszerűbb és energiatakarékosabb, de természetesen nem működhet mindig.

Ezekre az esetekre használja a logikai szenzorréteg a valódi GPS szenzort. A dolgozatnak nem célja a GPS működésének részletes ismertetése (ezért részletesen lásd [18]). Lényegében arról van szó, hogy a Föld körül keringenek műholdak, ezek jelét tudja venni a szenzor a készülékben, majd a műholdak pozíciói alapján megfelelő algoritmusokkal számolhatóak a megfelelő koordináták és paraméterek.

Attól függően, hogy a készülékben lévő fizikai szenzor mire képes, különböző adatokat nyerhetünk ki: szélesség, hosszúság, magasság, sebesség, irány északhoz képest, illetve a számításokra vonatkozó hibahatárok. Ezek mellett – ha támogatott – akkor akár a polgári címet is visszakaphatjuk (ország, állam, város, irányítószám).

A többi szenzorhoz hasonlóan ilyen adatokat is kétféleképpen nyerhetünk: vagy direkt rákérdezve, vagy a szenzor által biztosított eseményre feliratkozva.

Fontos még, hogy mivel a Windows 8 platformra lehetőség van JavaScript nyelven is programokat fejleszteni, ezért a Windows Geolocation API mellett akár a W3C Geolocation API-t is használhatjuk. Igazi különbség természetesen nincs, mindkettő ugyanazt a fizikai eszközt használja.

3 A keretrendszer működése

Ebben a fejezetben bemutatom a bevezetőben már említett keretrendszer működését, utalva az előző fejezetekben leírt fontosabb technológiai eszközökre. Részleteiben tárgyalom azt, hogy milyen események és adatok kerülnek naplózásra, kitérve magára a naplózási folyamatra is.

3.1 A keretrendszer működésének alapjai

A keretrendszerben a naplózást a GlobalEventLogger osztály valósítja meg. Ezt az osztályt kell példányosítani, méghozzá úgy, hogy az alkalmazás egész életciklusa alatt, minden lap számára elérhető legyen.

Példányosítás után szükség esetén fel lehet iratkozni az életciklus-események naplózására az előre biztosított metódusokkal. Ez opcionális, de hasznos a későbbi mérésekhez. Alapesetben ennyire van mindössze szükség, hogy a modul használható legyen – a naplózáshoz szükséges konfiguráció teljes egészében a konstruktor dolga.

Inicializáláskor a modul először feliratkozik az alkalmazáshoz tartozó Frame navigációs eseményeire. Ezekben az eseményekben a naplózás mellett folyamatosan nyilvántartja az aktuális lap típusát is.

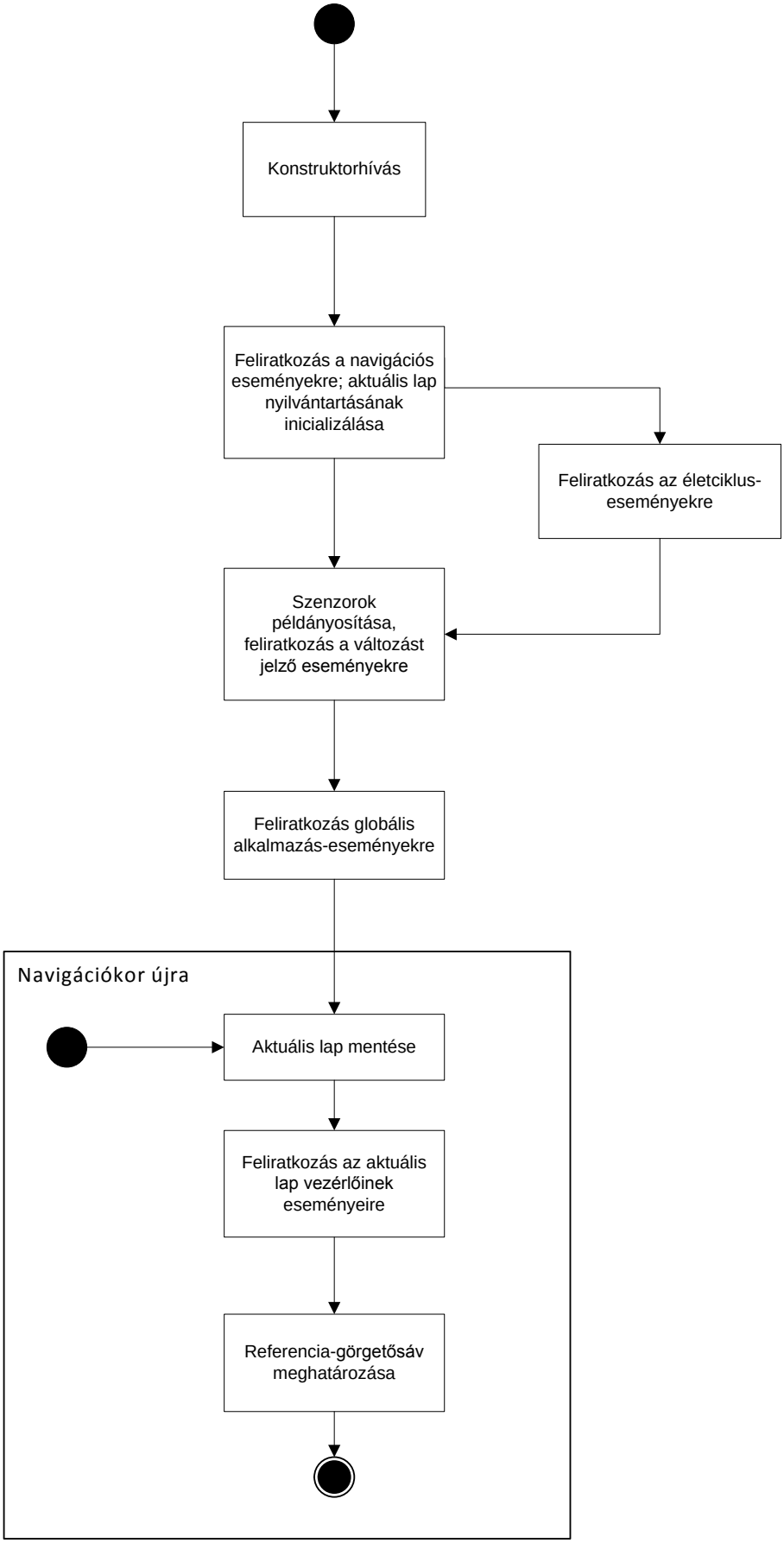
Ezután feliratkozik a rendelkezésre álló szenzorok mérésváltozást jelző eseményeire (lásd a 2.1.5 részben tárgyalt eseményalapú modellt). Ezekben az eseménykezelőkben történik a mérési eredmények naplózása. A szenzorokat reprezentáló objektumok az egész alkalmazásban közösek, így ez a feliratkozási folyamat egyszer történik meg.

Következő lépésben globális alkalmazás-eseményekre iratkozik fel a modul: ilyenek a DPI-változás vagy a kijelzés orientációjának változása (ez lényegében a szoftveres orientáció, azaz a megjelenített kép el van-e forgatva, és ha igen, mennyire). Ennek is csak egyszer kell megtörténnie.

Végül feliratkozik az adott lap összes vezérlőjének eseményeire. (a naplózott események és paramétereik részletes leírása megtalálható a későbbi fejezetekben illetve a B Függelékben). Fontos, hogy ezt minden lapváltáskor meg kell később is tenni, hiszen a következő lap egy újabb objektumpéldány. A feliratkozás a Visual Tree bejárásával kezdődik – ez a felhasználói felületen megjelenő elemeket tartalmazza fa struktúrában a konténer-gyerek hierarchikus kapcsolatoknak megfelelően. Minden

egyes elemnek megvizsgálom az eseményeit és feliratkozok a megfelelő naplózási metódussal a megfelelő eseményre.

Van még egy fontos lépés, amit szintén meg kell tenni minden lapváltáskor. A felhasználónak biztosított felület gyakran „hosszabb”, mint a rendelkezésre álló képernyőméret – ebben az esetben vízszintesen görgethetővé válik a felület. Ha azonban a felhasználó görgeti a felületet, majd rányom egy elemre, akkor ennek az eseménynek a paraméterei között megjelenő koordinátákban nincs benne a kompenzáció az eddigi görgetésre vonatkozóan. Ezért biztosítottam lehetőséget a modul felhasználóinak, hogy megjelöljenek minden oldalon egy ScrollViewer objektumot (ez felelős a görgetősáv kirajzolásáért és működtetéséért), aminek aktuális eltolását mindig figyelembe veszem a koordináták naplózásánál – ez lesz a referencia-görgetősáv. Ez az elem is a Visual Tree egyik eleme, így azt bejárva lokalizálható.



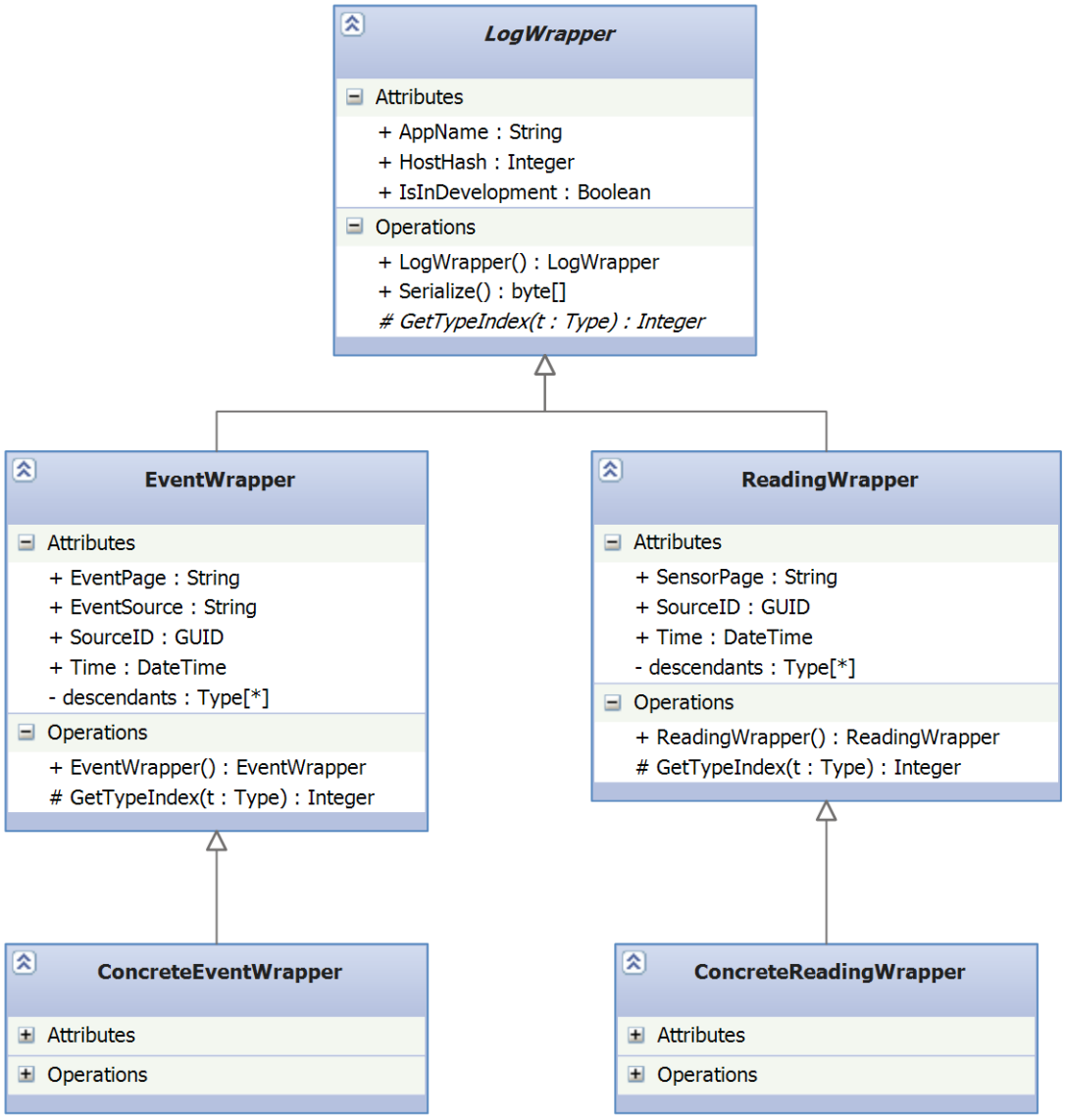
7. ábra - A keretrendszer inicializálásának folyamata

Inicializálás után a keretrendszer készen áll a megfelelő adatok naplózása. Ezek az adatok állnak részben eseményekből (amelyek a felhasználói interakciók részeit adják) és a hozzájuk tartozó paraméterekből (ezek egy része általános, mint például az aktuális oldal vagy az időpont, mások pedig esemény-specifikusak), másrészt pedig szenzoradatokból (amelyek lényegében a szenzorok méréseit rögzítik). Az adatgyűjtés módját a következő fejezetek, a gyűjtött adatok listáját pedig a B Függelék tartalmazza.

Az adatokat egy-egy állományba gyűjti a keretrendszer binárisan megformázva a .NET keretrendszer BinaryWriter osztályának segítségével. Ezeknek az állományoknak a tartalmát időnként visszaolvasom és egy-egy WCF szolgáltatáshívásnak egyszerűen átadom a bináris adatfolyamot. Ezután a szerveren történik meg az adatfolyam értelmezése és adatbázisba írása. Természetesen mind a fájlba írás, mind a beolvasás és a szolgáltatáshívások is aszinkron módon futnak a platform szellemiségének megfelelően (lásd a 2.1 fejezetben tárgyaltakat).

