Jari Väisänen

PUHELINSOVELLUKSEN KEHITTÄMINEN WINDOWS MOBILE-LAITTEELLE

Puhelinsovelluksen kehittäminen Windows Mobile- laiteelle
Kajaanin ammattikorkeakoulu
Kauppa ja Hallinto
Tietojenkäsittelyn koulutusohjelma
2011

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Kauppa ja hallinto	Koulutusohjelma Tietojenkäsittelyn koulutusohjelma
Tekijä(t) Jari Väisänen	
Työn nimi Puhelinsovelluksen kehittäminen Windows Mobile- laitteelle	
Vaihtoehtoiset ammattiopinnot	Ohjaaja(t) Matti Härkönen
	Toimeksiantaja
Aika 2011	Sivumäärä ja liitteet 47 + 1

Puhelin on ollut olemassa jo yli 100 vuotta ja matkapuhelimet ovat tuoneet liikkuvan viestinnän suomalaisten taskuihin liki 40 vuoden ajan. Viime vuosikymmen oli matkapuhelinten juhla-aikaa ja laitteet kehittyivät joka vuosi yhä monipuolisemmilla toiminnoilla. Nykypäivänä ei osteta enää vain matkapuhelinta, vaan älypuhelin, jolla voidaan tehdä monia samoja tehtäviä kuin kotitietokoneella. Mobiilikehitys ei enää tarkoita vain matkapuhelimen ja sen peruskäyttöliittymän kehittämistä, vaan kaikenlaisten mobiili laitteiden ominaispiirteitä hyödyntävien sovellusten kehitystä. Ei kuitenkaan kannata unohtaa näiden laitteiden alkuperäistä tarkoitusta; mahdollisuutta soittaa puheluita.

Tämä työ ei pyri kehittämään uutta kuningassovellusta uusimmalle trendikkäälle älypuhelimelle vaan sen tavoite on paljon lähempänä matkapuhelimen keskeisintä toimintoa, puheluiden soittamista. Tavoite on puhelintoimintojen mahdollistavan sovelluksen tai ratkaisun kehittäminen Windows Mobile- käyttöjärjestelmää käyttävälle mobiililaitteelle. Työ perustuu kajaanilaiselle Luovaliike Oy:lle tehtyyn työharjoitteluprojektiin, mutta ei suoranaisesti ole heidän tilaamansa, koska työ aloitettiin vasta projektin jälkeen. Luovaliike Oy on antanut luvan projektin käytölle työn lähdemateriaalina.

Työ on toteutettu takautuvasti ensin kertaamalla kehitystyöhön liittyvät teoriat ja mukailemalla projektissa kehitetyn ohjelman suunnittelua, toteutusta, ongelmia ja testausta. Käsiteltävä teoria koostuu viidestä osasta: työssä käytetty .NET Compact Framework ja sen emoympäristö .NET Framework, .NET ympäristöjen ensisijainen ohjelmointikieli C#, rinnakkaisohjelmointi, kieltenvälinen yhteistoiminta ja kehitystyöhön läheisesti liittyvä puhelinsovellusten kehitysrajapinta TAPI 2.2.

Luovaliike Oy:n projekti katsottiin onnistuneeksi ja sen tuottama ratkaisu liitettiin osaksi HOTI Mobile Tool- sovellukseen. Tämän perusteella myös tämän työn käsittelemä ohjelma on kelvollinen ratkaisu puhelinsovelluksen suunnitteluun ja toteutukseen.

Kieli	Suomi
Asiasanat	TAPI, Windows Mobile, .NET Compact Framework
Säilytyspaikka	☐ Verkkokirjasto Theseus ☐ Kajaanin ammattikorkeakoulun kirjasto

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

THESIS ABSTRACT

School Business and Administration	Degree Programme Business Information Technology
Author(s) Jari Väisänen	
Title Development of Phone Application for Windows Mobile Device	
Optional Professional Studies	Instructor(s) Matti Härkönen
	Commissioned by
Date 2011	Total Number of Pages and Appendices 47 + 1

The phone has existed for well over 100 years and cellphones have brought mobile communications to Finns' pockets for almost 40 years. The last decade was the golden age of mobile devices with more diverse functions ensued with every passing year. Today you would not buy just a cellphone but a smartphone, a device capable of tasks similar to your desktop computer. Mobile development no longer means only the design and development of a mobile device and its core system but all kinds of software that benefit from the properties of a mobile device. One should not, however, forget the original purpose of these devices: the possibility to make phone calls.

The thesis does not attempt to develop the new king application for the latest trend smartphone, but aims much closer to the core function of a cellphone: making a phone call. The goal is the development of an application or solution that enables phone functions on mobile devices using the Windows Mobile operating system. The work is based on similar work practice project commissioned by the company Luovaliike Oy based in Kajaani. Luovaliike has given their permission to use their project as source material for this thesis.

The work is done retrospectively by first revising theories behind the development and then adapting the design, execution, problems and testing of the software development of the project. The theory consists of five sections: .NET and .NET compact frameworks, C# programming language, concurrent programming, language interoperability and TAPI.

The project done for Luovaliike Oy was a success and the result was implemented in their HOTI Mobile Tool application. Therefore the software discussed in this thesis is also a valid example for designing and developing this kind of software.

Language of Thesis	Finnish
Keywords	TAPI, Windows Mobile, .NET Compact framework
Deposited at	☐ Electronic library Theseus ☐ Library of Kajaani University of Applied Sciences

SISÄLLYS

SYMBOLILUETTELO

API	Application Programming Interface eli ohjelmointirajapinta. Ulkoinen kirjasto jota käytetään täydentämään omaa sovellusta.
Asiakas	Jonkin palvelun tai toiminnon käyttäjä. Voi olla ihminen tai kone.
Asynkroninen	Toiminnon suoritus samanaikaisesti muusta ajosta riippumatta.
Callback	Funktio, jota ulkopuolinen ohjelma voi kutsua.
COM	Component Object Model. Microsoftin kehittämä kieliriippumaton rajapinta standardi.
Järjestely	kahden ohjelman tai rajapinnan välissä tiedonsiirrossa tehtävä operaatio, jolla välitettävä tieto muunnetaan vastaanottajan ymmärtämään muotoon.
Kahva	Erikoistunut muuttumaton osoitin, jolla voidaan globaalisti viitata luotuihin olioihin. Kahvaa ei viittaa muistiin, vaan on järjestelmän ylläpitämä kertaluonteinen tunnus.
Nimiavaruus	Kuvitteellinen alue tai ryhmä, jonka alaisuuteen koodielementtejä voidaan sijoittaa ja myöhemmin viitata staattisesti.
Offset	Yksittäisen arvon tai muuttujan sijainti lukujonossa suhteessa alkupisteeseen.
Osoitin	Muistiosoitteen sisältävä muuttuja. Osoittimella on tietotyyppi, jonka kokoista muistialuetta se osoittaa.
Prosessori	Tietokoneen keskeisin osa, joka ohjaa kaikkea sen toimintaa.
Rajapinta	Eri toimijoiden välinen raja-alue, jonka kautta ne ovat vuorovaikutuksessa. Alue kahden järjestelmän välillä, jonka ylittäessään tieto siirtyy järjestelmästä toiseen.

Semafori, ohjain	Käytetään myös termiä lukko, lock. Monisäikeisissä ohjelmissa käytettävä ohjainfunktio, joka estää ulkopuolisia säikeitä käyttämästä tiettyä muuttujaa toisen säikeen käyttäessä sitä. Usea säie voi kuitenkin jakaa saman semaforin.
Säie	Rinnakkaisohjelmoinnissa käytettävä termi järjestelmän pienimmälle toimijalle.
Wrapper	Yhden tai useamman ulkoisen ohjelmointirajapinnan tai kirjaston yhteen liittävä ja sovittava ohjelma. Käytetään yksinkertaistamaan, mukauttamaan tai mahdollistamaan ulkoisen kirjaston käyttö.

1 JOHDANTO

Pelkällä matkapuhelinlaitteella ei pysty soittamaan. Vaikka laite sisältäisi kaikki puheluiden soittamiseen tarvittavat osat, se tarvitsee ohjelmiston laiteen hallintaan ja muun muassa puheluiden soittamiseen. Soittaminen on kuitenkin puhelimen perusominaisuuksia, joten sitä harvemmin tulee ajateltua tai tiedostettua. Lisäksi soittaminen kuuluu puhelimen jo olemassa olevaan käyttöjärjestelmään, joten sitä harvemmin tarvitsee ohjelmistokehittäjänkään enää miettiä. Tämän työn kohdalla asia ei kuitenkaan ole näin, vaan mobiililaitteelle suunniteltuun isäntäsovellukseen halutaan omat mukautetut puhelintoiminnot.

Kehitystyön tavoitteena on tuottaa puhelinsovellus, jota voidaan käyttää osana olemassaolevaa sovellusta. Työni perustuu vuonna 2009 tehtyyn työharjoitteluuni Luovaliike Oy:ssä, jossa toteutin vastaavanlainen projekti. Projektin tavoite oli tuottaa korvaava sovellus Windows Mobile- käyttöjärjestelmän oletussovelluksen tilalle ja yhtenäistää puhelintoimintojen käyttöliittymä isäntäsovelluksen muun käyttöliittymän kanssa. Luovaliike Oy:ssä käytettiin Visual Studio 2008 kehitysympäristöä, joka tarjoaa hyvät työkalut Windows Mobile- kehitykseen, joten kyseistä ympäristöä käytettiin myös tämän työn toteutukseen.

Windows Mobile on Microsoftin kehittämä käyttöjärjestelmä älypuhelimille. Se perustuu aikaisempaan Windows Embedded CE käyttöjärjestelmään ja näki päivänvalon vuonna 2003 nimellä Windows Mobile 2003. Sen jälkeen käyttöjärjestelmästä on julkaistu useita versio, joista uusin on Windows Phone 7.

Luovaliike Oy on kehittänyt potilastietojärjestelmän nimeltään HOTI. Yksi järjestelmän osasovellus on mobiililaitteille suunnattu HOTI Mobile Tool. Tämä sovellus on alunperin kehitetty Windows Mobile 5- järjestelmälle ja tätä kautta määräytyi myös kehitystyön kohdejärjestelmä. Projektin aikana yrityksessä mietittiin myös siirtymistä Windows Mobile 6:n käyttöön, mutta projektiini tällä ei ollut vaikutusta.