3.2 Az adatgyűjtés megvalósítása

Az adatok gyűjtéséhez megvalósított adatmodell a következőképpen néz ki:



8. ábra - A keretrendszer adatmodellje

A közös ős lehetővé teszi, hogy a logika egy részét (a sorosítást, a típus azonosítószámának lekérdezését, különböző azonosítók az adatok csoportosításához) egy helyen tartsunk. Az első szintű leszármazottak az események és a szenzoradatok naplózásához szükséges adatosztályok. Ezek egyrészt tartalmazzák az összes leszármazott típusinformációját, másrészt további közös adatokat (időbélyeg, aktuális oldal, események esetén a vezérlőneve). A levélszintű leszármazottak az egyes eseményeket és szenzorméréseket reprezentáló adatosztályok. A részletes megvalósítást a következő fejezetek tartalmazzák.

3.2.1 Szenzoradatok gyűjtése

A szenzoradatok gyűjtése a 2.1.5 fejezetben bemutatott eseményalapú modellre épül – változáskor kiváltódik egy esemény, erre iratkozik fel a keretrendszer egy metódussal, ami elvégzi a naplózást. A feliratkozás kódrészlete:

```
Accelerometer ameter = Accelerometer.GetDefault();
if (ameter != null)
{
  ameter.ReadingChanged += ameter_ReadingChanged;
  ameter.Shaken += ameter_Shaken;
}
```

Minden egyes szenzorméréshez tartozik egy burkoló osztály, ennek tartalma kerül feltöltésre a mérési adatok alapján, majd ez kerül kiírásra a LogData metódusban.

Az adatok akkor kerülnek csak naplózásra, ha az előző méréshez képesti változás meghalad egy bizonyos értéket, ezeket küszöbváltozókban („treshold”) tárolom.

```
private void ameter_ReadingChanged(Accelerometer sender,
AccelerometerReadingChangedEventArgs args)
{
 if (lastacceleroreading == null ||
                  Math.Abs(lastacceleroreading.AccelerationX –
                  args.Reading.AccelerationX) > accelerotreshold ||
                  Math.Abs(lastacceleroreading.AccelerationY –
                  args.Reading.AccelerationY) > accelerotreshold ||
                  Math.Abs(lastacceleroreading.AccelerationZ –
                  args.Reading.AccelerationZ) > accelerotreshold)
 {
  AcceleroMeterReading_Data rw = new AcceleroMeterReading_Data()
  {
   SensorPage = currentpagetype == null ? "" : currentpagetype.Name,
   Time = args.Reading.Timestamp.DateTime,
   SourceID = this.SourceID,
   X = args.Reading.AccelerationX,
   Y = args.Reading.AccelerationY,
   Z = args.Reading.AccelerationZ
  };
  LogData(rw);
 }
 lastacceleroreading = args.Reading;
}
```

A kiíráshoz egyetlen fájlt használok az összes szenzoradatnak, ezért megfelelő kölcsönös kizárást kell megvalósítani:

```
private void LogData(LogWrapper lw)
{
  string storagefilename =
          lw is EventWrapper ? EventLogFileName : SensorLogFileName;
  if (lw is EventWrapper)
  {
    eventmutex.WaitOne();
  }
  else
```

```
  {
    sensormutex.WaitOne();
  }
  StorageFile log = ApplicationData.Current.LocalFolder
                  .CreateFileAsync(storagefilename,
                                   CreationCollisionOption.OpenIfExists)
                  .AsTask().Result;
  var writeStream = log.OpenAsync(FileAccessMode.ReadWrite).AsTask()
                   .Result;
  using (Stream outStream = Task.Run(() => writeStream
                                          .AsStreamForWrite()).Result)
  {
    outStream.Seek(outStream.Length, SeekOrigin.Begin);
    using (BinaryWriter sw = new BinaryWriter(outStream))
    {
      byte[] bytes = lw.Serialize();
      sw.Write(bytes);
    }
  }
  if (lw is EventWrapper)
  {
    eventmutex.ReleaseMutex();
  }
  else
  {
    sensormutex.ReleaseMutex();
  }
}
```

Maga a sorosítás egy nagyon hatékony bináris formátumot használ, amit minden csomagolóosztályra egységesen implementáltam. Erre mindenképpen szükség volt, mert a Windows Runtime API kész sorosítási megoldásai (DataContractSerializer) sajnos nem elég tömörek, ez pedig mobil eszközökön különösen nagy hátrányt jelent. Egyrészt az adatokat tárolni kell, ami önmagában probléma, másrészt a nagy mennyiségű adat átküldése a szerverre jóval tovább tarthat. A méréseim alapján a DataContractSerializer még egy egyszerű lenyomáshoz tartozó csomagolóosztály egy példányából is közel tízszer annyi adatot generál, mint a saját megoldásom, ez a különbség pedig a komplexebb szenzoradatok és események esetében az alkalmazott megoldások miatt tovább nőne (lásd részletesebben [19]). Látható tehát, hogy szükséges volt saját megoldást kidolgozni. A sorosítás részletes megvalósítása megtalálható a C Függelék: Sorosítás kódrészletében.

3.2.2 Események naplózása

Az eseményekre történő dinamikus feliratkozáskor elsőként bejárásra kerül a teljes Visual Tree. Mivel fa-struktúráról van szó, egy rekurzív algoritmus könnyen be tudja járni a fát. Mindig megvizsgálom a fa adott elemének eseményeit és a megfelelő nevű eseményre feliratkozok a következő módon (lásd a technológiai részleteket a 2. fejezetben tárgyaltak között):

```
private void SubscribeToEvents(DependencyObject d)
{
  DependencyObject current = d;
  foreach (var ev in current.GetType().GetRuntimeEvents())
  {
    if (ev.Name == "Click")
    {
      WindowsRuntimeMarshal.AddEventHandler<RoutedEventHandler>
                    (
                       del => (EventRegistrationToken)ev.AddMethod
                            .Invoke(current, new object[] { del }),
                       token => ev.RemoveMethod
                            .Invoke(current, new object[] { token }),
                       new RoutedEventHandler(LogMouseClick)
                     );
    }
    if (ev.Name == "Tapped")
    {
      WindowsRuntimeMarshal.AddEventHandler<TappedEventHandler>
                   (
                      del => (EventRegistrationToken)ev.AddMethod
                            .Invoke(current, new object[] {del}),
                      token => ev.RemoveMethod
                            .Invoke(current, new object[] { token }),
                      new TappedEventHandler(LogTapped)
                    );
    }
  ///..
  int childrencount = VisualTreeHelper.GetChildrenCount(d);
  for (int i = 0; i < childrencount; i++)
  {
    current = VisualTreeHelper.GetChild(d, i);
    SubscribeToEvents(current);
  }
}
```

Az adatok fájlba írása hasonló módon történik a szenzoradatok kiírásához (lásd az előző fejezetben).

Az események buborékozásának kezelése alapos átgondolást igényelt. A 2.1.3 részben bemutatottaknak megfelelően ez azt jelenti, hogy az események a kiváltódás helyéről felfelé terjednek a Visual Tree hierarchiájának megfelelően és a felsőbb szintű vezérlőkben is kiváltódnak, emiatt pedig alapesetben többször is naplózásra kerülne ugyanaz az esemény. A keretrendszer elsődleges célja viszont az, hogy a felhasználók viselkedésének vizsgálatát segítsége – a legtöbb felhasználó pedig nem rendelkezik olyan programozási ismeretekkel, hogy az alkalmazás használata közben tudatosan figyeljen erre a jelenségre. Végül tehát úgy döntöttem, hogy minden eseményt csak egyszer, a kiváltódás helyén naplózok.

3.2.3 Referencia-görgetősáv kijelölése

A referencia-görgetősáv kijelölése a kódban a megfelelő ScrollViewer objektum Tag tulajdonságában történő True logikai érték elhelyezésével lehetséges:

```
<ScrollViewer.Tag>
  <x:Boolean>True</x:Boolean>
</ScrollViewer.Tag>
```

Ezt később a vizuális fa bejárásával megkeresem:

```
private void InnerGetCurrentScroller(DependencyObject d)
{
 if (currentscroller == null && d.GetType() == typeof(ScrollViewer))
 {
   ScrollViewer sw = (ScrollViewer)d;
   if (sw.Tag != null && (bool)sw.Tag)
   {
     currentscroller = sw;
     return;
   }
 }
 int childrencount = VisualTreeHelper.GetChildrenCount(d);
 for (int i = 0; i < childrencount; i++)
 {
   InnerGetCurrentScroller(VisualTreeHelper.GetChild(d, i));
 }
}
```

3.2.4 Adatok küldése a szerverre

Az adatok az alkalmazás indulása után nem sokkal, ezután pedig 15 percenként kerülnek felküldésre a szerverre. Ehhez először kiolvasom a fájl tartalmát (a kölcsönös kizárása figyelve), majd egyszerűen meghívom a megfelelő szolgáltatásoperációt.

A szerveren először a beérkező bájtfolyamot értelmezem a saját sorosítási protokollomnak megfelelően (lásd a C Függelékben a kliensoldali megvalósítást). Az így előállított objektumok kerülnek beírásra a naplózási adatbázisba. Ennek az adatbázisnak a struktúrája egyszerű, elsősorban fejlesztői célokat szolgál és igyekszik megkönnyíteni az adatértelmezést: mindössze egyetlen táblából áll és ebben minden egyes burkolóosztály minden tulajdonságának megfelel egy-egy oszlop.

4 A gyűjtött adatok elemzése és értelmezése

A gyűjtött adatokat különböző módszerekkel elemzem a következő fejezetekben. A levont következtetés esetenként egyértelmű, más esetekben viszont a következtetés valójában egy kérdés felvetése. Ezeknek a kérdéseknek a megválaszolása alkalmazásonként eltérő (pl. mást jelent a kevés navigáció egy hírújság és egy webáruház esetén), így a kérdésekre magukra nem adom meg a választ, csupán leírom, hogyan lehet felismerni a problémát.

A keretrendszert az egyik legnagyobb magyar aukciós portálnak fejlesztett Windows 8 kliensalkalmazásba építettem be. Az alábbi következtetéseket az ennek az alkalmazásnak a tesztelési szakasza alatt gyűjtött, közel 40 ezer adatsor elemzésével vontam le (az adatok megtalálhatóak a dolgozat lemezmellékletén).

4.1 Az alkalmazás látogatottsági adatai

Ezeknek a mérőszámoknak a számítása és értelmezése meglehetősen evidens, mégis nagyon hasznos információkkal szolgálhat az alkalmazáshasználattal kapcsolatban.

Átlagos munkamenet hossz: A munkamenet fogalmára saját definíciót vezettem be. Mivel a „bezárás” eseményt a Windows Store alkalmazások esetében nem értelmezhetjük (lásd a 2.1.1 fejezetben leírtakat) ezért úgy definiáltam a munkamenetet, hogy azok az egy felhasználótól érkező események, amelyek között kevesebb, mint 5 perc telik el.

Ez alapján az adatbázisban található adatokat 133 munkamenetbe osztottam. Az így kiszámított **átlagos session hossz 5 perc 39 másodperc.** Ezzel kapcsolatban a következőt érdemes megvizsgálnunk:

Kérdés: Elég-e az, hogy a felhasználók 5 percet és 39 másodpercet töltenek el az alkalmazásunkban?

A válasz természetesen alkalmazásfüggő – egy játékalkalmazásban például kevés lehet, míg egy webáruházban lehet akár pont elég. A Google Analytics egyik, 2011-ből származó jelentése alapján a felhasználók átlagosan 5 percet és 8 másodpercet töltenek el egy webhelyen [20]. Ezek alapján elégedettek lehetünk a jelenleg elért eredménnyel, lévén hogy egy webáruházról van szó.

Átlagosan meglátogatott oldalak egy munkamenet alatt: A munkamenet alatt a felhasználók átlagosan körülbelül 9,5 lapot látogatnak meg az alkalmazásunkban

(minden egyes navigáció egy új lapot nyit meg; a főoldalra történő első belépés is egy navigáció). A kérdés megint adja magát:

Kérdés: Elég-e az, hogy a felhasználók 9,5 lapot látogatnak meg egy session alatt?

A válasz természetesen itt is alkalmazásonként eltérő lehet. Egy 2004-es mérés szerint a felhasználók általában 4,6 lapot látogatnak meg egy alkalommal egy webhelyen [21]. Ezzel összehasonlítva a mért érték jóval több webáruházunk esetében – ettől az alkalmazásunk dinamikusabb lesz, több információt tudunk közölni a több különböző lapon.

Egyes oldalak látogatottsági mutatói: Az alkalmazásunk 11 lapból áll, ezeknek a látogatottsági adatai a következők:

Oldal	Látogatások (db)	Látogatások az összes látogatás %-ában
BuyPage	213	8,00
ExceptionPage	82	3,08
GalleryPage	6	0,23
InterestedProductsPage	40	1,50
MainPage	1303	48,97
NoServicePage	165	6,20
ProductPage	172	6,46
ProfilePage	40	1,50
SearchProductsPage	554	20,82
UserPage	6	0,23
WatchedProductsPage	80	3,01
Összesen	**2661**	**100,00**

1. táblázat - Látogatások eloszlása laponként

Az adatokat megvizsgálva a következő kérdések merülhetnek fel:

Kérdések: A főoldal látogatottsága kiemelkedően nagy. Valószínűleg nagyon sokan csak a főoldalt nézik meg, aztán elhagyják az alkalmazást – ez nem kívánatos. Mit tehetünk ez ellen?

A galériánkat nagyon kevesen látogatják. Miért? El van rejtve nagyon, vagy nincs rá szükség?

Bizonyos alkalmazásszolgáltatásokat (ezeket a WatchedProductsPage és az InterestedProductsPage valósítják meg) meglehetősen keveset használnak. Fölöslegesek, vagy nem megfelelően kidolgozottak? Esetleg nem megfelelően promotáltak?

A látogatóinknak mindössze 8% része dönt úgy, hogy vásárol is valamit. Ez kielégítő?

A fenti kérdések mellett talán egyebekre is rámutathatnak az adatok, ezek a legfontosabbak. Általában ezek megválaszolása nagyon alkalmazásfüggő, mindig igaz azonban, hogy érdemes megvizsgálni a látogatottsági adatokat és átgondolni, hogy egyrészt a tartalom megfelelően van-e elrendezve (fontos tartalmak a nagy látogatottságú lapokon és fordítva), másrészt valóban szükség van-e a keveset használt lapokra.

A vásárlással kapcsolatban fontos megjegyezni, hogy önmagában az, hogy a felhasználó eljut az oldalra, ahol a vásárlás adatait meg kell adni, még nem jelenti azt, hogy biztosan vásárolt is. Mindenesetre elég valószínű, így jól használható mérőszáma a vásárlásoknak (itt természetesen tesztvásárlásokról van szó, hiszen az alkalmazásnak a tesztelési ciklusából származnak az adatok). Egy 2007-es tanulmány szerint az átlagos konverziós ráta (azaz a valóban vásárló és a csupán látogató felhasználók aránya) mindössze 2,9% [22]. Ha az alkalmazás éles bevezetése után is így teljesít, hatalmas profitot fog termelni…

Egy oldalon eltöltött átlagos és összes idő: A látogatások száma mellett azoknak a hossza (összes, átlagos) is értékes adatokat szolgáltathat az egyes lapokon:

Oldal	Átlagos idő	Összes idő	Összes idő teljes idő %-ában
BuyPage	0:01:03	1:28:58	13,49
ExceptionPage	0:01:40	0:35:06	5,32
GalleryPage	0:00:08	0:00:15	0,04
InterestedProductsPage	0:00:44	0:14:33	2,21
MainPage	0:00:37	5:58:17	54,31
NoServicePage	0:00:49	0:26:44	4,05
ProductPage	0:00:10	0:13:49	2,09
ProfilePage	0:01:03	0:17:57	2,72
SearchProductsPage	0:00:22	1:33:34	14,18
UserPage	0:00:19	0:00:57	0,14
WatchedProductsPage	0:00:14	0:09:34	1,45
	0:00:35	**10:59:44**	**100,00**

2. táblázat - Időeloszlások oldalanként

Az itteni mérések összhangban vannak az előző mérés eredményeivel (lásd 1. táblázat). A felvetendő kérdések is hasonlóak:

Kérdés: Miért töltenek el a felhasználók X időt az Y oldalon? Ez sok, kevés vagy megfelelő? Ennek megfelelően rendeztük el a tartalmat?

4.2 Interakció- és eseményelemzések

A következőkben az interakciók és események száma alapján mutatok be néhány hasznos következtetést. Ezek a következtetések kezdenek rámutatni igazán a

keretrendszer egyediségére – bár az általam később megvizsgált piaci megoldások is képesek hasonlóra, jóval több előkészületet igényelnek és még akkor is csak korlátozott adatgyűjtési lehetőséget biztosítanak ilyen téren.

Magában az alkalmazásban sok esemény nem került felhasználásra (ilyenek pl. a Manipulation események), így ezek természetesen nem szerepelnek ezekben a kimutatásokban. Továbbá az áttekinthetőség érdekében, azokban az esetekben, ahol munkamenetenkénti elemzést végzek, csupán néhány munkamenetre mutatom be az eredményt.

Átlagos eseményszám egy munkamenetben: Az összes munkamenetre és eseményre vonatkoztatva megállapítottam, hogy **átlagosan 244** esemény van egy sessionben. Ez önmagában nem szolgál sok információval, de a későbbiekben fontos referenciaérték lesz. Mindemellett ha több alkalmazásra is megvizsgáltuk volna ezt az átlagértéket, akkor a következőt mindenképpen érdemes lenne átgondolni:

Kérdés: Az átlagos eseményszám hogyan viszonyul a saját alkalmazásomban és a piacon lévő többi alkalmazásban?

Különböző események százalékos eloszlása: Érdemes megvizsgálni a használt interakciók gyakoriságát összesen illetve egymáshoz képest:

Esemény	Összesen (db)	Eloszlás (%)
Click	525	1,78
DoubleTapped	12	0,04
Holding	6	0,02
KeyDown	2668	9,03
PointerCanceled	2	0,01
PointerCaptureLost	1470	4,97
PointerEntered	5134	17,37
PointerExited	4016	13,59
PointerMoved	11855	40,11
PointerPressed	888	3,00
PointerReleased	444	1,50
PointerWheelChanged	57	0,19
RightTapped	117	0,40
Tapped	2359	7,98
Összesen	**29553**	**100,00**

3. táblázat - Események eloszlása

A 3. táblázat adataiból látható, hogy az összes eseménynek több mint 40% része csupán a mutató (az egérkurzor érintőképernyős megfelelője) mozgatásából áll (PointerMoved), további 42% részét pedig hasonlóan mutatóesemények (egyéb Pointer események) adják (ezek akkor váltódnak ki, ha a mutató például egy vezérlő fölé kerül

vagy elhagyja azt), a maradék 18% további eseményeket tartalmaz. Ezzel kapcsolatban rögtön felmerülnek a következő kérdések:

Kérdések: A felhasználóinknak meglehetősen sokat kell mozgatniuk a mutatót a különböző tartalmak között. Miért? Rosszul rendeztük el a tartalmat? Vagy ez megfelelőnek számít?

Sajnos ahhoz, hogy ezt a kérdést megválaszoljuk, szükség lenne más alkalmazásokból származó adatokra is, de egyelőre maga a platform nagyon fiatal még, főleg fejlesztők és érdeklődőbb felhasználók dolgoznak csak vele. Viszont a fenti adatokból a feltett kérdést egyértelműen meg lehet válaszolni. Valószínűleg ez az érték egyébként nem olyan sok – a mutatóesemények közül a mozgatás csak akkor naplózódik a keretrendszerben, ha az előzőhöz képest 50 pixellel arrébb és legalább 0,1 másodperccel később történt. Ez azt jelenti, hogy egy 1366 pixel széles képernyő egyik oldaláról a másikra elérni több mint 27 ilyen bejegyzés lassan mozgatva a mutatót.

A mutató mozgatása és a görgetés (PointerWheelChanged) fontos események, mert ez rengeteg interakciónak szerves része. A mozgatáshoz tartozó PointerMoved esemény például a gesztusrekonstrukció egyik fontos része lesz; erre a 4.3 fejezetben bemutatok majd egy hatékony eszközt. A többi mutatóesemény azonban kevésbé fontos, így egyelőre érdemes egy ezek nélküli korrigált adatsort is vizsgálni:

Esemény	Összesen (db)	Eloszlás (%)
Click	525	2,98
DoubleTapped	12	0,07
Holding	6	0,03
KeyDown	2668	15,16
PointerMoved	11855	67,36
PointerWheelChanged	57	0,32
RightTapped	117	0,66
Tapped	2359	13,40
Összesen	**17599**	**100,00**

4. táblázat - Releváns események eloszlása

A 4. táblázat adatai sokkal határozottabban mutatják a mutatómozgatás dominanciáját – és könnyebben válaszolhatunk esetleg a szükségesség kérdésére is.

További fontos kérdéseket válaszolhatunk meg ugyanennek a táblázatnak a részletesebb elemzésével:

Kérdés: A Holding eseményt sehol nem implementáltuk, mégis van néhány ilyen. Ennek mi lehet az oka?

A Holding esemény akkor váltódik ki, ha a felhasználó tartósan nyomva tartja az ujját egy ponton. A kérdésre a válasz a jelenlegi helyzetben valószínűleg az, hogy egyszerűen csak „leragadt” a felhasználók keze. Valamivel több ilyen esemény esetén azonban érdemes lehet átgondolni a megjelenésük okát. A Microsoft javaslata szerint, ha a felhasználó egy vezérlőn a Holding interakciót hajtja végre, akkor erre válaszul valamilyen formában a vezérlő szerepéről, az általa ellátott feladatról kell neki visszajelzést adnunk (lásd [15]).

Másik oka lehet az ilyen események megjelenésének például egy olyan elem az alkalmazásban, amitől a felhasználó interakciót remél, de mégsem kap, ezért még tovább tartja lenyomva a vezérlőt, így remélve valami választ. A keretrendszer naplózza az eseményt kiváltó vezérlő nevét – ha a programozó ellátta ilyennel – így akár ilyen hibákra is fény derülhet.

A Windows Store alkalmazásoknak központi kérdése a lapozhatóság. A Microsoft javaslata szerint a tartalom – az eddig általánosan bevett módszerrel ellentétben – vízszintesen görgethető – csak és kizárólag. További javaslat, hogy ne legyen több vízszintes görgetősáv sem egy lapon. Ezek tudatában érdemes kicsit több figyelmet szentelni a PointerWheelChanged eseményekre. A következő fontos kérdés merülhet fel egy lapon a tartalommal kapcsolatban:

Kérdés: A felhasználók X lapon vajon meddig görgetnek el?

Ez egy fontos kérdés, hiszen a tartalmat ennek ismeretében érdemes elhelyezni. Ha úgysem görget el senki a végére, akkor oda érdemes a kevésbé fontos tartalmat tenni – vagy még jobb, ha úgy tervezzük meg a felületet, hogy ne kelljen görgetni. Természetesen ez is alkalmazásfüggő, azonban az adatokból megválaszolhatjuk figyelembe véve saját igényeinket:

EventPage	EventSource	MouseWheel
MainPage	RootFullGrid	-120
MainPage	RootFullGrid	-120
MainPage	RootFullGrid	120
MainPage	RootFullGrid	-120
MainPage	RootFullGrid	-360
MainPage	RootFullGrid	-120

5. táblázat - Görgetések egy munkamenetben

Az 5. táblázat egy munkamenet összes PointerWheelChanged adatát mutatja. Ebből rögtön két dolog látszik: egyrészt, hogy ez a felhasználó csak a MainPage lapon használta a görgetést a RootFullGrid vezérlőn. Másrészt a görgetés nagysága látszik a MouseWheel értékből. Ennek értéke vázlatosan, azt mutatja, hogy hány „egységet”

ugrott a képernyő. Egy egységnyi ugrás például a szövegszerkesztőkben tipikusan egy sort jelent, a MouseWheel értékében pedig egy +/- 120-as változást (részletesebben lásd [23]). A ScrollViewer vezérlő nagyjából a képernyő 1/5 részét lapozza egy egységre, tehát a fenti adatsor azt jelenti, hogy az eredetihez képest még közel 120%-kal (két egység egy irányba, egy másik vissza, újra egy, majd három és megint egy egység az eredeti irányba), azaz több mint egy teljes lappal arrébb ment a felhasználó. Ezt összehasonlítva a lapmérettel (ezt az alkalmazás fejlesztőjeként ismerjük) megállapíthatjuk, hogy a felhasználó elgörgetett a lap végére, azaz látta az összes tartalmat, tehát sikerélményünk lehet.

Persze azzal, hogy a többi lapon ez a munkamenet miért nem görgetett, még nem foglalkoztunk. Ennek tárgyalását most kihagyom, de értelemszerűen fel kell vetni a megfelelő kérdéseket itt is:

Kérdés: X lapon egyáltalán használják-e a görgetés lehetőségét? Milyen gyakran?

Tovább vizsgálódva feltűnhet, hogy van néhány RightTapped esemény is (ennek gesztusára a név egyáltalán nem utal – lényegében egy rövid lenyomás-lehúzás pár). Ezekkel kapcsolatban hasonló kérdés vethető fel, mint ami a Holding esemény kapcsán is:

Kérdés: A RightTapped interakciót sok olyan helyen is használják a felhasználók, ahol nem implementáltam. Ennek mi az oka?

Bár ezek száma sem túl nagy, mégis jogos a kérdés. Itt már kevésbé valószínű, hogy a felhasználóknak csak megbotlott a keze. A RightTapped a Microsoft javaslatai szerint a kijelölés gesztusa. Könnyen lehet, hogy a felületen az adott elem azt sugallta a felhasználóknak, hogy ki lehet jelölni egy elemet további, egyedi interakció céljából.

A KeyDown eseményekre is érdemes ránézni – ezek értelemszerűen a billentyűlenyomások bejegyzései. Természetesen biztonsági okokból maguk a lenyomott karakterek nincsenek naplózva – csupán az kerül lejegyzésre (a szokásos eseményadatok mellett), hogy a lenyomott gomb alfanumerikus vagy nem. A mérések szerint az összes KeyDown esemény (2668 darab) kevesebb, mint a fele (1301 db) volt csak „valódi” karakter. Jogosan merül fel tehát a következő kérdés:

Kérdés: Miért (próbálnak) használni a felhasználóink nagyszámú nem alfanumerikus karaktert?

Az adatokból kiderül, hogy a legtöbb ilyen karaktert a telefonszámok megadására szolgáló szövegdoboz váltja ki – könnyen lehet tehát, hogy ezek a

karaktereke a „/”, a „-” vagy a „+”. Érdemes lehet megfontolni ezek után olyan beviteli mezők biztosítását, ahol ezek eleve rögzítettek.