Yritys halusi käyttöönsä C++ taitoisen ohjelmoijan, joka voisi toteuttaa tämän projektin. Minulla oli kokemusta C, C++ ja C# kielestä sekä .NET kehitysympäristöstä työpöytäsovelluksissa, mutta en ollut aikaisemmin tehnyt ohjelmia mitään mobiililaitteille. Projektin haasteet siis keskittyivät mobiilikehityksen erikoisuuksiin sekä Windows Mobile- järjestelmässä käytettävään .NET Compact Framework ympäristöön. Aikaisempi kokemus antoi eväät oppimiselle, mutta joissakin tilanteissa tutut ratkaisut eivät toimineet .NET Compactissa. Oman

haasteensa kehitykseen toi sovelluksen testaus. Vaihtoehtoina oli testata sovellusta emuloimalla puhelinlaite pöytäkoneella tai ajaa sovellusta konkreettisella laitteella telakassa tai irtonaisena. Joka tilanteessa sovellus toimi hieman eri tavalla eikä testaustyökalut olleet aina täysin käytettävissä.

Työhön liittyvä teoriaosuus käsittelee asioita niiltä osin kuin olen katsonut tarpeelliseksi. Teksti ei pyri täysin selittämään aiheita vaan esittelemään perustan ja kattamaan kehitystyötä koskevat alueet. Pois on jätetty esimerkiksi kehitystyön kohdealusta Windows Mobile, koska itse järjestelmän erikoispiirteet eivät merkittävästi vaikuttanut kehitystyöhön, ja Luovaliike Oy:n kehittämä HOTI ja sen liitännäiset.. Tärkeimmät osa-alueet kehitystyön osalta ovat Telephony Application Programming Interface ja .NET- kehysympäristö.

2 .NET FRAMEWORK KEHYSYMPÄRISTÖ

.NET, lausutaan dotnet, on Microsoftin kehittämä kehysympäristö. .NET:in tavoite on korvata Microsoftin aikaisemmat ohjelmointiteknologiat kuten MFC, ASP ja ATL, yhtenäistää sovelluskehitys käyttämään yhtä alustaa ja tarjota kehittäjälle helppo ja mukava kehitysympäristö. .NET:in avulla on mahdollista kehittää sovelluksia mille tahansa .NET alustalle, millä tahansa .NET kielellä. Kehyssympäristö koostuu ajonaikaisesta ympäristöstä, Common Language Runtime eli CLR, ja luokkakirjastosta, Framework Class Library eli FCL. (CLR via C#, Jeffery Richter. 2006, Microsoft Press.)

CLR hallitsee muistia, sovellusten ajoa ja muita järjestelmän toimintoja. CLR sisältää myös yleisen tyyppimäärittelyn, Common Type System eli CTS, ja yleisen kielen määrittely, Common Language Specification eli CLS. CTS antaa pohjan kaikille sovelluksissa käytettäville tyypeille ja CLS määrää millaisia ominaisuuksia oliot voivat itsestään paljastaa, jotta niiden toiminta kaikissa CLR:ssä voidaan taata. FCL on oliosuuntautunut kokoelma työkaluja ja rajapintoja jotka ovat yhteisiä kaikille .NET sovelluksille. Kokoelmaan kuuluu muun muassa tiedostojen ja verkon käyttö, säikeiden ohjaus, piirtotoiminnot ja algoritmeja. CLR siis tarjoaa puitteet ja ehdot ohjelmille ja hallitsee järjestelmää. FCL tarjoaa ohjelmoijalle työkalut CLR:n käyttöön. (CLR via C#, Jeffery Richter. 2006, Microsoft Press.)

Ennen .NET:iä Windows- järjestelmän eri toimintoja käytettiin COM-objektien ja dynaamisten kirjastojen avulla. .NET:ssä nämä kaikki toiminnot löytyvät FCL:stä ja ovat näin osa .NET sovellusta. COM-objektien poistumisen myötä myös monia niihin liittyviä käsitteitä, kuten rekisteri, GUID, AddRef ja Release, ei enää tarvita. (CLR via C#, Jeffery Richter. s. XXIII)

.NET:iä käyttäessä ohjelmaa ei kehitä varsinaisesti millekään laitteelle tai järjestelmälle vaan CLR:lle. Yleensä kääntäjät tuottavat tukemastaan kielestä vain yhden järjestelmän tukemaa konekieltä, esimerkiksi C++ vaatii eri kääntäjän Windows ja Linux käännöksille vaikka lähdekoodi olisi sama. .NET kääntäjä puolestaan tuottaa välikieltä, Intermediate Language eli IL, jonka CLR kääntää tarvittaessa konekieleksi. Kääntämiseen käytetään prosessoriarkkitehtuurikohtaista tarvekääntäjää, Just-In-Time compiler eli JIT. .NET ei siis poista tarvetta

kääntää ohjelmaa konekielelle, vaan jakaa käännöstyön kahteen osaan. (CLR via C#, Jeffery Richter. 2006, Microsoft Press.)

Microsoft itse on kehittänyt .NET kääntäjiä C++/CLI, C#, Visual Basic, Jscript ja J# kielille. Näiden ohella muut yhtiöt ja organisaatiot ovat kehittäneet .NET kääntäjiä muun muassa Ada, COBOL, Eiffel, LISP, Pascal, Perl, PHP ja Python kielille. Kaikki nämä kääntäjät voivat myös tulkita välikieltä. (Richter 2006.)

Käytettävästä kielestä riippumatta .NET kääntäjät tuottavat hallitun moduulin, joka tallennetaan Microsoft Windows portable executable eli PE tiedostoksi. Moduulista voidaan tehdä joko 32- tai 64- bittinen ja se sisältää PE otsikon, CLR otsikon, metadataa ja välikieltä. Otsikot sisältävät määrityksiä itse ohjelmasta. Metadata koostuu kahdentyyppisistä tauluista: kuvauksia lähdekoodin käyttämistä ja lähdekoodin viittaamista tyypeistä ja jäsenistä. Välikieli on lähdekoodista käännetyt varsinaiset käskyt, joita JIT puolestaan kääntää prosessorin suoritettavaksi. (Richter 2006.)

Yhteisen tyyppimäärittelyn avulla .NET kääntäjät luovat yhtenäistä välikieltä ja yhteisen välikielen ansiosta kaikki .NET sovellukset voivat viitata kaikkiin .NET sovelluksiin ja kaikki CLR:t voivat ajaa kaikkia .NET sovelluksia. (Richter 2006.)

2.1 .NET Compact Framework

.NET Compact Framework, .NET CF, on Windows Mobilen ja Windows Embedded CE järjestelmille kehitetty sovelluskehys, jonka avulla nämä järjestelmät voivat ajaa hallittuja sovelluksia. Se on hyvin .NET:in työpöytäversion kaltainen. Sovelluskehysten arkkitehtuuri, ohjelmointimalli ja tyyppikirjasto ovat samoja. .NET CF on kuitenkin luokiltaan ja toiminnoiltaan suppeampi kuin .NET. Kehittäjälle siirtyminen on .NET:stä .NET CF:ään on helppoa, mutta voi aiheuttaa yllätyksiä jos jokin tuttu funktio tai luokka puuttuu .NET CF:stä. (MSDN.)

.NET CF koostuu mobiiliympäristöön optimoidusta CLR:stä, karsitusta .NET luokkakirjastosta ja muutamasta yksinomaan .NET CF:lle tarkoitetusta luokasta. .NET Compact Framework- ympäristöllä on myös oma mobiilijärjestelmille tarkoitettu tarvekääntäjä. (MSDN.)

3 C# OHJELMOINTIKIELI

C# on Microsoftin kehittämä ohjelmointikieli, joka korvaa Visual Basic kielen .NET- ohjelmien kehityksessä. Pohjana C#- kielessä on käytetty C- ja C++-kieliä, joista on otettu parhaat puolet mukaan ja jätetty monimutkaisia ominaisuuksia pois, jotta kieli olisi esikuviinsa verrattuna helpompi ja vähemmän virhealtis. Pääpaino C#- kielen kehityksessä on ollut käyttömukavuus ja helppous. (Wille 2001.)

C#:sta on poistettu kokonaan C++:ssa käytettävät makrot , mallit ja tyyppimäärittelyt. Näistä makrot ja tyyppimäärittelyt mahdollistavat lyhyemmän koodin kirjoittamista, mutta varsinaista tehohyötyä niistä ei ole ja ne ovat hyvin virhealttiita. Malleille puolestaan on käyttöä kuormituksessa ja monimuotoisuudessa, mutta voi olla hankala käyttää ja virhealtis. Virtaviivaistettuja ominaisuuksia ovat muun muassa moniperintä, muuttujien ja tyyppien viittausoperaattorit, osoittimet, muistinhallinta ja perustietotyypit. (Wille 2001.)

Siinä missä C++ -luokalla voi olla monta kantaluokkaa ja monta abstraktia kantaluokkaa, C#- luokalla voi olla yksi kantaluokka ja monta rajapintaa. Rajapinta on luokka, joka sisältää vain funktioiden, delegaattien ja tapahtumien esittelyjä. Abstraktin luokan ainoa rajoitus on, että siitä ei voi tehdä ilmentymää. Luokan periessä rajapinta, sen täytyy myös sisältää toteutus jokaiselle rajapinnan jäsenelle. Tietyn rajapinnan toteuttavia luokkia voidaan näin käsitellä samalla tavalla, aivan kuten C# tai C++:ssa voidaan käsitellä samoin tietyn luokan perineitä luokkia. Esimerkiksi rajapinta Tiekulkuneuvo sisältää funktion VaihdaRenkaat ja jokin algoritmi käsittelee Tiekulkuneuvo – tyyppisiä olioita. Tälle algoritmilla voidaan välittää mikä tahansa Tiekulkuneuvo- rajapinnan toteuttava luokka, kunhan sille tehdään ensin ulkoinen tyyppimuunnos Tiekulkuneuvo – tyyppiin. Algoritmi voi kutsua kaikkien sille välitettyjen olioiden VaihdaRenkaat – funktiota, koska niiden on pakko toteuttaa rajapinnan kaikki funktiot. (Wille 2001.)

C++:ssa käytetään monia eri operaattoreita viitattaessa erilaisten kokonaisuuksien sisältöön: Arvoviittausoperaattori piste, jolla viitataan ilmentymien sisältöön, nuoli (->) jolla viitataan osoittimen osoittaman muistialueen sisältöön ja tupla kaksoispisteet (::) jolla viitataan nimiavaruuksien sisältämiin staattisiin ominaisuuksiin. C#:ssa nämä kaikki on yhdistetty pisteeksi. (Wille 2001.)

Perusmuuttujatyyppejä on C kielissä monia. Näillä tarkoitetaan tavuissa eri pituisia muuttujia, joista kehittäjän täytyy olla perillä ja kussakin tilanteessa käyttää oikean kokoista muuttujaa. C#:ssa nämä kaikki eri pituudet on kätketty perustietotyyppien sisään. Kokonais- ja liukuluvut täytyy edelleenkin erotella ja merkitsemättömän, unsigned, tyypin käyttö ilmoittaa. (C#, Christopher Wille 2001.)

Kaikille C ja C++ ohjelmoinnille tyypilliset osoittimet ja muistinhallinta on myös piilotettu ja hallitussa kielessä korvattu automaattisella muistinhallinnalla ja roskienkerääjällä. Eri kielien yhteystoiminnan mahdollistamiseksi on kuitenkin mahdollista ottaa muistinhallinta ja osoittimet käyttöön. (Wille 2001.)