A Tapped esemény (egyszerű lenyomás) kétségkívül az egyik legjobb forrása felhasználói interakció elemzésnek. Erre később bemutatok egy nagyon hatékony eszközt (lásd a 4.3.1 fejezetben), így most itt csupán egy kérdésre térek ki:

Kérdés: A felhasználóim mekkora hányada használ egeret és mekkora része érintőképernyőt?

Ez egy nagyon fontos kérdés. Az 1.1 fejezetben tárgyaltaknak megfelelően nagyon sokban eltér a két interakciós modell. Érdemes meggondolni, hogy melyikre tervezünk jobban vagy mennyire próbáljuk meg az arany középút elvét követni. Ehhez persze ismerni kell a célközönség szokásait:

Érintőképernyő (db)	Egér (db)	Toll (db)
5132	19953	0

6. táblázat - Beviteli eszközök gyakorisága

A 6. táblázatból jól láthatóan egyelőre az egeret részesítik előnyben a felhasználók. Ez várható volt – egyrészt a tesztelési időszak alatt főleg PC-ken vizsgálták az alkalmazást, másrészt a dolgozat írásakor csak olyan Windows 8-as tabletek voltak elérhetőek jórészt, amelyeket kifejezetten fejlesztőknek szántak. Érdemes megfigyelni azt is, hogy maga az operációs rendszer egyébként akár tollat is tud kezelni – ez pedig egy harmadik interakciós modellt jelent…

4.3 Vizualizációs adatelemzés és következtetések

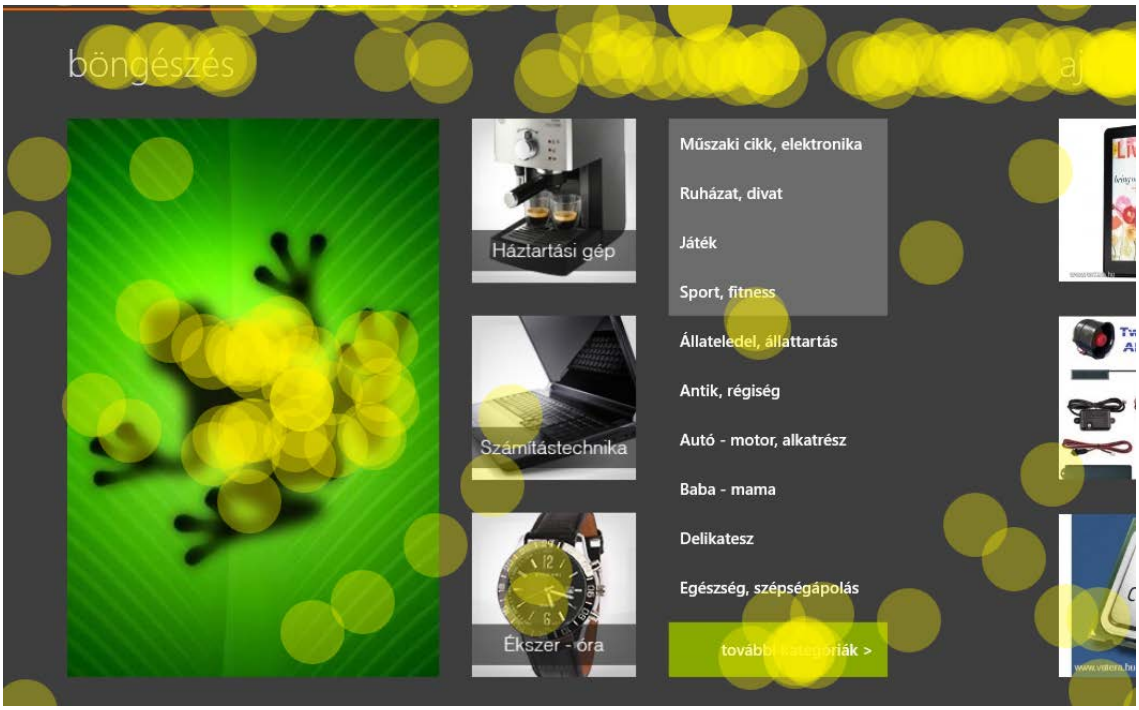
A következőkben olyan eszközöket mutatok be, amivel hatékonyan elemezhetőek már ránézésre is bizonyos felhasználói interakciók. Ezeknek a módszereknek nagy előnye az, hogy mivel vizualizálják az interakciókat, így további vizsgálat nélkül, egyszerűen tudjuk értékelni alkalmazásunk használhatóságát ilyen szempontból.

4.3.1 A hőtérkép

Az egyik ilyen eszköz a hőtérkép. Itt arról van szó, hogy a felhasználó által generált Tapped események koordinátáit rávetítjük az alkalmazás megfelelő felületére. Amire fontos figyelni, hogy a felhasználó ujjai nem egy pixel méretűek, így a koordináták köré is érdemes egy legalább akkora sugarú kört rajzolni, mint egy nagyobb ujjbegy. A körök lehetnek egymáson, átlapolódhatnak, így beszínezve az adott felületet. Minél sötétebb adott pont a felületen, annál aktívabb.

A referencia-görgetősáv bevezetésének köszönhetően (lásd 3.2.3 fejezetben leírtak) helyesen tudjuk azt is kezelni, ha görgetés után nyom le valamit a felhasználó a képernyőn.

Íme az alkalmazás főképernyőjének egy részlete a rávetített lenyomás eseményekkel:



9. ábra - Hőtérkép

A következő nagyon fontos kérdéseket tehetjük fel magunknak az ilyen hőtérképek alapján.

Kérdések: A hőtérképen látom, hogy az X nevű vezérlőre egyáltalán nem kattintanak. Miért van ez? Nem sugallja a vezérlő elég erősen, hogy vele interakcióba lehet lépni?

A hőtérképről leolvasható, hogy a képernyő jobb szélén lévő szövegre gyakran rányomnak, pedig ez nem interaktív vezérlő. Miért gondolják, hogy valami történni fog? Érdemes lenne mégis valami tartalmat helyezni a vezérlő mögé? Vagy esetleg válasszak neki olyan megjelenési stílust, ami jobban sugározza a statikusságot? Távolítsam el?

A hőtérkép bal/jobb oldala (felső/alsó része) sokkal aktívabb a másiknál – lehet, hogy érdemesebb lenne a fontos tartalmakat áthelyezni ide?

A fenti kérdések megválaszolása minden esetben körültekintést igényel, erősen alkalmazásfüggő, gyakran szükséges szakember (ergonómus, dizájner) bevonása is, így ezek tárgyalását nem vállalhatom.

4.3.2 Gesztusrekonstrukció

A 2.1.4 részben bemutatott gesztusokhoz tartoznak megfelelő események – ezek azonban csak akkor váltódnak ki adott gesztus esetében, ha a programozó előre jelezte a kódban, hogy ezeket implementálni fogja. Ilyenkor tipikusan implementálja is és az elemzés is egyszerűbb, elég a Manipulation eseményeket vizsgálni. Ha azonban a programozó ezeket nem akarja implementálni, tipikusan nem is „ad engedélyt” az események kiváltására – így az egyetlen mód a gesztusok rekonstrukciójára az, ha alaposan megvizsgáljuk a mutató mozgását és megpróbálunk benne mintákat felfedezni.

A következő képen például látható, hogyan sikerült Swipe gesztusokat rekonstruálni egy munkamenetben az alkalmazás főoldalán. A nyilak a mozdulat végét és irányát jelzik, a pötty a kezdőpont, a felhasználó pedig nagyjából a vonal mentén húzta az ujját:

10. ábra - Gesztusrekonstrukció a főoldalon

Egy másik rekonstrukció a webáruház termékeket bemutató oldaláról:

11. ábra - Gesztusrekonstrukció a termékoldalon

Ehhez hasonlóan lehetőség van a többi gesztus rekonstruálására is.

A kérdések nagyon sokfélék lehetnek, ám a lényegük mindig ugyanaz:

Kérdés: X felületen az Y gesztust nem implementáltam, a felhasználó azonban mégis használni szeretné. Miért van ez?

A megválaszolása ennek is gondos áttekintést igényel. Mindenképpen figyelembe kell venni a Microsoft ajánlásait a gesztusokkal kapcsolatban és érdemes valamilyen UX-szakemberrel konzultálni.

A webáruházunk esetén például sok vízszintes Swipe gesztust sikerült rekonstruálni az alkalmazás felső részén, ami lényegében a címsor. Ezen a részen a Swipe gesztus nem vált ki görgetést, pedig a felhasználóknak feltehetően ez volt a célja vele. Érdemes lehet átgondolni ennek a megváltoztatását.

4.4 Szenzoradatok elemzése

A beépített Sensor Fusion platform adatainak elemzése is értékes információval szolgálhat. Ezeknek az adatoknak az automatizált gyűjtése különösen hasznos funkciója a keretrendszeremnek – jelenleg nincsenek elterjedt ipari megoldások ennek megvalósítására. A szenzoradatok önmagukban is értékes következtetésekre vezethetnek, az eseményadatokkal együtt elemezve pedig további, komplex eredményekre juthatunk. Ezek közül mutatok be néhányat a továbbiakban.

Fizikai és kijelzés orientáció: A mobil alkalmazások egyik általános tulajdonsága, hogy képesek megfelelően kezelni azt, ha a felhasználó elfordítja a kijelzőt. Ilyenkor a fejlesztőkre van bízva az, hogy a tartalmat ennek megfelelően rendezzék át – persze csak ha ezt szükségnek látják.

A keretrendszerem folyamatosan naplózza az egyszerűbb orientációs szenzor (SimpleOrientationSensor) jelzéseit, ami képes megmondani azt, maga a kijelző milyen pozícióban van, illetve az alkalmazás DisplayProperties tulajdonságai közül az Orientation nevűt, ami megmutatja azt, hogy maga a kijelzés milyen módban történik. Ha megváltozik a kijelző orientációja (jelez a SimpleOrientationSensor) akkor ideális esetben a DisplayProperties Orientation tulajdonsága ezt követi. Erre látunk egy példát a 131. munkamenetben az alkalmazás képgaléria oldalán:

CurrentOrientation	EventName	SensorSource	SimpleOrientation	Time
NULL	NULL	SimpleOrientationSensor	Rotated270DegreesCounterclockwise	16:31:51
Portrait	DisplayOrientationChanged	NULL	NULL	16:31:51

7. táblázat - Összetett adatelemzés: forgatás

A 7. táblázat adataiból leolvasható, hogy ugyanabban a másodpercben változott a kijelző és a kijelzés orientációja is. Természetesen először a kijelző orientációja változik meg – a felhasználó elforgatta az óramutató járásával ellentétesen 270°-kal. Ezután a kijelzés is megváltozott, az óramutató járásával megegyezően 90°-t fordult (Portrait) – vegyük észre, hogy a két forgatás ugyanazt eredményezi, csak más módon.

Tovább vizsgálva az adatokat azonban kiderül, hogy például 127. munkamenetben elforgatta a felhasználó a kijelzőt többféle módon is, mégsem változik a kijelzés orientációja. Ennek oka, hogy az előzőleg vizsgált munkamenetben a galériaoldalon történt a forgatás (ahol implementáltuk a kijelzés módjának változását is), ez utóbbi esetben a főoldalon (ahol pedig nem).

Az evidens kérdés, amit ennek a szenzornak az adatai alapján fel kell tennünk:

Kérdés: A felhasználók elforgatják a kijelzőt, de én erre nem reagálok. Szükséges lenne mégis?

Ennek eldöntése nagyban függ az alkalmazástól, sőt – ahogy az előző példában láttuk – attól is, hogy melyik lapra értelmezzük a kérdést. A vizsgált adatok során mindössze két felhasználó próbálkozott a forgatással, így feltételezhetően helyesen döntöttünk, amikor nem fordítottunk az implementációra erőfeszítéseket.

DPI: Nem közvetlenül szenzorokhoz kapcsolódó mérés ugyan, érdemes azonban megvizsgálni, milyen DPI beállítás mellett használják a felhasználók leggyakrabban az alkalmazást. Fontos ismerni ezt az értéket, hogy tudjunk választani egy olyan beállítást, amelyre elsődlegesen tudunk tervezni (persze a platform filozófiájával összhangban a többire is fel kell készülnünk, lásd 2.1 részben bemutatottak). Ez a mérések alapján **96 DPI**.

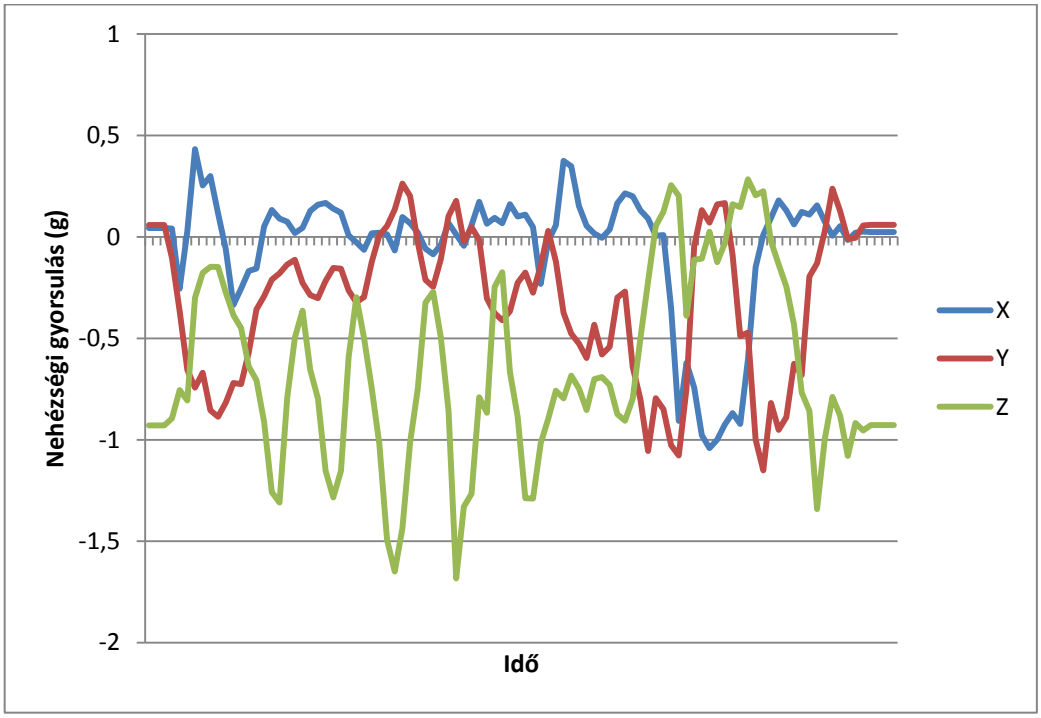
A gyorsulásmérő: A gyorsulásmérő adataiból két dologra mutatok rá. Az első egy kicsit komikusnak ható, valójában azonban annál fontosabb megfigyelés.

A legtöbb ember gondolkodása még a digitális világban is meglehetősen mechanikus. Ha valami nem működik, akkor tipikus reakció, hogy megkocogtatjuk, ráütünk, belerúgunk, ráfújunk, vagy éppen megrázzuk. Nincs ez másképpen a mobil eszközökkel sem – ha „nem működik”, akkor (különösen azok, akik kevésbé vannak hozzászokva az ilyen eszközökhöz) fizikai buzdítással próbálnak az eszközből választ kicsikarni. Az egyik ilyen „gesztus” az, hogy megrázzák a tabletet, a gyorsulásmérő pedig képes ezt érzékelni. Mivel a rázás nem egy intuitív gesztus, nagyszámú ilyen esetén joggal lehetünk kíváncsiak:

Kérdés: Miért rázzák folyamatosan a felhasználóim az eszközt? Talán nem elégedettek vele?

Szerencsére a naplózott adatok között nincs ilyen esemény, ami megint sikerélményre ad okot.

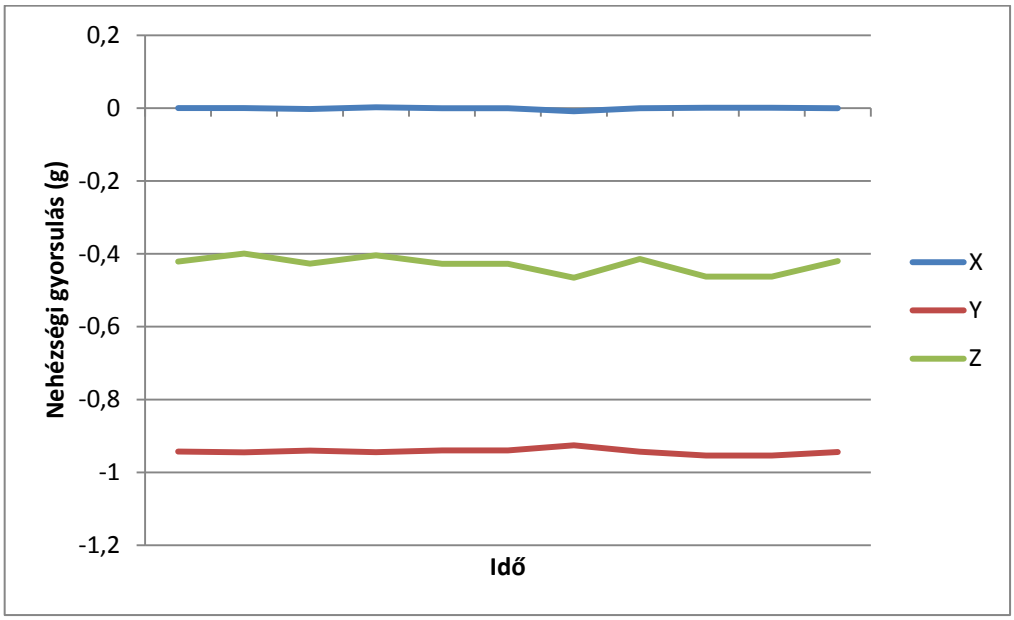
A másik adatelemzési lehetőség kevésbé kézenfekvő, viszont annál hasznosabb. A gyorsulásmérő a fizikából ismert nehézségi gyorsulás nagyságát méri a tér 3 irányában. Képzeljük el egy pillanatra, ahogy a kezünkbe vesszük a tabletet és sétálunk vele. A szokásos módon választott koordinátarendszerben az X és az Y irányú gyorsulás kevésbé változik – mivel azonban lépéskor a lábunkat először felemeljük, majd letesszük, az egész testünket megmozgatjuk a Z irány mentén – ezzel pedig a kezünkben lévő tabletet is, így jelentősen változik abban az irányban a nehézségi gyorsulás. Erre látunk példát a 127. munkamenetben:



12. ábra - Mozgás érzékelése gyorsulásmérővel

A munkamenet első felében történt a mozgás változása (a mérés utolsó harmadában egy forgatást is láthatunk; itt a SimpleOrientationSensortól is kaptunk erre vonatkozó eredményeket) – szépen látszik a Z irányú erőhatások nagyobb mértékű, közel periodikus, valamint a másik két tengely mentén észlelhető kisebb frekvenciájú és amplitúdójú változások. Ez a mérés konzisztens egyéb hasonló kísérletekkel (lásd pl.: [24]).

Összehasonlításképpen egy asztalon tartott munkamenet (pl.: 131) képe:

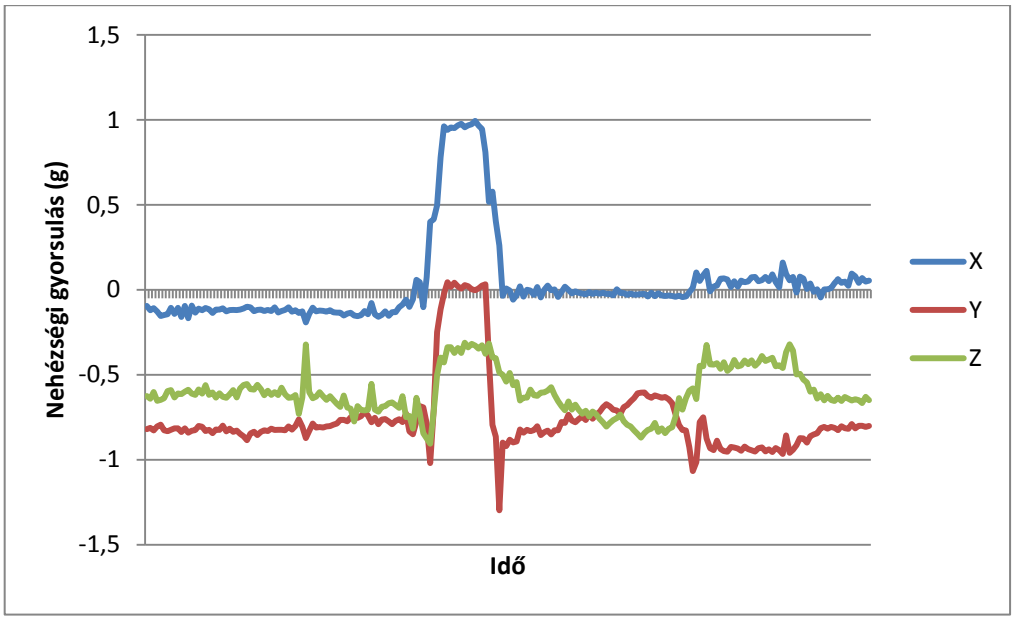


13. ábra - Gyorsulásmérő nyugalmi helyzetben

Mivel mobil eszközökről van szó, semmi nem garantálja azt, hogy felhasználóink mindig asztal mellett vagy ágyban fogják használni alkalmazásainkat. Mozgás közben viszont legalább az egyik kezükkel tartják a tabletet, így bonyolultabb gesztusokat egyszerűen nem tudnak végrehajtani. További problémát okozhat, hogy járás közben a figyelmük is megosztott, ezzel még jobban leszűkítve az interakciók számát. Ezzel összhangban a 127. munkamenetben (a jelentéktelen mutatóeseményeken és az életciklus-eseményeken túl) **mindösszesen 9 eseményt rögzítettünk, amelyből ráadásul 5 navigációs célú volt**. Ezek után tehát világos, hogy érdemes ismernünk felhasználónk szokásait – ha a célközönség inkább mozgás közben használja az alkalmazást, akkor egy egyszerűbb interakciós modellt implementáljunk alkalmazásunkba. Ezek után egy evidens kérdés lehet:

Kérdés: Tekintettel vagyok-e arra, hogy felhasználóim sokat mozognak alkalmazáshasználat közben?

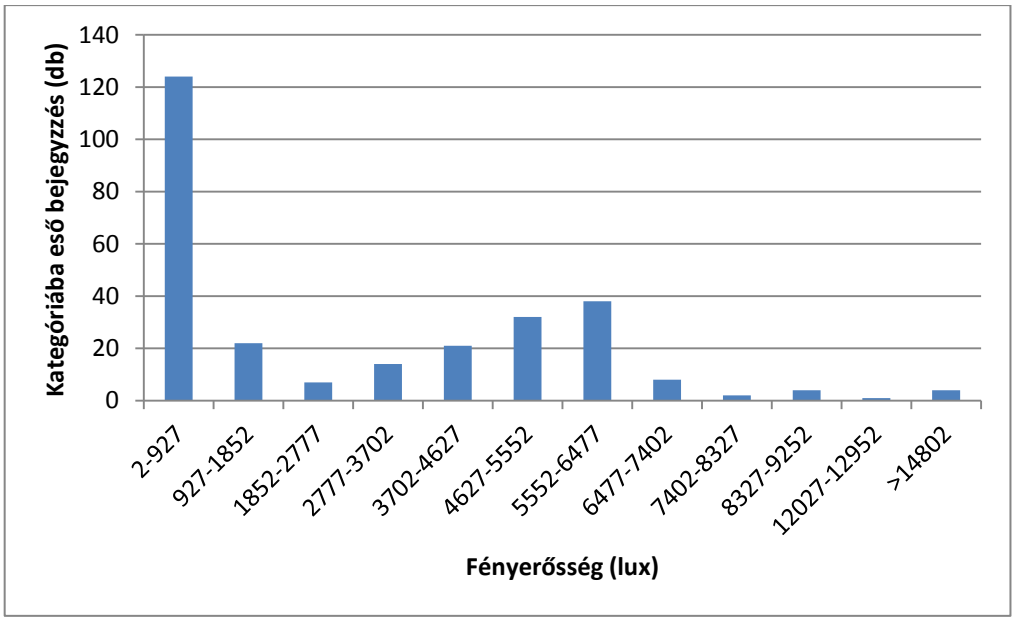
Esetenként érdekes kiugrásokat találhatunk a mért adatokban. Ilyet láthatunk például a 132. munkamenetben:



14. ábra - Forgatás a gyorsulásmérőn

Az ilyen jellegű kiugrások okának érdemes mindig utánagondolni. Természetesen ezek az okok esetenként nagyon egyediek lehetnek, azonban a többi szenzor adatiból következtethetünk rájuk. Ebben az esetben például valószínű, hogy egy forgatásról van szó, mert a SimpleOrientationSensortól is elég sok mérést kaptunk ugyanebben a sessionben.

A fényszenzor adatai: A fényszenzor adatainak elemzése egyszerű, mégis fontos információkhoz juthatunk belőle. Egyszerű hisztogramot készítve és az adatokat ábrázolva a következő diagramot kaptam:



15. ábra - A használat körülményeinek fényviszonyai hisztogramon

A 15. ábra alapján a leggyakoribb fényerő, amely mellett az alkalmazást használják 2 és 927 lux közé esik, ami a majdnem teljes sötétségtől egy borús nap vagy erős beltéri világítás fényerejéig terjed [25]. Ez nagyjából várakozásainknak megfelel – a legtöbb ilyen eszközt valóban beltérben használjuk. Fontos azonban ismerni ezeket a fényviszonyokat, mert például egy tipikusan sötétben használt alkalmazás hátterét nem szabad tiszta fehérre változtatni – ez nagyon megerőltető a szemnek. Érdemes tehát legalább érintőlegesen megvizsgálni az alkalmazás tipikus használati fényviszonyait.

Az inklinometer adatai: Az inklinometer adatai szintén nagyon fontosak. Emlékeztetőül, ez az a szenzor, ami megmondja, hogy az eszköz saját tengelyei mentén mennyire van elforgatva (lásd részletesebben 6. ábra).