C# on ensisijaisesti tarkoitettu CLR:n käyttöön. Tämä ympäristö mahdollistaa kielen hallitut ominaisuudet, kuten muistinhallinnan ja roskien keräämisen. Oletusarvoisesti kaikki C# -kielellä kirjoitettu ohjelmakoodi on hallittua. Tämä tarkoittaa sitä, että koodissa ei voi olla mahdollisesti vaarallista koodia, kuten muistinhallintaa ja virheellisiä tyyppimuunnoksia. On kuitenkin mahdollista kirjoittaa tällaista koodia myös C# kielellä, jolloin koodia kirjoitetaan kuin C++ kieltä. (Wille 2001.)

3.1 Delegaatti

Delegaattien tehtävä on kapseloida funktio ja sen käyttötarkoitus on lähellä C++ -kielen funktio-osoitinta. Osoittimesta poiketen delegaatti on tyyppiturvallinen. Tämän vuoksi delegaatti voi osoittaa vain saman määrittelyn omaavaan funktioon. Delegaatit luodaan kahdessa vaiheessa. Ensin kehittäjä esittelee uuden delegaattityypin. Tästä tyypistä voidaan luoda ilmentymiä, joihin funktiot kiinnitetään. Delegaattityyppi esitellään esimerkiksi seuraavasti:

```
delegate int MyDelegate (int tunnus, string nimi);
```

Kuva 1 Delegaattityypin määrittely.

Näin luodaan delegaattityyppi, joka on nimeltään MyDelegate, se palauttaa kokonaisluvun ja se voi ottaa parametreiksi kokonaisluvun ja merkkijonon. Tähän delegaattiin voitaisiin kiinnittää vaikka seuraavanlainen funktio:

```
int MyFunction(int i, string ii)
```

Kuva 2 Kuvan 1 delegaatin mukainen funktio

Delegaattityyppiä voidaan esittelyn jälkeen käyttää kuten luokkaa. Muuttujaa luotaessa esitellään ensin, jonka jälkeen muuttujan nimi. Delegaattimuuttuja täytyy alustaa joko *null* arvoon tai sitoa jokin funktio heti muodostimessa. Funktiota voidaan myös sitoa jälkikäteen eri sijoitusoperaattoreilla. Mikäli delegaattiin halutaan sijoittaa jokin muuttuja ajettavien lohkojen ulkopuolella, täytyy sidottavan funktion olla staattinen. (MSDN.)

```
MyDelegate delegateTest = new MyDelegate(MyFunction);
```

Kuva 3 Funktion sidonta alustuksessa

```
MyDelegate delegateTest = null;
delegateTest = new MyDelegate(MyFunction);
```

Kuva 4 Kuvan sidonta sijoituksella.

Delegaattimuuttuja voidaan ajatella funktiona tästä eteenpäin. Siihen sidottua funktiota voidaan kutsua normaalin funktiokutsuntapaan. Delegaattimuuttujaan voidaan liittää myös useita funktioita ja C#:ssa delegaatit ovat kykeneviä monikutsuun. Delegaatti pitää listaa siihen kiinnitetyistä funktioista ja tavallisella funktiokutsulla delegaattimuuttuja kutsuu kaikkia siihen liitettyjä funktioita. Delegaattimuuttuja toimii myös taulukkona, jonka ansiosta siihen kiinnitettyjä funktioita voidaan käyttää yksitellen, mikäli tiedetään funktion indeksi taulukossa. (Akadia AG 2002.)

4 RINNAKKAISOHJELMOINTI

Rinnakkaisohjelmointi on ohjelman kehittämistä siten, että se suorittaa eri tehtäviä rinnakkaisajona. Rinnakkaisajo on tehtävien näennäisesti samanaikaista suorittamista. Sen vastakohta on lineaarinen ajo, jossa ohjelman tehtävät suoritetaan järjestyksessä, uuden tehtävän aloitus odottaa edellisen tehtävän valmistumista. Samanaikaisuus on näennäistä koska prosessoriydin voi suorittaa vain yhtä laskua kerrallaan. Nykyään moniytimiset prosessorit ovat yleisiä ja aito samanaikainen suoritus voidaan saavuttaa jakamalla tehtävät usealle prosessoriytimelle, mutta tällöinkin tehtävien määrä saa olla maksimissaan prosessoriytimien määrä. Pelkkä käyttöjärjestelmä pitää yllä monta kymmentä ohjelmaa. Tätä kirjoittaessa Windowsin tehtävienhallinta näyttää prosessien lukumääräksi 90 ja säikeiden lukumääräksi noin 1000. Yksinkertaisuuden vuoksi kuitenkin tästä eteenpäin käytetään pelkkää samanaikaisuus- termiä. Rinnakkaisohjelmointiin liittyvät myös termit yksi- ja monisäikeisyys, joista myöhemmin lisää. (Clifton 2008.)

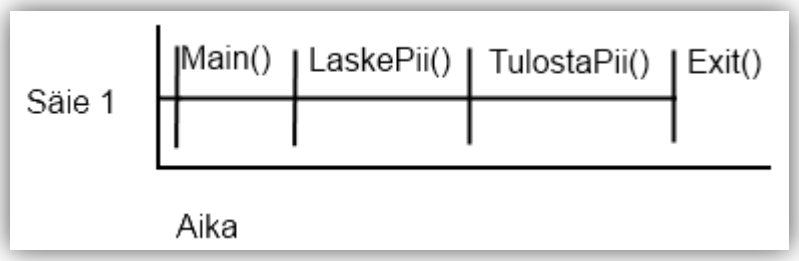
Rinnakkaisuus luodaan aikaviipaloinnilla, jolloin prosessoriytimen suoritusaika pilkotaan osiin ja järjestelmä jakaa nämä aikaviipaleet eri tehtävien kesken. Yksi viipale käytetään tietojen lukemiseen ja toinen käyttöliittymää piirtämiseen. Riittävällä nopeudella tämä luo käyttäjälle samanaikaisuuden illuusion. Lyhyet ja yksinkertaiset ohjelmat näyttävät samanaikaiselta joka tapauksessa, mutta aikaviipaloinnilla suuretkin sovelluksen saadaan näyttämään samanaikaisilta. Rinnakkaisajoa on myös hajautettu laskenta, jossa tehtävät jaetaan aikaviipaleiden sijaan eri järjestelmille jonkin viestiyhteyden välityksellä, esimerkiksi Internetin. (Clifton 2008.)

Rinnakkaisajo ei pelkällä olemassaolollaan paranna suoritustehoa. Pitkään tietokoneiden suorituskyky parani tuplaamalla prosessorin kellotaajuus kahden vuoden välein. Tämä ei enää nykypäivänä onnistu vaan teholisäys saavutetaan kasvattamalla prosessoriytimien määrää. Rinnakkaisajon avulla prosessien tehtävät saadaan jaettua eri prosessoreille ja näin ohjelman suoritus nopeutuu. (Ignatchenko 2010 a.)

Säikeet

Rinnakkaisuus saadaan aikaan käyttämällä useita säikeitä. Säie on ohjelman sisäinen toimija, joita ohjelmalla on aina ainakin yksi, ensisijainen säie. Tämä ensisijainen säie suorittaa ohjelman kaikki tehtävät ja tällöin ohjelma on yksisäikeinen. Halutessaan ohjelmoija voi määrätä ohjelman luomaan lisää säikeitä tehtävien suorittamiseen, jolloin ohjelmasta tulee monisäikeinen. Monisäikeisessä ohjelmassa kaikki sen säikeet käyttävät samoja resursseja, kuten ohjelman muistialuetta, jonka ansiosta ohjelman säikeet löytävät saman tiedon saman muistiosoitteesta. Säikeet ovat toimivat pääasiassa toisistaan riippumattomia, jonka ansiosta ne voivat jatkaa toimintaansa, vaikka jollakin toisella säikeellä kestäisi pitkään tehtävänsä suorittamiseen. Säikeet ovat kuitenkin yhteydessä luojasäikeeseensä ja luomiinsa säikeisiin. Ympäristöstä ja ohjelmointikielestä riippuen säikeiden elinkaari on riippuvainen luojasäikeestä. C# kielessä sammutuskäskyn saanut säie lähettää sammutus käskyn luomilleen säikeille, odottaa näiden sammumista ja sammuu vasta näiden jälkeen. C++ kielessä säie sammuu itse välittömästi ja lähettää sammutuskäskyt luomilleen säikeille, mutta ei välitä näiden tilasta. (Wille 2001.)

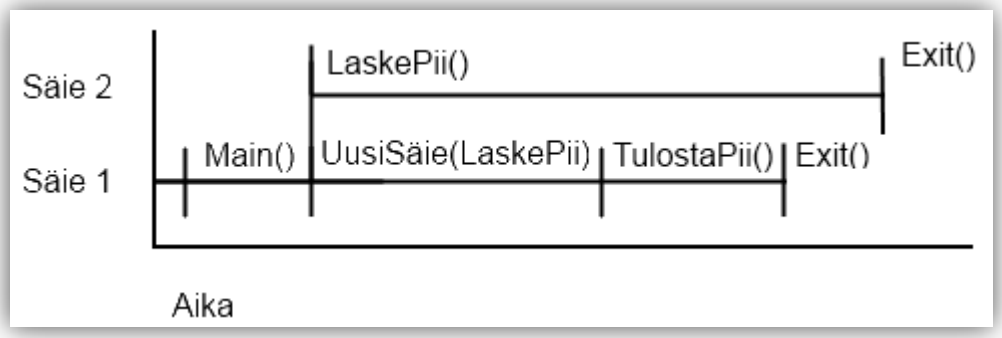
Ajatellaan yksinkertainen sovellus, jonka tehtävä on laskea piin likiarvo, tulostaa likiarvo näytölle ja sammua. Ohjelma alkaa Main- funktiosta, joka kutsuu ensin LaskePii- funktiota, jota seuraa tulostusfunktio TulostaPii ja ohjelma loppuu Exit- funktioon. Suorituksen kulku yhdellä säikeellä ajettaessa näyttäisi seuraavalta.



Kuva 5 Yksisäikeinen sovellus

Samanaikaisuus saavutetaan luomalla ohjelman eri tehtäville uusia säikeitä. Rinnakkaisajon alussa selitettiin aikaviipaloinnin käsite. Säikeet ovat niitä tekijöitä, jotka käyttävät aikaviipaloinnilla luotuja aikaviipaleita. Prosessori siis antaa pienen osan ajastaan jokaiselle säikeelle. (Liberty 2001.)

Otetaan uudelleen tutkinnan alle piinlaskentasovellus, mutta tällä kertaa säie 1 luo LaskePii-funktion suoritukseen oman säikeen. LaskePii tallentaa tuloksen globaaliin muuttujaan, jonka sisältämän arvon TulostaPii tulostaa. Suoritus voisi kulkea kuvan 6 mukaisesti.



Kuva 6 Kaksisäikeinen sovellus

Kuva 6 tarkoituksella esittää LaskePii- funktion suorituksen kestävän kauemmin kuin säie 1:n suorituksen. Ohjelmassa on muutama ongelma: TulostaPii ei voi tulostaa arvoa, koska sitä ei ole vielä saatu laskettua tai tallennettua ja isäsäikeen sammuessa myös sen lapsisäikeet saattavat sammua. Seuraavassa luvussa käsitellään miten näitä ongelmia voidaan kiertää.

4.1 Edut ja ongelmat

Monisäikeisyyden selkein etu yksisäikeisyyteen on sen tarjoama tehtävien rinnakkaistaminen. Rinnakkaisuuden avulla ohjelmista voidaan tehdä sujuvampia ja käytettävämpiä. Vuorovaikutusta käyttäjän ja ohjelman välillä ei estetä isoilla ja aikaa vievillä tehtävillä, vaan ne siirretään omiin säikeisiinsä muun suorituksen taustalle. Usea säie myös mahdollistaa useamman pro-

sessorin käytön, jolloin järjestelmän kaikki resurssit saadaan käyttöön. Tästä on erityisesti hyötyä paljon laskentatehoa vaativissa tehtävissä. (Varela 2007)

Esimerkiksi olio A odottaa käyttäjän syötettä kuten hiiren liikkeitä tai näppäimistön painalluksia ja olio B lukee tiedostoja kiintolevyltä. Ohjelmaa käyttäessään käyttäjä antaa syötteen, joka käskyttää ohjelmaa avaa tiedoston. Yhdellä säikeellä ajettaessa olio B pitää suorittaa tiedoston luku kokonaisuudessaan ennen kuin olio A voi jatkaa syötteiden lukuun. Käyttäjän syötteitä ei käsiteltäisi ennen kuin luku on saatu loppuun. Säikeillä tiedoston luku voidaan pilkkoa, jolloin haetun tiedoston alkuosa saadaan esitettyä nopeasti ja loput tiedot voidaan ladata esiteltyjä tietoja käsitellessä. Esimerkiksi monisivuinen dokumentti ladataan sivu kerrallaan ja uusia sivuja näytetään sitä mukaan kuin niitä saadaan ladattua. (Liberty 2001.)

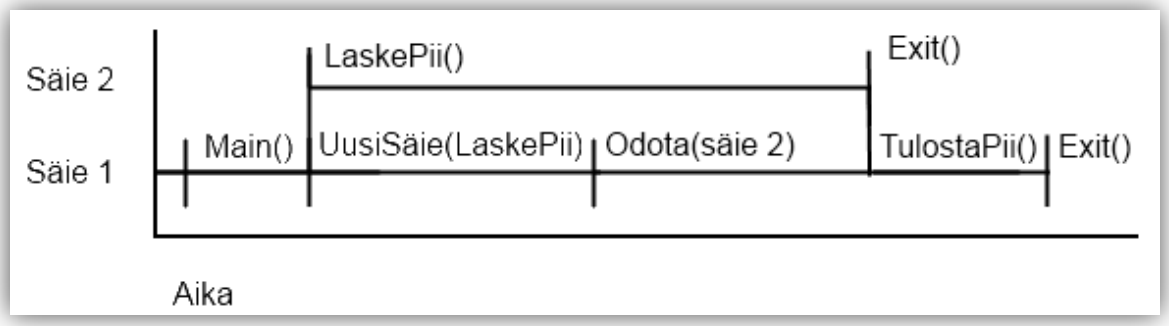
Tämä säikeiden kyky toimia itsenäisesti on kuitenkin se syy ensimmäiseen rinnakkaisohjelmoinnin ongelmaan: Miten taata ohjelman oikea suoritus kun se jaetaan usean toisistaan riippumattoman tekijän kesken. Yksisäikeinen ohjelma ajetaan lineaarisesti alusta loppuun, siinä järjestyksessä kun ne ovat lähdekoodiin kirjoitettu. Monisäikeinen ohjelma luo tehtävälle uuden säikeen, jättää tehtävän tämän säikeen hoidettavaksi ja jatkaa seuraavaan tehtävään. Jos tämä seuraava tehtävä on mitenkään riippuvainen aikaisemmasta tehtävästä, ohjelma käyttäytyy odottamattomasti. (Varela 2007.)

Vaikka säikeet toimivatkin itsenäisesti, niin erikoisesti seuraava ongelma on säikeiden ominaisuus käyttää samaa muistialuetta. Hallitsematon tiedon luku ja tallentaminen johtaa tiedon korruptoitumisen. Tieto korruptoituu, kun säie lukee tai tallentaa sen samaan aikaan kun jokin toinen säie käyttää samaa tietoa. Esimerkiksi Säie T1 lukee kokonaisluvun ja alkaa muokata sitä. Muokkauksen aikana säie T2 lukee saman kokonaisluvun. Säie T1 saa muutoksensa tehtyä ja tallentaa muuttujan. Nyt säie T2 sisältää version muuttujasta, jossa säikeen T1 tekemä muutos ei ole mukana. Kun säie T2 tallentaa kokonaisluvun osittain vanhoilla tiedoilla, säikeen T1 muutokset katoavat. Säikeet voisivat esimerkiksi kasvattaa muuttujan arvoa yhdellä. Molemmat lukevat alkuarvon yksi, lisäävät siihen ykkösen ja tallentavat arvon. Oikein suoritettuna lopputulos olisi kolme, mutta arvoksi tallentuu kaksi. (Liberty 2001.)

Säikeiden toimintaa ohjataan erilaisilla hallintarakenteilla kuten jonoilla, lukoilla, synkronoinnilla ja varauksilla. Keskeinen ajatusmalli kaikissa on jotenkin rajata muistin käyttöä jonkun

säikeen tehdessä siihen muutoksia. Hyvällä suunnittelulla säikeet saadaan toimimaan halutulla tavalla eikä tieto korruptoidu (Clifton 2008.)

Otetaan taas tarkkailuun piinlaskija. Tällä kertaa ohjelmaan on lisätty hallintarakenne, joka käskee säie 1:den odottamaan säie 2:ta, jonka jälkeen se voi tulostaa laskun tuloksen ja sammua itsekin.



Kuva 7 Hallittu monisäikeinen sovellus

Hallintarakenteista kuitenkin seuraa uusia ongelmia: Säie voi nääntyä tai säikeet voivat päätyä erilaisiin umpikujiin. Nääntyessä säie ei saa tarvittavia resursseja toimintansa jatkamiseen. Umpikujissa säikeet lukitsevat tai vapauttavat resursseja ristiin. Umpikuja voi olla joko deadlock tai livelock. Deadlock- umpikujassa säikeet jäävät odottamaan toisen säikeen varaaman muuttujaa vapauttamatta itse lukitsemaansa muuttujaa, jonka vapautumista toinen säie odottaa. Livelock- umpikujassa säikeet jatkavat toimintaansa, mutta toistavat ristiin toimintoja, jossa ne aiheuttavat saman toiminnon toiston. (Hall 2008.)

Monisäikeisiin ohjelmiin liittyy läheisesti myös arvaamattomuus, vaikka ne suunniteltaisiinkin hyvin. Mitä enemmän ohjelmassa on säikeitä, sen useampi hämmentäjä sopassa on ja suurempi potentiaali arvaamattomalle käytökselle. Useita kymmeniä säikeitä käyttävässä ohjelmassa ohjelman kaatava virhe voi vaatia hyvinkin tarkkoja ehtoja jotka täyttyvät vain yhdellä kerralla miljoonasta. Tällaisten virheiden jäljittäminen on hyvin hankalaa, ellei mahdotonta.(Ignatchenko 2010 a.)

5 KIELTENVÄLINEN YHTEISTOIMINTA

Yhteistoiminta tarkoittaa erikielisien kirjastojen, moduulien tai sovellusten liittämistä toisiinsa. Tämä sisältää viestintäprotokollat, laitteistot, ohjelmat, sovellukset ja tiedon yhteensopivuuden. Esimerkkejä yhteistoiminnasta on ohjelmistorajapinnan käyttö osana sovellusta tai sähköpostin välitys kahden eri palvelinsovelluksen välillä. Tälle työlle keskeistä yhteistoimintaa on kieltenvälinen yhteistoiminta: C- kielisen TAPI kirjaston käyttö C#- kielisessä sovelluksessa. (Madiajagan & Vijayakumar 2006.)

Syitä ulkopuolisten kirjaston käyttöön ohjelmistokehityksessä ovat kehitysajan lyhentäminen, oman tietotaidon paikkaaminen tai korvaaminen, toisen kielen tehokkuus ja vanhojen ratkaisujen uudelleenkäyttö. Joskus kehittäjien ei ole edes mahdollista kirjoittaa itse joitakin toimintoja, vaan heidän täytyy käyttää ennalta määrättyä kirjastoa. Hyvä esimerkki tästä on tämän työn aihe, puhelintoiminnot. (Madiajagan & Vijayakumar 2006.)

5.1 Yhteistoiminta .NET ympäristössä

.NET:ssä on kaksi eri tapaa yhteistoimintaan: COM-objekteja käytettäessä COM-yhteistoiminta ja yleiskäyttöinen System.Runtime.Interop nimiavaruus. COM- yhteistoiminnalla COM- objektit saadaan toimimaan .NET sovelluksessa kuten .NET- objektit ja Interop- palveluilla voidaan tuoda muiden kielien funktioita .NET sovelluksiin. Interop on kuitenkin rajallinen eikä sillä voi tuoda esimerkiksi ulkopuolisen kirjaston olioita. (MSDN)

COM- yhteistoiminta on automaattista ja se tapahtuu lisäämällä COM- objekti .NET- projektin viittauksiin, jolloin .NET luo COM Callable Wrapper- luokan, CCW, jonka avulla .NET- sovellus osaa käyttää objektin tyyppejä ja funktioita. Tämän jälkeen COM- objekti on kuin mikä tahansa .NET- objekti. (MSDN.)