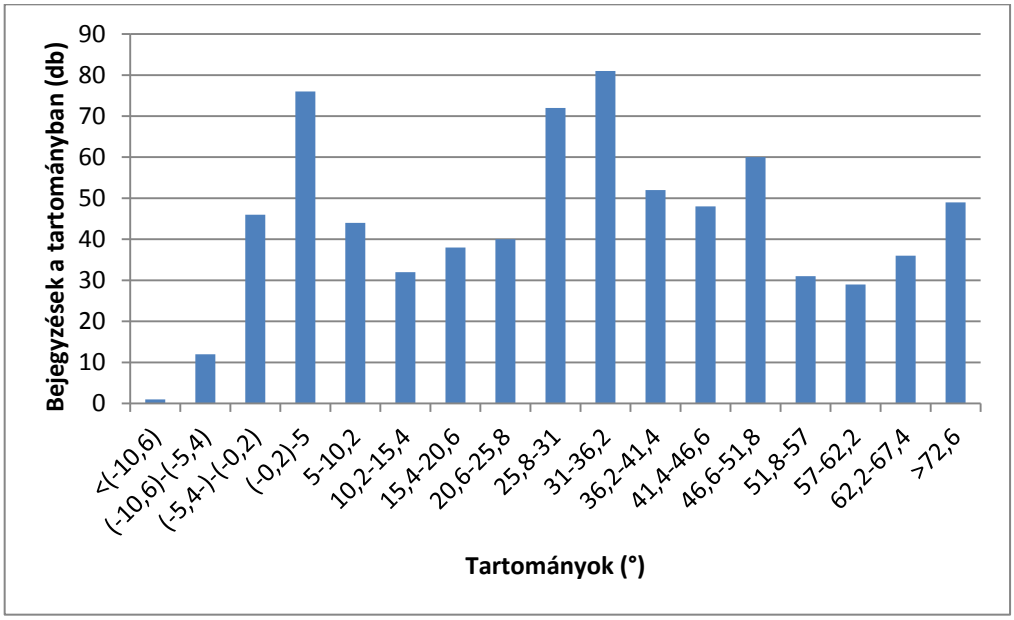
A témával kapcsolatban rengeteg tanulmányt készítettek, ami megmutatta az emberi test számára kényelmes tartási pozíciókat. A legtöbb ilyen tanulmány a pitch értékét vizsgálta és arra a következtetésre jutott, hogy a legkényelmesebb tartási pozíció 30° környékén van [26]. A roll értékére a kényelmes érték egyszerű megfontolásból 0° (ilyenkor a kijelzőnek a bal és a jobb széle is azonos távolságra van a szemünktől, azaz „szemből” látjuk). A yaw értékére kevesebb ilyen megkötés van (gondoljunk csak például arra, hogy sokan jegyzetelés közben is elforgatják a füzetet a padon), de mivel a legtöbb tartalom párhuzamos a tablet hosszabb oldalaival, így ennek is a kisebb tartományokban kell mozognia.

Ezek az értékek anatómiai szempontból fontosak – ezek fárasztják a legkevésbé a felhasználót. Ha valami miatt az alkalmazásunk arra készteti őket, hogy ettől nagyon

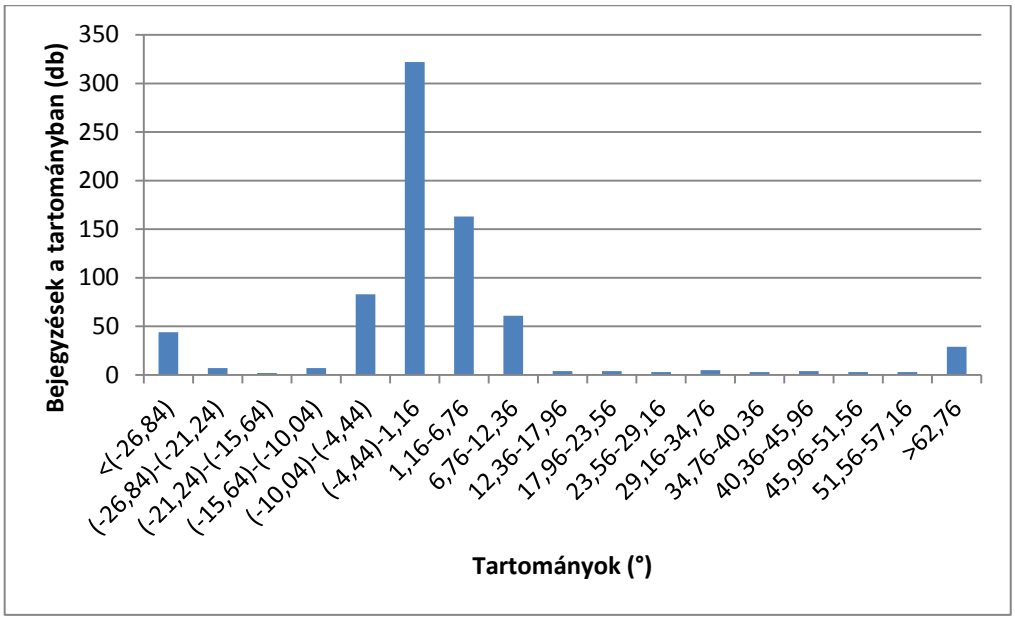
eltérő pozícióban használják az eszközt, hamar elfáradnak és váltanak. Fontos tehát meggondolni:

Kérdés: Sikerült az alkalmazás készítésekor az eszköz tartására vonatkozó anatómiai megkötéseket figyelembe venni?

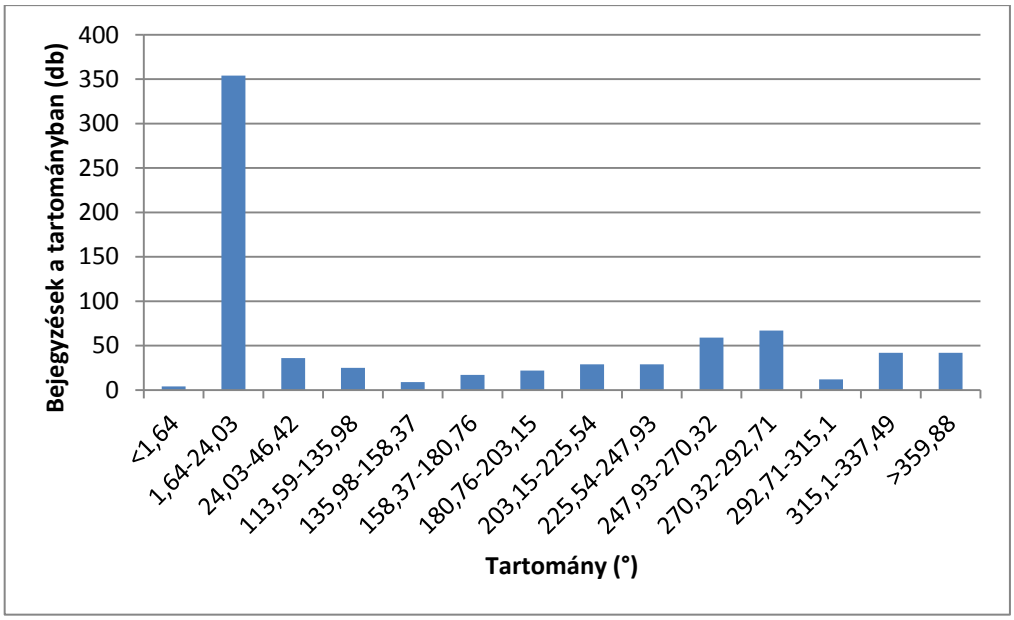
Webáruházunk esetén büszkén jelenthetjük, hogy sikerült. Az alábbi hisztogramok ábrázolják a pitch, roll és yaw értékeinek gyakoriságát:



16. ábra - A pitch értékeinek eloszlása hisztrogramon



17. ábra - A roll értékeinek eloszlása hisztogramon



18. ábra - A yaw értékeinek eloszlása hisztogramon

A hisztogramokról könnyedén leolvasható, hogy alkalmazásunk valóban megfelel az ergonómiai követelményeknek: a pitch értéke a 30° körüli tartományokba, a roll értéke pedig a 0° körüli tartományokba esik a leggyakrabban, és a yaw leggyakoribb értékei is a 30° alatti tartományból kerülnek ki.

Az iránytű mérési adatai: Az iránytű mérési adatainak hasznosítása nem magától értetődő. A legtöbb felhasználó – mint ahogy a legtöbb ember – sokszor még csak tisztában sincs azzal, hogy merre van észak, nemhogy azzal foglalkozzon, hogy az alkalmazást északi irányba nézve használja. Mindemellett semmilyen ergonómiai hatása nincs annak, hogy melyik irányba nézve használjuk a szoftvert (hacsak nem arra gondolunk, hogy a szabadban nem jó a Napnak háttal állni, mert a kijelző nehezen látható).

Amit mégis meg tudtam vizsgálni az iránytű adataiból az az, hogy a felhasználók az egyes munkamenetek során mennyit forognak. Ez – a gyorsulásmérő adataihoz hasonlóan – azért fontos, mert a forgáshoz szükség van arra, hogy legalább egyik kézzel az eszközt magát tartsák.

Az adatokból a következő egyszerű következtetést sikerült levonni: a munkamenetek során vagy közel egy helyben voltak a felhasználók (pl. a 129. vagy a 130. munkamenet) vagy megtettek közel egy fordulatot. Ez utóbbiakat az alábbi táblázat mutatja:

Munkamenet	Minimum irány (°)	Maximum irány (°)
127	5,0	358,4
131	0,9	356,9
132	1,8	357,1

8. táblázat - Az iránytű adatainak összegzése

A 8. táblázat mutatja, hogy egyes munkamenetekben a felhasználó leírt közel egy teljes kört (az értékek majdnem 0°-tól egészen 360° közeléig változnak). Ez tehát azt jelenti, hogy a felhasználók vagy nem mozognak, vagy ha forognak, leírnak egy teljes kört. Ez legrosszabb esetben azt jelenti, hogy állandóan forgolódnak, ez pedig hasonló következményekkel jár a gyorsulásmérőnél leírtakhoz. Ha tekintettel vagyunk az ott leírtakra, akkor azzal az iránytű által felvetett problémákat is megoldjuk.

A giroszkóp és a komplex orientációszenzor adatait sajnos a felhasználók szokásainak monitorozására nem igazán lehet felhasználni. Előbbi a forgatások gyorsaságát méri a tér 3 iránya mentén – a legtöbb alkalmazás (a játékok kivételével) azonban nem reagál különböző módon a gyors vagy a lassú 90°-os elforgatásra, így a felhasználók sem várnak más eredményt a gyors és a lassú forgatások után (és ezek mellett ergonómiai különbség sincs köztük). A komplex orientációszenzor adatai pedig túl bonyolultak ahhoz, hogy felhasználói szokásokat figyeljünk meg velük. Ennek (és a giroszkópnak is elsősorban) az a legfőbb célja, hogy különböző játékprogramokban élethűen tudjuk modellezni a világot.

4.5 A GeoLocation platform adatai

A GeoLocation platform adatait kétféleképpen tudjuk felhasználni. Az egyik felhasználási lehetőség demográfiai adatok gyűjtése. Behatárolhatunk például olyan környékeket (országok, államok, városok) ahol alkalmazásunkat kevesen használják. Ennek okát érdemes kideríteni és tenni ellene – mondjuk megfelelő promóciós és marketingeszközökkel. Webáruházunkból egyelőre csak a tesztelés során szereztünk adatokat, így ennek vizsgálata értelmetlen.

Emellett bizonyos szenzorok képesek sebességadatokkal is szolgálni – ezekből pedig egyértelműen tudunk például arra következtetni, hogy például valaki utazás közben (vezetés, vagy sokkal inkább vonatozás/metrózás közben) használja alkalmazásunkat. Ez is fontos és bár a mérési keretrendszer naplózza ezeket az adatokat, sajnos egyelőre kevés beépített GPS szenzor tudja ezt az adatot szolgáltatni – a tesztelés során használt eszközökben található GPS sem képes erre.

5 Ipari alkalmazás és további fejlesztési lehetőségek

Az alábbi néhány fejezetben kitérek az ipari felhasználási lehetőségekre és összehasonlítom a keretrendszert egy már létező, hasonló célú megoldással, majd további fejlesztési lehetőségeket vázolok fel.

5.1 Az eddigi felhasználás

Az előző fejezetben bemutatott adatok tehát egy éles, ipari alkalmazás tesztelési szakaszából származnak – ez volt a keretrendszer felhasználásának első éles tesztje. Az előző fejezet adatelemzései alapján bebizonyosodott, hogy a keretrendszer alkalmas az elsődleges feladatára, a naplózás megvalósítására egy ipari alkalmazásban és a gyűjtött adatok valóban értékesek. Mindemellett segített monitorozni a tesztelési ciklust és a tesztelői tevékenységet, ami hasznos visszajelzésekkel szolgált a fejlesztői csapatnak. További hasznos funkciója a modulnak, hogy párhuzamosan végezhetőek a funkcionális tesztelések (amelyeket a szoftvertesztelők a megfelelő technikákat alkalmazva végeznek) és a használhatósági tesztelések, hiszen a modul folyamatosan és automatikusan gyűjti az adatokat, amik elemezhetőek.

Ezek adják tehát az elsődleges ipari alkalmazhatóságát a keretrendszernek: az előző fejezetben bemutatott felhasználói élmény automatizált vizsgálata, valamint a használhatósági tesztelések segítése és a hozzájuk tartozó adatok szintén automatikus gyűjtése.

5.2 Összehasonlítás a Google Analytics keretrendszerrel

Léteznek már hasonló megoldások az iparban, az egyik ilyen a Google Analytics keretrendszer. Természetesen nehéz összehasonlítani ezt a keretrendszert a saját modulommal; egyrészt mert előbbi a világ egyik legnagyobb IT profilú cége fejleszti hatalmas csapattal és sokéves tapasztalattal, másrészt pedig az elsősorban webhelyeken képes monitorozni a felhasználói szokásokat, nem pedig asztali vagy tabletes alkalmazásokban. Ezzel együtt most mégis felsorolok néhány előnyt és hátrányt, amivel a saját modulom rendelkezik (és néhány dolgot, amire mindkettő képes) a Google Analytics (továbbiakban GA) megoldáshoz képest.