Interop palveluiden ydin on DllImportAttribute- määrite. Tämä määrite lisätään funktioiden eteen esittelyn yhteydessä, jolloin kääntäjä osaa etsiä funktion rungon määritteen osoittamasta kirjastosta eikä .NET tiedostosta. Funktiota ei siis varsinaisesti tuoda .NET- ohjelmaan vaan .NET- ohjelman funktion kerrotaan käyttävän ulkoista runkoa. Minimissään määritteel-

le pitää antaa käytettävä lähdekirjasto, jolloin kirjastosta haetaan samannimistä funktiota. Määritteen lisäksi .NET:in funktio täytyy esitellä staattiseksi ja ulkoiseksi, static extern. Funktion palautusarvon ja parametrien tyyppejä määritellessä täytyy huomioida mahdollinen järjestelyn tarve. Interop tekee oletusjärjestelyt perustyypeille, mutta poikkeavat tyyppimäärittelyt täytyy kertoa erillisellä määritteellä muuttujakohtaisesti. Lopuksi täytyy varmistaa käytettävän kirjaston sijainnin vastaavan määritteen polkua. Kirjaston oletetaan sijaitsevan samassa polussa ohjelman kanssa jos kirjastosta on annettu pelkkä tiedostonimi. DllImportAttribute-määrite on aina funktiokohtainen ja se täytyy määrittää jokaiselle tuotavalle funktiolla erikseen. (MSDN.)

```
[DllImport("sampleDynamicLibrary.dll")]
static extern void sampleFunction();
```

Kuva 8 Yksinkertaisin mahdollinen funktion tuonti C#:ssa

DllImportAttribute- määritettä voidaan myös tarkentaa muutamalla lisäkentällä, jotka parantavat koodin luettavuutta tai muokkaa kutsun toimintaa. Lisäkenttiä ovat muun muassa EntryPoint ja ThrowOnUnmappableChar. EntryPoint- parametrilla voidaan määrittää haettavan funktion nimi kirjastossa, jolloin funktion voidaan nimetä uudelleen .NET:ssä, esimerkiksi ohjelmoijan omaan nimeämiskäytäntöön sopivaksi. ThrowOnUnmappableChar- kenttällä voidaan totuusarvoisesti määrätä heittääkö ohjelma vihreen, jos määritteessä annetun unicode-merkistöä käyttävän merkkijonon jotain merkkiä ei voida tulkita ja se kääntyy ANSI-merkistön '?'- merkiksi. (MSDN.)

Yhteistoiminta Compact Framework ympäristössä

.NET:in työpöytäversioon verrattuna CF - ympäristön yhteistoiminta ei ole aina ollut yhtä laaja. CF versio 1.0 tukee vain Platform Invoke- menetelmää, jolla voidaan kutsua natiivia koodia. Tällöin COM- objektien käyttö pitää toteuttaa räätälöidyn wrapperin ja Platform Invoke avulla. Nämä wrapperit ovat dynaamisia kirjastoja, jotka kutsuvat COM- kirjaston funktioita. Compact puolestaan kutsuu wrapperin funktioita platform invoken kautta. .NET CF:n versiosta 2.0 lähtien myös COM yhteistoiminta on ollut mahdollista. (MSDN.)

5.2 Rajoitukset

Yleensä ulkoisista ohjelmista tuodaan vain välttämättömät funktiot. Ohjelma voi kuitenkin sisältää omalle sovellukselle hyödyllisiä jäseniä.. .NET- projekteihin COM- objektit voidaan tuoda kokonaisuudessaan, mutta C tai C++- kielisistä kirjastoista voidaan tuoda vain funktioita. Muut räätälöidyt rakenteet, kuten tietueet, täytyy uudelleenkirjoittaa C#- kielen syntaksin mukaisesti. (MSDN.)

Yhteistoiminta voi myös vaikuttaa sovelluksen tehovaatimuksiin, sekä positiivisesti että negatiivisesti. Jotkin ohjelmointikielet ovat lähempänä konekieltä kuin toiset jonka ansiosta näillä kielillä tehdyt ohjelmat ovat nopeampia suorittaa. Tällaista matalamman tason kieltä käytettäessä voidaan nopeuttaa ohjelman suoritusta. Vastaavasti korkeamman kielen käyttö vai hidastaa ohjelman suoritusta. Tietotyyppien eroilla on myös negatiivinen vaikutus suoritusnopeuteen. Kaikki siirtyvä data pitää järjestellä vastaanottavaan kieleen sopivaksi. Mitä monimutkaisempi tietotyyppi, sen kauemmin sen järjestelyyn menee. Yksinkertaiset perustietotyypit, kuten kokonaisluku, eivät vaadi merkittävästi järjestelyä, mutta suuret kokonaisuudet kuten pitkät merkkijonot, tietueet tai luokat vievät aikaa. (MSDN.)

5.3 Callback- funktiot

Yhteistoimintaan liittyy tiiviisti callback- funktioiden käyttö. Kirjastolla voi olla tarve välittää useita arvoja kerralla tai sen omalla tahdilla eikä vain asiakasohjelman pyytäessä. Ilman mitään mekanismia kirjasto voi viestiä takaisin vain funktioiden palautusarvoilla. Yksinkertainen ratkaisu on callback- funktio. Ne mahdollistavat kaksisuuntaisen yhteyden kirjaston ja asiakasohjelman välillä antamalla asiakasohjelmalta funktion kirjaston käyttöön.

Kirjasto määrittää funktio-osoittimen, jonka on määrä osoittaa ulkoiseen funktioon. Asiakasohjelma esittelee ja toteuttaa kirjaston funktio-osoitinta vastaavan funktion ja välittää kirjastolle osoittimen toteutettuun funktioon. Nyt kirjasto voi käyttää funktio-osoitinta asiakasohjelman funktion kutsumiseen. Tämä asiakasohjelman funktio on callback- funktio

```
typedef void (CALLBACK *CALLBACKINTERFACE)(
            LPWSTR strParam,
            int intParam1,
            int intparam2);
```

Kuva 9 Callback- funktio-osoitintyypin määrittely

Funktio-osoitintyypin määrittely vastaa pitkälti normaalin funktion esittelyä. Typedef tarkoittaa tyyppimäärittelyä, void on funktion palautusarvoa, CALLBACK on windows.h otsikkotiedostossa esitelty makro, '*' tarkoittaa osoitinta, CALLBACKINTERFACE on osoitintyypin nimi ja lopuksi suluissa on funktion parametrilista. CALLBACK *CALLBACKINTERFACE pitää olla suluissa jotta kääntäjä tajuaa typedefin viittaavan funktio-osoittimen määrittelyyn eikä CALLBACKINTERFACE osoittimen määrittelyä CALLBACK sanaan ja lopun olevan syntaksivirhettä. (MSDN.)

```
void callbackFunction(
    [MarshalAs(UnmanagedType.LPWStr)]
    string strParam,
    int intParam1,
    int intParam2
    );
```

Kuva 10 kuvan 9 callback- tyyppiä vastaava callback- funktio.

```
CallbackDelegate Callbackinstance = new CallbackDelegate(sampleFunction);
```

Kuva 11 callback- funktion sidonta delegaattiin.

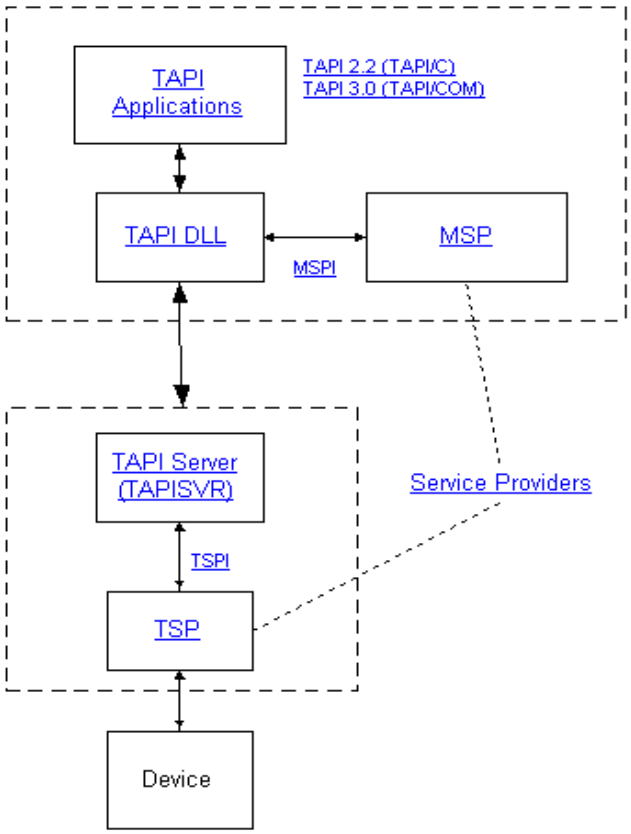
C# ei tue funktio-osoittimia eivätkä sen funktiot ole olioita, joten funktioita ei sellaisenaan voida välittää muihin ohjelmiin. Tätä varten delegaatit ovat olemassa. Delegaatti kirjoitetaan vastaamaan välitettävää funktiota, funktio sidotaan delegaatin ilmentymään ja ilmentymä välitetään kirjastolle.

6 OHJELMOINTIRAJAPINTA TAPI

Telephony Application Programming Interface eli TAPI on ohjelmointirajapinta Windows ympäristöihin ja mahdollistaa erilaisten yhteyksien, kuten puhelinverkon, käytön ohjelmistossa. TAPI:n avulla on mahdollista käyttää seuraavanlaisia toimintoja:

1. perinteisten puheluiden soittamisen ja vastaanottamisen
2. puhelinkeskuksen, joka seuraa useita toimijoita
3. modeemin hallinta

TAPI erottaa yhteyksien hallinnan laitteen hallinnasta. Näin ohjelmistoa ja laitteistoa ei tarvitse räätälöidä toisilleen ja niitä voidaan kehittää erillään. (Microsoft Telephony Programming Model, MSDN.)



Kuva 12 abstraktio TAPI:n rakenteesta

TAPI:n osat jakautuvat seuraavasti: TAPI sovellus tietää käyttäjän tarpeet, TAPI DLL ja TAPISVR ymmärtää yleisen viestinnän ja palveluntarjoajat, Telephony Service Provider ja Media Service Provider osaavat laitehallinnan. (Microsoft Telephony Programming Model, MSDN.)

TAPI:n eri versioista

TAPI:sta on käytössä kaksi versiota: TAPI2 ja TAPI3. TAPI2 on C- kielellä kirjoitettu kirjasto, jonka tuki alkaa Windows 95:stä ja ulottuu uusimpiin Windows versioihin. Viimeisin päivitetty versio on 2.2. TAPI3 on uudempi, COM-pohjainen kirjasto, jota tukevat palvelin versiot Windows Server 2003 lähtien ja työpöytä versiot Windows 2000 lähtien. Windows Mobile käyttää TAPI 2.0 aliversio, jota on paranneltu joillakin 2.1 osilla. TAPI:a käyttävä ohjelma voi itse määrittää alimman ja korkeimman tukemansa version ja vaatia ohjelmia muilta ohjelmilta tietyn version käyttöä. (MSDN.)

Kirjaston kehittyessä sen vaatimukset ovat myös kasvaneet ja kirjaston funktioiden on täytynyt laajentua. C- kielessä funktioiden kuormitus ei ole sallittua, joten uudet, laajemmat funktiot on täytynyt nimetä lisäämällä funktionimien loppuun Ex, Extended. (MSDN.)