Egy fontos aspektusa mindkét megoldásnak az eseménynaplózás. A Google keretrendszerében ezek úgy történnek meg, hogy egy megfelelő URL-t küldünk a Google felé, amiben van egy kategória, egy eseménynév és két opcionális paraméter

(ebben szinte bármi lehet). Ezután a weben lehet szűrni eseményekre, kategóriákra, a paraméterekre. A saját keretrendszerem kategóriákat nem kezel – ezek bevezetésére viszont adott a lehetőség. Az események paraméterei közül pedig nem csak kettőt, hanem tetszőleges számút lehet naplózni (jelenleg is rengeteg paraméter van naplózva egy-egy eseményhez, részletesen lásd a B Függelékben). Bár a GA-ban is van lehetőség arra, hogy egyéb paramétereket esetenként naplózzunk, ezek száma fiókonként legfeljebb 5 lehet. Elmondható tehát, hogy ezen a téren jobban teljesít a keretrendszerem.

Mivel a szenzoradatok naplózása is eseményalapú, ezért ezekhez kapcsolódóan ugyanazokat az előnyöket és hátrányokat lehet elmondani, mint a felhasználói eseményekhez kapcsolódóan. A szenzoradatok esetében azonban fokozottabban jelentkezik hátrányként az, hogy a GA nem ad lehetőséget arra, hogy nagy mennyiségű, tetszőleges adatot naplózzunk.

A GA képes munkamenetet definiálni, visszatérő felhasználókat azonosítani. Erre az én modulom is képes. Az előző fejezetben említett webáruház alkalmazásban a megrendelő kérésére implementáltam a GA API hívásait is – ehhez viszont egy saját munkamenet-kezelést is kellett implementálni, amihez hasonló felhasználó-azonosítási módszert választottam, mint amit a saját modulomban is alkalmaztam.

Az előzőekben definiált munkamenethez tartozik természetesen egy hossz is. Ezt a GA úgy definiálja, mint a munkamenet indítása és a munkamenetbeli utolsó lapváltás között eltelt időszak [27]. Ennek a módszernek természetesen egyértelmű hátránya, hogy az utolsó oldalon eltöltött idő nem kerül beszámításra a munkamenetbe. A saját modulom a munkamenetet magát úgy definiálja, hogy adott felhasználótól olyan események sorozata, amelynek szomszédos tagjai között legfeljebb 5 perc telik el. Ennek a módszernek is van egy egyértelmű hátránya, hiszen 5 percen belül esetenként akár több munkamenetet is indíthat egy felhasználó, ez mégis egyben jelenik meg az adatokban.

A GA keretrendszer tud a lapváltásokkal kapcsolatban is adatokat szolgáltatni (egy lapon eltöltött idő, átlagosan adott lapon töltött idő, átlagosan egy lapon töltött idő, látogatások száma, navigációs gráf stb.). Erre a saját modulom is tökéletesen alkalmas, hiszen naplózom a lapváltásokat, a hozzájuk tartozó időbélyeget és hogy mely lapokról van szó.

A GA-ban folyamatosan naplózva vannak demográfiai információk a felhasználókról (pl.: az alkalmazás nyelve, a felhasználók helye). Ezeknek egy részét a munkamenethez tartozóan magának a programozónak kell definiálnia (ilyen a nyelv), egy másik részét „kitalálja” a GA (pl. pozíciót a hálózati címek alapján). A programozó

által adott munkamenethez definiált adatok naplózása természetesen egyszerűen megoldható, míg például a pozícióra vonatkozó információk egyébként is folyamatosan naplózva vannak.

A GA egyáltalán nem tud olyan dolgokat naplózni, mint a kijelzés orientációja vagy a felbontás. Természetesen saját, asztali vagy tablet GA-implementációban akár ez is megoldható kerülőutakon, de webes környezetben semmiképp, hiszen a böngésző nem fér hozzá ilyen információkhoz biztonsági okokból.

Ezen felül a GA használata sajnos esetenként meglehetősen nehézkes. Az elsősorban webes környezet miatt különböző JavaScript függvényeket kell meghívni a megfelelő helyeken. Ez jóval több programozói beavatkozást igényel, mint amit az én keretrendszerem (lásd részletesen az A Függelékben).

A GA működésének lényege egyébként csupán annyi, hogy megfelelően felépített URL-eket kell elküldeni a megfelelő címre. Így tehát van lehetőség implementációra akár asztali vagy tablet alkalmazások monitorozásához is, mint ahogyan említettem fentebb is. Sajnos azonban az API-nak ez a része nagyon rosszul dokumentált és egyáltalán nem tesztelhető az implementáció helyessége, így nem webes környezetbe nem ajánlatos ennek a használata.

Természetesen a GA mögött hatalmas infrastruktúra van, mind az adatok tárolására, mind pedig azok elemzésére vonatkozóan, amit egy ilyen volumenű munka nem tud felvonultatni. Nem is volt feladata – a fő cél az volt, hogy a sikerüljön behatárolni a megfelelő adatokat és bizonyítani, hogy ezekből értékes következtetéseket lehet levonni.

5.3 Hibák, hiányosságok és fejlesztési lehetőségek

Néhány ponton érdemes lehet a későbbiekben fejleszteni a modul működését.

- A 3.1 részben bemutatott eseménykezelő mechanizmus a reflexiós API hiányosságai miatt nagyon megnehezíti a dinamikus feliratkozást az eseményekre. Bár az API nem specifikálja az eseményre feliratkozott metódusok lefutási sorrendjét (mint ahogy a .NET keretrendszerben sem), tapasztalat, hogy a feliratkozás sorrendjében futnak le az metódusok (mint ahogy .NET keretrendszerben is). Mivel a saját eseménykezelőm, amiben a naplózás meg van valósítva, utolsóként iratkozik fel (erre nincs más lehetőség, mert a Visual Tree teljes mértékben fel kell, hogy legyen építve a feliratkozáskor), utolsóként is fut le. Viszont emiatt könnyen előfordulhat, hogy a már lefutott

eseménykezelők megváltoztatják a környezetet, amit naplózni kellett volna (példa erre az a lenyomás, ami navigál, azaz teljesen új környezetet hoz létre). Ez esetenként problémákat vet fel, amire találni kell egy általános megoldást (például a navigálás miatt keletkező problémák önmagukban kezelhetőek a navigációs eseményekből történő utólagos rekonstrukcióval – ezt figyelembe is vettem az előző fejezetekben).

- Az események naplózásakor a buborékozás jelenségének jelenlegi kezelését átgondoltam és megfelelőnek tartom. Ettől függetlenül elképzelhetőnek tartok olyan kivételes eseteket, amikor a programozó szeretné naplózni az események további terjedését is. Ehhez az API-ban érdemes lehet biztosítani egy megfelelő mechanizmust.
- Jelenleg a naplózó metódusok leiratkozása nem történik meg az adott eseményekről. Ennek a hatását a memóriahasználatra, a szemétgyűjtésre még vizsgálni kell és szükség esetén a megfelelő helyen kezelni a leiratkozást. Ehhez az eszközök adottak, hiszen a navigációs eseményekben ez kezelhető, a feliratkozáskor megkapott token pedig rendelkezésemre áll.
- A hálózaton átutazó adatfolyamot érdemes lehet majd tömöríteni.
- A felhasználó saját gépére történő naplózás minden esetben nehéz feladat, hiszen kompromisszumot kell kötni a tárolható adatok mennyisége és pontossága között. További tapasztalatok gyűjtése szükséges arra vonatkozóan, hogy a jelenleg naplózott adatok mennyisége megfelelő, vagy tovább csökkenthető (esetleg növelni kell). Ehhez természetesen a platform terjedése is szükséges, hiszen az, hogy mely interakciók lesznek népszerűek a programozók és a felhasználók körében, csak sok alkalmazás írása után derülhet ki.
- Az adatok elemzése egy kreatív folyamat, folyamatos tanulás és tapasztalatszerzés eredménye. A dolgozatban bemutatott eredményeken túl további következtetések levonása, mindig újabb és újabb összefüggések feltárása folyamatosan cél.
- A szenzorok analóg környezetet digitális formában rögzítő rendszerek. Az ilyen rendszerek egy gyakori hibája a „jitter” jelenség – ilyenkor a szenzorok nagyon kis változásokat rögzítenek szinte állandó gyakorisággal. Ennek kezelésére lettek bevezetve a küszöbváltozók, maga a naplózás pedig csak akkor történik meg, ha az adott szenzorra vonatkozó küszöbváltozónál nagyobb a változás mértéke. A megoldás

megfelelő, de a szabályozástechnikában ez egy ismert probléma és vannak ennél kifinomultabb, bevett megoldások a jelenség kezelésére (pl. aluláteresztő szűrő, aminek egy limitált megvalósítása a sajátom). Ezeknek a vizsgálata és alkalmazása hasznos lehet a későbbiekben.

6 Összefoglalás

A dolgozatban a felhasználói élmény és a szoftver-használhatóság mérése alkalmas adatok gyűjtésére dolgoztam ki egy automatikus keretrendszert, részletesen implementáltam egy referencia-alkalmazást a Windows 8 platformra, majd bemutattam a modul használatát és a gyűjtött adatokból különböző egyszerű és komplex következtetéseket vontam le.

Elsőként megvizsgáltam a felhasználói élmény és a szoftver-használhatóság kérdéskörének jelentőségét, majd a mobil eszközök jelenlegi és várható piaci terjedését mutattam be különböző tanulmányokra hivatkozva. Ezekkel kapcsolatban megvizsgáltam röviden a mobil eszközök használata miatt felmerülő kérdéseket a használhatóság és a felhasználói élmény területén.

Ezután bemutattam a Windows 8 platformnak a dolgozat szempontjából fontos tulajdonságait: röviden kitértem az operációs rendszer általános tulajdonságaira és az alkalmazások életciklusára. Ezek után részleteiben bemutattam az eseménykezelés modelljét és magukat az eseményeket, valamint az új Sensor Fusion platform működését.

Azt ezt követő fejezetekben bemutattam a referencia-alkalmazás működését: általánosan az adatgyűjtési folyamatot, a keretrendszerben használt adatmodellt, a modul felhasználását egy kész alkalmazásban. Külön kitértem a szenzoradatok és az események naplózására.

A keretrendszer bemutatása után a gyűjtött adatokat különböző statisztikai módszerekkel analizáltam, összefüggéseiben vizsgáltam meg, így hasznos következtetéseket tudtam levonni a felhasználók alkalmazás- és eszközhasználati szokásaira vonatkozóan. Különböző vizualizációs eszközökkel sikerült néhány felhasználói interakciót rekonstruálni és „rávetíteni” az alkalmazás felületére.

Fontos, hogy az elemzés során nem csak adott alkalmazással kapcsolatban sikerült rámutatni fontos tulajdonságokra, hanem a keretrendszer működési elvének helyessége is bizonyításra került – azaz a gyűjtött adatok valóban alkalmasak arra, hogy mérjük vele a felhasználók szokásait. Ezáltal lehetőség van arra, hogy ugyanezt a keretrendszert akár egy teljesen más mobilplatformra (Android, Apple) implementáljuk és ott is hasznos adatokat gyűjtsünk róla.

Bemutattam azt is, hogyan válhat a keretrendszer hasznos eszközzé már az alkalmazás tesztelési fázisában és lehet hatékony kiegészítője a használhatósági tesztelések során alkalmazott eszközöknek.

Végül összehasonlítottam saját munkámat egy hasonló, de sok helyen koncepcionálisan eltérő monitorozó keretrendszer működésével – ekkor megállapítottam, hogy hosszú távon saját modulom az elvi alapjait megtartva akár versenyképessé is válhat ezzel megoldással. Utolsóként pedig felvázoltam néhány lehetséges fejlesztési irányt a keretrendszer bővítésére és hatékonyabbá tételére.

Irodalomjegyzék

[1] J. J. Garrett, The Elements of User Experience (2nd Edition), New Riders Publishing, 2010.

[2] P. Ferenc, Magyar értelmező kéziszótár, Akadémiai Kiadó, 2011.

[3] J. Sauro, „8 Research Based Insights for User Experience Surveys," 2011. május 4. [Online]. Elérhető: http://www.measuringusability.com/blog/ux-surveys.php. [Hozzáférés dátuma: 2012. október 19.].

[4] „SUMI," [Online]. Elérhető: http://sumi.ucc.ie/. [Hozzáférés dátuma: 2012. október 16.].

[5] *ISO/IEC 9126 Software engineering — Product quality*, 2001.

[6] „ICT Facts and Figures," 2011. [Online]. Elérhető: http://www.itu.int/ITU-D/ict/facts/2011/material/ICTFactsFigures2011.pdf. [Hozzáférés dátuma: 2012. október 16.].

[7] „Portio Research Mobile Factbook," 2012.

[8] „World Population Prospects, the 2010 Revision," 2011. június 28. [Online]. Elérhető: http://esa.un.org/wpp/Excel-Data/population.htm. [Hozzáférés dátuma: 2012. október 16.].

[9] „Mobile Devices Will Outnumber World Population By 2016: Cisco Study," *Huffington Post*, 2012. február 14.

[10] „Media Tablet Shipments Outpace Fourth Quarter Targets; Strong Demand for New iPad and Other Forthcoming Products Leads to Increase in 2012 Forecast, According to IDC," 2012. március 13. [Online]. Elérhető: http://www.idc.com/getdoc.jsp?containerId=prUS23371312. [Hozzáférés dátuma: 2012. október 16.].

[11] G. Gear, Szerző, *Architecting and Integrating Sensor Drivers.* [Performance]. Microsoft Corporation, 2011.

[12] „Application lifecycle (Windows Store apps)," 2012. szeptember 4. [Online]. Elérhető: http://msdn.microsoft.com/en-us/library/windows/apps/hh464925.aspx. [Hozzáférés dátuma: 2012. október 16.].

[13] „MSDN: Routed Events Overview," [Online]. Elérhető: http://msdn.microsoft.com/en-us/library/ms742806.aspx. [Hozzáférés dátuma:

2012. október 16.].

[14] „Wikipedia. Touchscreen,” [Online]. Elérhető: http://en.wikipedia.org/wiki/Touchscreen. [Hozzáférés dátuma: 2012. október 16.].

[15] „MSDN: Gestures, manipulations, and interactions (Windows Store apps),” 2012. szeptember 4. [Online]. Elérhető: http://msdn.microsoft.com/en-us/library/windows/apps/hh761498.aspx. [Hozzáférés dátuma: 2012. október 16.].

[16] G. Gear, Szerző, *Using location & sensors in your app.* [Előadás]. Microsoft Corporation, 2011.

[17] Microsoft, „Integrating Motion and Orientation Sensors,” 2012.

[18] „Wikipedia: Global Positioning System,” [Online]. Elérhető: http://hu.wikipedia.org/wiki/Global_Positioning_System. [Hozzáférés dátuma: 2012. október 16.].

[19] Á. Nagy, „Hatékony sorosítás dinamikus kódemittálással .NET platformon,” in *BME-VIK TDK*, Budapest, 2011.

[20] A. Kapin, „Google Analytics Report Shows People Are Spending Less Time on Websites,” 2011. július 2. [Online]. Elérhető: http://www.frogloop.com/care2blog/2011/7/2/google-analytics-report-shows-people-are-spending-less-time.html. [Hozzáférés dátuma: 2012. október 16.].

[21] E. Holter, „Average Website Traffic Statistics,” 2004. július [Online]. Elérhető: http://www.newfangled.com/average_website_traffic_statistics. [Hozzáférés dátuma: 2012. október 16.].

[22] „What is the Average Conversion Rate?,” 2007. július 31. [Online]. Elérhető: http://www.searchmarketingstandard.com/what-is-the-average-conversion-rate. [Hozzáférés dátuma: 2012. október 16.].

[23] „PointerPointProperties.MouseWheelDelta,” 2012. szeptember 4. [Online]. Elérhető: http://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.input.pointerpointproperties.mousewheeldelta. [Hozzáférés dátuma: 2012. október 16.].

[24] S. Sinofsky, „MSDN Blogs: Supporting sensors in Windows 8,” 2012. január 24. [Online]. Elérhető: http://blogs.msdn.com/b/b8/archive/2012/01/24/supporting-sensors-in-windows-8.aspx. [Hozzáférés dátuma: 2012. október 16.].

[25] „Wikipedia: LUX,” [Online]. Elérhető: http://en.wikipedia.org/wiki/Lux. [Hozzáférés dátuma: 2012. október 16.].

[26] T. Albin és H. McLoone, „Effect of Tablet Tilt and Display Conditions on User

Posture, Performance and Preference," in *10th Applied Ergonomics Conference*, Dallas, 2007.

[27] „Google Analytics: Visit Duration, Avg," [Online]. Elérhető: http://support.google.com/analytics/bin/answer.py?hl=en&answer=1006253. [Hozzáférés dátuma: 2012 október 18.].

[28] I. Albert, Szerző, *Szoftverfejlesztés Windows 8 platformra.* [Előadás]. 2012.

A Függelék: Fejlesztői útmutató a keretrendszer beépítéséhez

A következőkben röviden bemutatom, hogyan kell a modult beüzemelnie a fejlesztőknek. A Visual Studio 2012 Blank Application sémáját veszem alapul – természetesen egyes esetekben szükség lehet arra, hogy a fejlesztők néhány dolgot másként, vagy máshol végezzenek el az itt leírtakhoz képest (az aktuális lépés fontosabb változtatásait lásd ***kiemelve***).

1. lépés: A modulra mutató referencia felvétele

A modulra mutató referenciát úgy kell felvenni, hogy az az alkalmazás bármely részén használható és elérhető legyen. Érdemes az App.xaml.cs fájlban, az Application - leszármazott osztályban felvenni ***publikus, statikus*** tulajdonságként vagy mezőként.

```
sealed partial class App : Application
{
  public static GlobalEventLogger ev;
  ///....
}
```

2. lépés: A modul példányosítása

A megfelelő referenciák hozzáadása után a modult példányosítani kell. A konstruktornak át kell adni az alkalmazáshoz tartozó Frame objektumot, ezért fontos, hogy annak már léteznie kell az átadáskor.

Magát a modult célszerű az alkalmazás OnLaunched() metódusában példányosítani, hiszen a Frame is itt kerül példányosításra:

```
protected override void OnLaunched(LaunchActivatedEventArgs args)
{
  ///...
  var rootFrame = new Frame();

  ev = new GlobalEventLogger(rootFrame);

  if (!rootFrame.Navigate(typeof(MainPage)))
  {
    throw new Exception("Failed to create initial page");
  }
  Window.Current.Content = rootFrame;
  Window.Current.Activate();
}
```

Megjegyzés: Maga a modul nincs felkészítve arra, hogy többszálú környezetből is használják. Célszerű megfontolni a megfelelő tervezési minták (pl.: szálbiztos Singleton) használatát.

3. lépés: Életciklus-események monitorozása

A fenti kód még nem képes az életciklus-események naplózására. Ha szeretnénk ezeket is naplózni, a modul példányosítása után a következő kódot kell beilleszteni (opcionális lépés):

```
protected override void OnLaunched(LaunchActivatedEventArgs args)
{
  /// ....
  var rootFrame = new Frame();
  ev = new GlobalEventLogger(rootFrame);

  this.Resuming += ev.LogResuming;
  this.Suspending += ev.LogSuspending;
  Window.Current.Activated += ev.LogActivated;
  Window.Current.Closed += ev.LogClosed;

  if (!rootFrame.Navigate(typeof(MainPage)))
  {
    throw new Exception("Failed to create initial page");
  }
  /// ....
}
```

4. lépés: Referencia-görgetősáv kijelölése

Megadhatunk egyetlen ScrollViewer objektumot, aminek a vízszintes irányú eltolása mindig figyelembe lesz véve a képernyő-koordináták számolásánál (opcionális lépés). Ehhez a következő kódot illesszük be az objektum példányosítása után:

```
<ScrollViewer Style="{StaticResource HorizontalScrollViewerStyle}">

  <ScrollViewer.Tag>
    <x:Boolean>True</x:Boolean>
  </ScrollViewer.Tag>


  <!--...-->
</ScrollViewer>
```

Megjegyzés: Természetesen akár kódból is beállítható a Tag tulajdonság, a lényeg, hogy a modul példányosításakor már be legyen állítva a megfelelő Scrollvieweren ez a tulajdonság.

Megjegyzés: Van lehetőség arra is, hogy közvetlenül adja meg a fejlesztő a referencia-görgetősávot; ehhez használhatja a public void SetCurrentScrollViewer(ScrollViewer sw) metódust.

B függelék: A naplózott adatok teljes leírása

A következőkben megadom azokat a Windows Runtime eseményeket és tulajdonságaikat, továbbá azokat a szenzoradatokat, amelyeket naplóz a modul.

A naplózást megvalósító keretrendszer adatmodelljét lásd a 3.2 részben.

A ConcretEventLogger osztályok a következő eseményeknek és eseményparamétereknek felelnek meg (természetesen a modell alapján tartalmazzák az ősosztályok tulajdonságait is):

- Activated (életciklus)
- Click
- Closed (életciklus)
- DoubleTapped
  - X
  - Y
  - PointerDeviceType
- DragEnter
  - RequestedDragandDropOperation
- DragItemsStarting
  - DragItemCount
  - RequestedDragandDropOperation
- DragLeave
  - RequestedDragandDropOperation
- DragOver
  - RequestedDragandDropOperation
- Drop
  - RequestedDragandDropOperation
- Holding
  - X
  - Y
  - PointerDeviceType
  - HoldingState
- KeyDown
  - IsASCII (saját definiált tulajdonság, az alfanumerikus karakterek esetén igaz, egyébként hamis)
- LogicalDPIChanged
  - LogicalDPI (az új felbontás)
- ManipulationCompleted
  - X
  - Y
  - PointerDeviceType
  - ManipulationCumulativeExpansion
  - ManipulationCumulativeRotation
  - ManipulationCumulativeScale
  - ManipulationCumulativeTranslationX

    - ManipulationCumulativeTranslationY
    - IsInertial
    - ManipulationLinearVelocityX
    - ManipulationLinearVelocityY
    - ManipulationAngularVelocity
    - ManipulationExpansionVelocity
- ManipulationDelta
    - X
    - Y
    - PointerDeviceType
    - ManipulationCumulativeExpansion
    - ManipulationCumulativeRotation
    - ManipulationCumulativeScale
    - ManipulationCumulativeTranslationX
    - ManipulationCumulativeTranslationY
    - IsInertial
    - ManipulationDeltaExpansion
    - ManipulationDeltaRotation
    - ManipulationDeltaScale
    - ManipulationDeltaTranslationX
    - ManipulationDeltaTranslationY
    - ManipulationLinearVelocityX
    - ManipulationLinearVelocityY
    - ManipulationAngularVelocity
    - ManipulationExpansionVelocity
- ManipulationInertiaStarting
    - PointerType
    - ManipulationCumulativeExpansion
    - ManipulationCumulativeRotation
    - ManipulationCumulativeScale
    - ManipulationCumulativeTranslationX
    - ManipulationCumulativeTranslationY
    - ManipulationDeltaExpansion
    - ManipulationDeltaRotation
    - ManipulationDeltaScale
    - ManipulationDeltaTranslationX
    - ManipulationDeltaTranslationY
    - ManipulationLinearVelocityX
    - ManipulationLinearVelocityY
    - ManipulationAngularVelocity
    - ManipulationExpansionVelocity
    - ManipualtionExpansionDesiredDec
    - ManipualtionExpansionDesired
    - ManipualtionTranslationDesiredDec
    - ManipualtionTranslationDesired
    - ManipualtionRotationDesiredDec
    - ManipualtionRotationDesired
- ManipulationStarted
    - X
    - Y

    - PointerDeviceType
    - ManipulationCumulativeExpansion
    - ManipulationCumulativeRotation
    - ManipulationCumulativeScale
    - ManipulationCumulativeTranslationX
    - ManipulationCumulativeTranslationY
- ManipulationStarting
    - ManipulationCenterX
    - ManipulationCenterY
    - ManipulationRadius
    - ManipulationMode
- Navigated
    - FromPage (honnan)
    - ToPage (hová)
- Navigating
    - FromPage (honnan)
    - ToPage (hová)
- OrientationChanged
    - CurrentOrientation
    - NativeOrientation
- PointerCanceled, PointerCaptureLost, PointerEntered, PointerExited, PointerMoved, PointerPressed, PointerReleased, PointerWheelChanged
    - X
    - Y
    - Pressure
    - LeftButton
    - MiddleButton
    - RightButton
    - XButton1
    - PointerDeviceType
    - XButton2
    - MouseWheel
- Resuming (életcilkus)
- RightTapped
    - X
    - Y
    - PointerDeviceType
- Suspending (életciklus)
- Tapped
    - X
    - Y
    - PointerDeviceType

Megjegyzések:

- Az eseményekről és paramétereikről az MSDN fejlesztői portálon tájékozódhat.
- A paraméterek elnevezésének konvenciója az volt, hogy ha az eseményargumentum „A” nevű tulajdonságát naplózom, a listában „A”

néven szerepel. Ha az „A” nevű tulajdonság „B” nevű tulajdonságát naplózom, akkor „AB” néven szerepel.

- Ahol nem közvetlen eseményparaméter kerül naplózásra, ott zárójelben megmagyarázom a jelentését.
- Az X és Y paraméterek minden esetben a – szükség esetén a referencia-görgetősáv értékét figyelembe véve kiszámított – megfelelő képernyő-koordináták.

C Függelék: Sorosítás

```
public byte[] Serialize()
{
  MemoryStream ms = new MemoryStream();
  BinaryWriter bw = new BinaryWriter(ms);
  bw.Write(this.GetTypeIndex(this.GetType()));
  IEnumerable<PropertyInfo> props = this.GetType()
                                    .GetRuntimeProperties();
  foreach (var prop in props.OrderBy(p => p.Name))
  {
    if (prop.PropertyType == typeof(int))
    {
      bw.Write((int)(prop.GetValue(this)));
    }
    else if (prop.PropertyType == typeof(bool))
    {
      bw.Write((bool)(prop.GetValue(this)));
    }
    else if (prop.PropertyType == typeof(double))
    {
      bw.Write((double)(prop.GetValue(this)));
    }
    else if (prop.PropertyType == typeof(float))
    {
      bw.Write((float)(prop.GetValue(this)));
    }
    else if (prop.PropertyType == typeof(DateTime))
    {
      bw.Write(((DateTime)(prop.GetValue(this))).Ticks);
    }
    else if (prop.PropertyType == typeof(Guid))
    {
      byte[] guidbytes =((Guid)(prop.GetValue(this))).ToByteArray();
      bw.Write(guidbytes);
    }
    else if (prop.PropertyType == typeof(string))
    {
      bw.Write((string)(prop.GetValue(this)));
    }
  }
  byte[] res = ms.ToArray();
  ms.Dispose();
  return res;
}
```