TAPI:n eri versioissa on joitakin poikkeuksia. Esimerkiksi TAPI2 käyttää yhteyksien ja puheluiden hallintaan linjalaitteen linjoja. TAPI3:ssa linjat on korvattu osoitteilla. Tässä työssä keskitytään TAPI2 versioon. Kaikki TAPI2:lle sopivat ratkaisut eivät välttämättä sovi TAPI3:lle tai toimi siinä lainkaan.

6.1 Rakenne ja käyttöönotto

TAPI:a ohjataan sen funktioilla, joilla voidaan esimerkiksi käynnistää ja lopettaa sen toimintoja tai pyytää lisätietoja. Tapahtumista TAPI kertoo viestien avulla, joita on yhteensä neljä eri tyyppiä: Linjalaitteen viestit, puhelinlaitteen viestit, puhelinkeskuksen viestit ja virheviestit. TAPI:n dokumentaatio mainitsee myös käytöstä poistetut avustettujen puhelintoimintojen viestit. (MSDN.)

Jotkut funktiot vaativat useamman kutsun toimiakseen täysin tai vaativat jonkin toisen funktion herättämään viestin TAPI:lta, joka sisältää sen ajoon tarpeellista tietoa ja joitain funktioita ei ole mahdollista hyödyntää ennen tietyn laitteen tapahtumaa. Sujuvaan toimintaan vaaditaan siis varsin läheinen yhteistyö TAPI:n ja sitä käyttävän ohjelman välille. (MSDN.)

TAPI:a käyttäessä tulee huomata, että vain osa sen funktioista on asynkronisia. Funktion toiminta kannattaa aina tarkistaa TAPI:n dokumentaatiosta, joka löytyy Microsoft developer networkista, lyhenne MSDN. Tämä voi aiheuttaa virheitä jos asiakasohjelmassa eri säikeet kutsuvat useaa ei-asynkronista funktiota. TAPI:n funktiot eivät myöskään välitä aikatietoja. Tästä johtuen esimerkiksi puhelujen keston seuraaminen on kehittäjän harteilla. (MSDN.)

C/C++ ohjelmassa TAPI:n käyttö vaatii otsikkotiedoston liittämisen niihin tiedostoihin, joissa TAPI:a käytetään. Liitettävä otsikkotiedosto riippuu käytettävästä TAPI versiosta. TAPI2 käytettäessä riittää tapi.h, TAPI3 käytettäessä tapi3.h sekä mahdolliset laajennukset: tapi3cc.h, tapi3ds.h, tapi3err.h ja tapi3if.h. Näiden lisäksi täytyy varmistaa, että sovellus löytää käytettävään versioon sopivan kirjasto-tiedoston, 2.2 versiolla tapi.dll tai 3.0 versiolla tapi3.dll. Hyvä sijainti näille on sovelluksen oma polku. (MSDN.)

Muissa ohjelmointikielissä, kuten C#, joudutaan otsikkotiedostojen sijaan käyttämään kielen yhteistoiminta ominaisuuksia, jotta metodit saadaan tuotua TAPI:n tiedostoista. Yhteistoimintaan ei toimiakseen tarvitse kuin TAPI:n kirjastotiedostot, mutta otsikkotiedostot ovat hyvä apu ohjelmoinnissa, koska ne sisältävät TAPI:n kaikkien struktuurien, funktioiden, bittiflagien ynnä muiden määrittelyn. Tässä työssä käydään läpi miten tämä tehdään C# kielellä. Muissa kielissä TAPI oletettavasti toimii samalla periaatteella, mutta toteutus voi poiketa merkittävästi C# ratkaisuista. (MSDN.)

WIN32 API:lle tyypillisesti TAPI sisältää ja käyttää paljon tyyppimäärittelyjä ja unkarilaista notaatiota. Tyyppimäärittely tarkoittaa uuden nimen määrittämistä jollekin tietotyypille, esimerkiksi hyvin yleinen DWORD, joka on uudelleenmääritys uint32_t tietotyypille, merkitsemätön 32 bittinen kokonaisluku. Unkarilainen notaatio tarkoittaa muuttujien nimeämistä siten, että muuttujan tietotyyppi käy selville muuttujan nimestä. Tämä tehdään lisäämällä tietotyypin lyhenne muuttujan nimen alkuun, esimerkiksi muuttujan nimeksi halutaan Luku ja sen tietotyyppi on int, lopullinen nimi on iLuku, char olisi cLuku ja DWORD olisi dwLuku. (MSDN.)

Kaikki TAPI:n funktiot noudattavat palautusarvoiltaan samaa kaavaa. Onnistuessaa ne palauttavat nollan ja virheen sattuessa nollasta poikkeavan arvon. Näitä poikkeavia arvoja voidaan verrata TAPI:n otsikkotiedostossa määriteltyihin bittiflageihin. (MSDN.)

TAPI:n toiminnot on muutamaan osaan: Line device, phone device ja call center. Line device eli linjalaite hallitsee yhteyksiä ja se on ainoa pakollinen puheluiden soittamiseen. Sen toiminnot koostuvat pääasiassa puhelinlinjojen hallinnasta, puheluiden soittamisesta ja puheluihin vastaamisesta. Phone device eli puhelinlaite mahdollistaa itse puhelimen hallitsemisen, kuten äänenvoimakkuuden säädön ja luurin tiedot. TAPI2 mahdollistaa myös puhelinkeskuksen hallinnan Call center funktiokokoelman avulla, mutta ne ovat tarkoitettu palvelinsovellusten tekemiseen eikä niitä käytetä tässä työssä. (MSDN.)

Puhelintoimintojen ytimenä toimii linjalaite ja sen hallitsemat linjat. TAPI tarjoaa käyttöön muitakin kuin puhelinlinjoja, kuten Bluetooth, ja puhelinlinjat voidaan välittää erityyppistä tietoa, kuten puhetta tai dataa. (MSDN.)

6.2 Toimintojen käyttö

Tässä luvussa käsitellään TAPI:n tapaa käsitellä ja välittää tietoa sekä näytetään miten joitakin TAPI:n keskeisiin toimintoihin päästään käsiksi.

6.2.1 TAPI:n tietueet

Suuri osa TAPI:n tiedoista tallennetaan tietueisiin, joilla on jäseninä muutama tietueen koosta kertova kokomuuttuja ja vaihteleva määrä valmiita arvoja ja offset eli siirtymä arvoja. Kokomuuttujat ovat dwTotalSize, dwNeededSize ja dwUsedSize ja ne ovat aina kokonaislukuja. Valmiita arvoja voidaan käyttää sellaisenaan, mutta siirtymänä välitetty tieto pitää lukea muistista ja tallentaa erilliseen muuttujaan ennen käyttöä. Tietueet ovat yhtäjaksoista tietoa tietokoneen muistiin, joka alkaa tietueen osoittimesta ja jatkuu tietueen kokonaiskoon verran, dwTotalSize. Siirtymällä esitetty tieto luetaan lisäämällä siirtymä tietueen osoittimeen ja lu-

kemalla muistia tietotyypin koon verran. Joihinkin siirtymiin on liitetty erillinen kokoarvo, jolloin tietotyypin koon sijata käytetään siirtymän kokoa.

```
strcpy( lpTempString,
        (LPSTR)((LPBYTE) lpMultipleStringStruct +
        (LPBYTE) lpMultipleStringStruct->SecondStringOffset)
        );
```

Kuva 13Tiedon luku siirtymällä

Kuvassa 13 kopioidaan merkkijono strcpy- funktiolla. Kohteena on pitkä merkkijono-osoitin lpTempString ja lähteenä pitkä merkkijono-osoitin, joka rakennetaan lpMultipleStringStruct-tietueen osoittimesta ja sen SecondStringOffset- jäsenestä.

```
memcpy( lpTempString,
        (LPSTR)((LPBYTE) lpMultipleStringStruct +
        (LPBYTE) lpMultipleStringStruct->SecondStringOffset),
        lpMultipleStringStruct->SecondStringSize
        );
```

Kuva 14 Tiedon luku siirtymä-koko parilla

Kuvassa 14 kopioidaan myös merkkijono samalla tavalla kuin kuvassa 13. Vaikka kopioimiseen käytettävä funktio onkin eri, se toimii pitkälti samalla tavalla. Erona on lähteestä luettavien bittien määrä. Strcpy lukee koko lähdeosoittimen osoittaman muistiosoitteen ja tallentaa sen kohteeseen. Memcpy lukee lähdeosoitteesta vain kolmannen parametrin osoittaman määrän bittejä.

TAPI:n tietueiden sisältämällä dwNeededSize arvolla voidaan tarkistaa onko tietueeseen saatu mahdutettua kaikki mahdollinen tieto. Tietueisiin saadaan harvoin tallennettua kaikki mahdollinen tieto niiden minimikoon puitteissa. C/C++- kielien avainsanalla *sizeof* saadaan selville tietueen minimikoko, mutta merkkijonojen tai muiden taulukoiden kokoa ei voi tietää ennakkoon, joten on yksinkertaisempaa ensin varata tietueen minimikoko ja ensimmäisen kutsun jälkeen tehdä uusi varaus tietueen dwNeededSize- muuttujan mukaan. Tietueilla on mahdollista varata mielivaltainen määrä muistia, mutta tällöinkin tilaa voi olla liian vähän ja varauksessa tehdään suuri ylilyönti.

6.2.2 Alustus

TAPI:a käyttävät ohjelmat eivät varsinaisesti käynnistä TAPI:a vaan sen alaisia laitteita. Puhelintoimintoja varten käytetään linjalaitetta, jonka alustus tapahtuu joko vanhalla lineInitialize- funktiolla tai uudemmalla ja laajemmalla lineInitializeEx- funktiolla. Näiden funktioiden prototyypit löytyvät liitteestä 1. (MSDN.)

LineInitializeEx käytetään 2.0 version ja uudempien yhteydessä ja lineInitialize käytetään vanhempien yhteydessä. Funktiot alustavat kaikki järjestelmästä löytyvät linjalaitteet, joten alustus täytyy tehdä vain kerran ohjelmaa kohden. Minimissään alustus vaatii kolme asiaa: viittauksen laitelaitesovelluksen kahvan tallentamiseen, tiedon käytettävästä viestinvälitystavasta ja korkeimman tuetun API version. Viittaus välitetään lphLineApp- parametrina. Korkein tuetta API versio tallennetaan LPDWORD tyyppiseen muuttujaan ja välitetään lpdwAPIVersion- parametrina. Viestinvälitystapa valitaan joko antamalla osoitin ohjelman callback- funktioon lpfnCallback parametrina tai määrittämällä vaihtoehtoinen menetelmä lpLineInitializeExParams- parametrissa. Oletuksena TAPI käyttää callback menetelmää, jonka vuoksi funktio-osoittimen välitys riittää. Vaihtoehtoisia viestinvälitystapoja ovat Hidden Window, Event Handle ja Completion port TAPI2:ssa. TAPI3 tukee lisäksi Call Hub Tracking menetelmää. LpDwNumDevs täytetään osoittimella, johon alustus tallentaa ohjelman käytössä olevien linjojen määrän. (MSDN.)

LpLineInitializeExParams- parametria varten TAPI:ssa on määritelty LINEINITIZALIZEEXPARAMS -tietue, josta ohjelmoija luo ilmentymän ja täyttää haluamillaan asetuksilla. Haluttujen asetusten bittiliput sijoitetaan tietueen dwOptions- muuttujaan. (MSDN.)

Mahdollisesti tyhjäksi jätettäviä parametreja ovat hInstance ja lpszFriendlyAppName. Hinstance voidaan jättää tyhjäksi, jolloin TAPI käyttää kahvana prosessin juurisovelluksen kahvaa. LpszFriendlyAppName parametrilla voidaan ohjelmalle antaa selkeä nimi, joka yhdistetään linjojen puhelintietoihin. Myös LpDwNumDevs on mahdollista jättää tyhjäksi, mutta tällöin käytettävissä olevien linjojen määrä täytyy selvittää muuta kautta tai tutkia linjat sokkona. (MSDN.)

6.2.3 Linjojen käyttöönotto

Alustuksen jälkeen täytyy vielä tutkia ohjelman käytössä olevat linjat. Aluksi linjoihin ei voi viitata muulla kuin niiden indekseillä, joka on välillä 0 ja linjojen maksimimäärä, lpdwNumDevs, miinus yksi. Puhelinlinjat ovat yleensä järjestyksessä ensimmäisiä. (MSDN.)

Ennen jokaisen linjan käyttöä täytyy sille tehdä kaksi operaatiota: Neuvotella käytettävä TAPI versio ja hakea linjan ominaisuudet. Version neuvottelu tapahtuu lineNegotiateAPIVersion- funktiolla ja ominaisuudet haetaan lineGetDevCaps- funktiolla, joiden LpDwNumDevs löytyvät liitteestä 1. (MSDN.)

Käytettävä versio on ensimmäinen funktio mikä linjalle tulee ajaa. LineNegotiateAPIVersion parametrit ovat varsin yksiselitteisiä. HineApp on sovelluksen kahva linjalaitteeseen, dwDeviceID on neuvoteltavan linjan indeksi, dwAPILowVersion on TAPI:n alin tuettu versio, dwAPIHighVersion on TAPI:n korkein tuettu versio, lpdwAPIVersion on osoitin muuttujaan joka täytetään neuvotellulla versiolla ja lpExtensionID on osoitin muuttujaan, joka täytetään tiedoilla linjan tukemista laajennuksista. (MSDN.)

LineGetDevCaps- funktion avulla linjasta saadaan selville sen tarkat omanaisuudet. Ennen funktion kutsumista täytyy esitellä LINEDEVCAPS tyyppinen muuttuja, johon linjan ominaisuudet tallennetaan. LineGetDevCaps- funktion parametrit ovat varsin suoraviivaiset: HlineApp on ohjelman kahva linjalaitteeseen, dwDeviceID on käsiteltävän linjan indeksi, dwAPIVersion on aikaisemmin neuvoteltu käytettävä TAPI:n versio, dwExtVersion on laajennusten käytettävä versio ja lpLineDevCaps on osoitin muuttujaan, joka täytetään linjan tiedoilla. Näistä parametreista ainoastaan dwExtVersion voidaan jättää tyhjäksi. Funktion käyttö ei kuitenkaan ole aivan yksinkertaista. Onnistuessaankin funktio ei välttämättä tallenna kaikki mahdollisia tietoja lpLineDevCaps osoittamaan muuttujaan. (MSDN.)

Ohjelmoija joutuu itse hallitsemaan LINEDEVCAPS- muuttujan muistia eikä voi etukäteen tietää paljonko muistia TAPI tarvitsee kaikkien tietojen tallentamiseen. Tämän takia lineGetDevCaps pitää ajaa ainakin kahdesti. Ensimmäisellä kerralla TAPI täyttää LINEDEVCAPS muuttujan dwNeededSize jäsenen. Tämän tiedon avulla muuttujalle voidaan varata tarvittava määrä muistia ja lineGetDevCaps ajaa uudelleen, jolloin TAPI:n pystyy täyttämään LINEDEVCAPS- muuttujan kaikilla linjan tiedoilla. (MSDN.)

6.2.4 Viestinvälitys

TAPI voi välittää viestejä asiakasohjelmille viidellä eri tavalla, joista tässä työssä on tutkittu kahta: Callback ja Event Handle. Käytettävästä tavasta riippumatta TAPI:n viesti koostuu viidestä muuttujasta, joiden tietotyypit tai nimet eivät muutu. Ohjelman käyttämää viestinvälitystapaa voidaan siis vaihtaa muuttamatta viestejä käsitteleviä algoritmeja merkittävästi, mutta jokainen viestinvälitystapa vaatii erilaisen vastaanottomekanismin. (MSDN.)

Callback on vanhempi menetelmä, jota käytettiin TAPI:n ensimmäisissä versioissa. Siinä TAPI kutsuu asiakasohjelman funktiota ja viesti välitetään kutsun parametreissa. Asiakasohjelman täytyy esitellä TAPI:n määritelmän mukainen callback- funktio ja välittää funktio-osoitin vähän funktioon laitealustuksien yhteydessä. (MSDN.)

Event handle on uudempi menetelmä, jossa TAPI ilmoittaa viestistä ensin tapahtumalla, jonka jälkeen asiakasohjelma osaa tarkistaa viestipuskurin. Asiakasohjelma luo tapahtumakahvan ja välittää sen TAPI:lle. TAPI lähettää tämän kahvan kautta tapahtuman asiakasohjelmalle, joka herättää asiakasohjelmassa viestinhakukutsun. Viestinhaku tehdään lineGetmessage- funktiolla, jonka parametrina välitetty LINEMESSAGE- tietue täytetään viestillä. LineGetMessage voi myös odottaa viestiä. Tällöin funktiolle voidaan luoda uusi säie, joka odottaa TAPI:n viestejä loputtomasti. (MSDN.)

Viestien tulkinta

```
typedef struct linemessage_tag {
  DWORD     hDevice;
  DWORD     dwMessageID;
  DWORD_PTR dwCallbackInstance;
  DWORD_PTR dwParam1;
  DWORD_PTR dwParam2;
  DWORD_PTR dwParam3;
} LINEMESSAGE, *LPLINEMESSAGE;
```

Kuva 15 TAPI:n LINEMESSAGE tietue. Kuvakaappaus MSDN:stä

TAPI:n viestit käsitellään käymällä viestin jäsenet asteittain läpi jollakin vertailurakenteella, kuten switch-case tai if-else. hDevice on kahva laitteeseen mihin viesti liittyy tai puhelun saa-

puessa puhelun kahva. DwMessageID sisältää jonkun tapi.h:ssa määritetyn viestityypin. dwParam1, dwParam2 ja dwParam3 sisältävät tarkentavia tietoja joko numeroina tai bittiflageina. Moni viesti käyttää vain kolmea ensimmäistä parametria, muutamat neljää ja hyvin harva kaikkia viittä. Tapi.h:ssa määritellään kaikki käytettävät bittiflagit. (MSDN.)

Erilaisia viestityyppejä ovat muun muassa uuden saapuvan puhelun LINE_APPNEWCALL, puhelun tilan muutoksesta kertoja LINE_CALLSTATE ja asynkronisen funktion suorituksesta kertova LINE_REPLY. Nämä ovat vain linjalaitteen viestejä ja puhelinlaitteella on omat viestinsä. (MSDN.)

6.2.5 Puheluiden hallinta

Puheluiden soittaminen

TAPI:ssa puheluiden soittaminen tapahtuu linjalaitteen lineMakeCall- funktiolla, jonka prototyyppi löytyy liitteestä 1. Tälle funktiolla annetaan käytettävän linjan kahva, hline, muuttujan osoitin puhelun kahvan tallentamista varten, lphCall, ja soitettava puhelinnumero merkkijonona, lpszDestAddress. Funktiokutsun jälkeen TAPI välittää viestin puhelun tiloista ja lopulta ”hälyttää” viestin. TAPI osaa välittää myös viestin kun puhelu on hyväksytty, mutta tämän viesti on dokumentoitu arvaamattomaksi. Tämä tarkoittaa, että ei ole takeita onko puhelu oikeasti vastaanotettu ja hyväksytty vastaanottajan päässä vaikka TAPI viestin välittääkin. Tästä huolimatta TAPI osaa avata puhelinlaitteen luurin ja puhelin toimii ongelmitta. Puhelu voidaan lopettaa kutsumalla funktiota lineDrop. Puhelun katketessa TAPI lähettää viestin puhelun katkeamisesta ja syyn sen katkeamiseen. Tästä voidaan tarkistaa katkesiko puhelin oikein eli puheluun osallistuneen osapuolen toimesta vai jostain teknisestä syystä. (MSDN.)

Puhelinnumero ja maakoodi pitää kääntää soitettavaan muotoon. Tämä tehdään lineTranslateAddress- funktiolla, joka vaatii linjalaitteen kahvan, käytettävän linjan tunnuksen, TAPI:n version, käännettävän numeron ja osoittimen TranslateOutput muuttujaan, johon käännetty numero tallennetaan. TranslateOutput pitää olla LPTRANSLATEOUTPUT- tietue ja sen tarvitsema todellinen koko tiedetään vasta ensimmäisen lineTranslateAddress- kutsun jälkeen. (MSDN.)

Puheluiden vastaanottaminen

Soittamisesta poiketen puheluiden vastaanottaminen on funktiokutsua vaikeampi toteuttaa, koska toteutus täytyy ajatella käänteisessä järjestyksessä. Saapuvan puhelun käsittely lähtee käyntiin viestinkäsittelijästä. TAPI välittää viestin, joka kertoo saapuvasta puhelusta. Viestin parametreissa on kuvan 16 mukaisia tietoja. (MSDN.)

```
LINE_APPNEWCALL
    hDevice = (DWORD) hLine;
    dwCallbackInstance = (DWORD) dwInstanceData;
    dwParam1 = (DWORD) dwAddressID;
    dwParam2 = (DWORD) hCall;
    dwParam3 = (DWORD) dwPrivilege;
```

Kuva 16 Viesti saapuvasta puhelusta. Kuvakaappaus MSDN:stä

Tämä viesti sisältää puhelun käyttämän linjan, puhelun osoitetunnuksen, puhelun kahvan ja sovelluksen oikeudet puheluun. DwCallbackInstace sisältää kahvan käytettävään callbackiin kun puhelu avataan, mutta tässä sovelluksessa ei käytetä callback menetelmää. Tärkein näistä tiedoista on dwParam2 sisältämä hCall eli puhelun kahva, jolla puheluun voidaan viitata suoraan. Viestissä välitetyt hLine ja dwAddressID puolestaan ovat puhelun tarkat paikkatiedot. Kahva on viittaus puheluun, mutta puhelu sijaitsee linjan hLine osoitteessa dwAddressID. Viestin viimeinen parametri dwPriviledge voi olla yksi kahdesta arvosta, LINECALLPRIVILEDGE_OWNER, omistaja, _MONITOR, seuraaja tai _NONE, ei oikeuksia. Omistaja voi selata puhelun tietoja sekä muokkaamaan sen tietoja ja tilaa. Seuraaja voi selata puhelun tietoja, mutta ei voi muokata sitä. Ei-oikeuksia -arvolla sovelluksella ei ole mitään oikeuksia puheluun, puhelun kahva on tyhjä eikä mitään viestin arvoja pitäisi käyttää. (MSDN.)

Välittömästi LINE_APPNEWCALL viestin jälkeen TAPI välittää LINE_CALLSTATE viestin, jonka hDevice- parametri vastaa uuden puhelun kahvaa, aikaisemmin välitetty hCall. Viestin dwParam1 kertoo puhelun ensimmäisen tilan, joka on saapuvalla puhelulla yleensä LINECALLSTATE_OFFERING, jota dwParam2 tarkentaa. (MSDN.)

Tarjolla olevalle puhelulle eli LINECALLSTATE_OFFERING tilassa olevalle puhelulle voidaan ajaa kaksi funktiota: hyväksyä se lineAccept- funktiolla tai hylätä se lineDrop- funktiolla. Puhelu voidaan myös katkaista samalla lineDrop- funktiolla milloin tahansa. Kumpikin funktio vaatii parametriksi puhelun kahvan. Puhelu voidaan myös jättää täysin huomiotta, jolloin soittaja jää vain ihmettelemään ongelmitta hälyttävää puhelinlinjaa. Hyväksyttäessä TAPI avaa puhelimen luurin ja käyttäjät voivat keskustella. (MSDN.)

7 KEHITYSTYÖ

Projektia aloitettaessa kehityskohteesta oli vain tarvemäärittely ja käyttöliittymä piirrokset, mutta ei mitään suunnitelmaa tai ohjeistusta sovelluksen toiminnasta, integraatiosta tai toteutuksesta. Ei myöskään tiedetty, miten Windows Mobile- järjestelmän puhelintoiminnot on toteutettu ja miten ne saataisiin ohjelmoijan hallintaan. Ensimmäiseksi täytyi siis selvittää laitteiston nykytila ja käyttökelpoiset rajapinnat. Näiden selvittyä voitiin suunnitella ja toteuttaa itse sovellus.

Windows Mobile- järjestelmän puhelintoiminnoista huolehtii sovellus nimeltä cprog. Sovelluksen tehtävät eivät rajoitu vain puheluiden käsittelyyn, vaan ohjelma muun muassa tarkkailee puhelinverkkosignaalin laatua. Windows Mobilen käyttämä .NET Compact Framework tarjoaa vain yksinkertaisen soittofunktion, mikä käyttää käyttöjärjestelmän puhelinsovellusta ja käyttöliittymää. Puhelinsovelluksen on tarkoitus korvata oletussovellus täysin, joten Compact- ympäristön tarjoama toiminnallisuus ei riittänyt projektin toteutukseen.

Ohjelmointirajapintaa etsiessä selvisi Windowsin jo sisältävän rajapinnan puhelintoiminnoille. Rajapinta on Windowsin TAPI eli Telephony Application Programming Interface. Cprogiin verrattuna TAPI:n ainoa puute on tieto puhelinverkon signaalin laadusta. TAPI ei välitä tietoa signaalin vahvuudesta, joten toiminta pitäisi toteuttaa eri kirjaston avulla. Tätä ei kuitenkaan katsottu tärkeäksi toiminnoksi projektin kannalta, joten selvitys kyseisen toiminnon osalta lopetettiin.

TAPI:sta on käytettävissä kaksi versiota, TAPI3 ja TAPI2. On haluttavaa käyttää TAPI 3 versiota, koska .NET:in COM yhteistoiminnalla COM objekteja voi käyttää kuin .NET objekteja ja sovelluskehitys voitaisiin tehdä kokonaisuudessaan C#- kielellä. .NET Compact-ympäristön dokumentaatiosta saatiin käsitys, että kehysympäristö ei tue COM- yhteistoimintaa. Näin ollen päädyttiin käyttämään TAPI2 versiota.

7.1 Sovelluksen alustava suunnittelu

TAPI:n käyttö on mahdollista toteuttaa kahdella tavalla: Tuodaan jokainen kirjaston funktiot, arvo ja määrittely .NET sovellukseen ja tehdään koko ohjelma C#:lla tai luodaan erillinen C/C++- kielinen ohjelma, joka toimii viestinvälittäjänä TAPI:n ja käyttöliittymän välillä. Valtavan kopiointityön välttämiseksi projekti toteutettiin jälkimmäisellä tavalla eli TAPI:lle kirjoitettiin omiin tarpeisiin mukautettu rajapinta.

Ohjelmaa kokeiltiin yksinkertaisella käyttöliittymällä ja wrapperilla. Wrapper kykenee alustamaan TAPI:n linjalaitteen, soittamaan puhelun ja sammuttamaan TAPI:n. Käyttöliittymä sisältää yhteystoiminnan kirjastoon, tekstikentän puhelinnumeron kirjoittamiseen ja napin puhelun soittamiseen. Sovelluksella onnistuttiin soittamaan useampaan eri numeroon, joten TAPI todettiin sopivaksi rajapinnaksi projektia varten.

Suunnittelun alussa toiminnot täytyi jakaa ensisijaisiin ja toissijaisiin. Sovellukselta ei vaadittu montaa isoa toimintoa, mutta pienempiä toimintoja oli useita. Ensisijaisiksi toiminnoiksi määriteltiin puheluiden soittaminen ja vastaanottaminen ja minimaalisten puhelutietojen esittäminen. Toissijaisia toimintoja olivat tarkat puhelutiedot, niiden tallentaminen ja selailu, puhelinluettelo ja puhelimen asetusten muokkaaminen, kuten soittoääni ja äänenvoimakkuus.

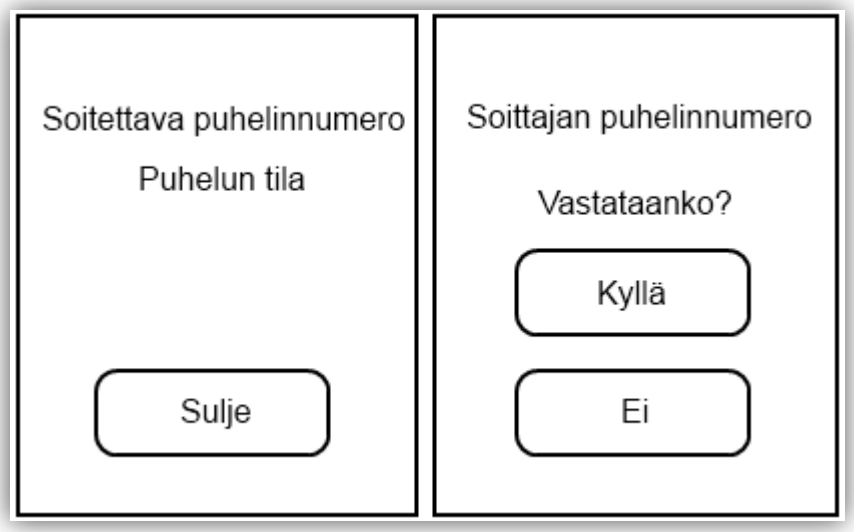
Käyttöliittymä suunniteltiin muistuttamaan HOTI:n muiden osien ulkoasua ja toimintaa. Kaikki HOTI:n moduulit ajetaan kehyksessä, jonka yläosassa on moduulin sammutus, nimi ja ohjenapit. Näiden alla on moduulin tämän hetkisen sivun nimi ja nuolet sivun vaihtamiseen. Sivun alalaidassa on nappi virtuaalisen näppäimistön avaamiseen. Loppunäyttö on varattu sivun esittämiseen.

7.1.1 Ensisijaiset toiminnot

Puhelimen keskeiset toiminnot ovat puheluiden soittaminen ja vastaanottaminen. Näitä toimintoja hallitaan puhelun tilasta kertoville ikkunoilla ja sovelluksen pääsivulta. Pääsivu sisältää tekstikentän ja numeropainikkeet puhelinnumeron syöttämiseen ja soita puhelu napin.

Saapuvasta puhelusta kerrotaan käyttäjälle viesti-ikkunalla, jonka kautta hän voi joko hyväksyä tai hylätä puhelun.

Puheluiden ohjaamiseen on kaksi viesti-ikkunaa: Saapuvan puhelun kysely ja toteutuneen puhelun tiedot.



Kuva 17 Toteutuneen puhelun ja saapuvan puhelun viesti-ikkunat

Soitettaessa avataan toteutuneen puhelun ikkuna, jonka "Puhelun tila" kentässä näytetään puhelun eteneminen, kuten *hälyttää* tai *varattu*. Vastaanottajan hyväksyessä puhelun tilaksi päivitetään puhelun kesto.

Puhelun saapuessa avataan kysely ja se näyttää soittajan numeron ja mahdollisen numeroon liitetyn nimen ja odottaa käyttäjän hyväksyntää tai hylkäystä. Hylkäys sulkee ikkunan välittömästi ja ohjelma palaa normaaliin toimintaansa. Hyväksyminen avaa toteutuneen puhelun ikkunan ja puhelun tilaksi päivitetään puhelun kesto.

Puhelun loppuessa avataan toteutuneen puhelun ikkuna. Kytkeytyneestä puhelusta näytetään kokonaiskesto ja epäonnistuneesta soitosta katkaisun syy.

Minimaaliset puhelutiedot voidaan kerätä osittain oman laitteen tiedoista ja osittain TAPI:lta. Vastapuolen puhelinnumero saadaan soittaessa käyttäjän syötteestä, mutta saapuvan puhelun numero pitää tiedustella TAPI:lta. Puhelun aloitusaika voidaan aina ottaa järjestelmän kellos-

ta siltä hetkeltä kun puhelun soittaminen aloitetaan tai kun käyttäjä hyväksyy saapuvan puhelun. Puhelun tilan määritys on lähes täysin TAPI:n vastuulla. Ei voida olettaa että puhelu hälyttää välittömästi tai ollenkaan kun käyttäjä hyväksyy puhelun soittamisen eikä puhelun katkeamisesta saada tietoja muuten kuin TAPI:lta. Ainoa tila mikä voidaan selvittää ilman TAPI:a on saapuvan puhelun hyväksyminen tai hylkääminen, mutta hyväksymisen jälkeenkään ei voida tietää milloin ja mihin tilaan puhelu vaihtuu ilman TAPI:n viestiä.

7.1.2 Toissijaiset toiminnot

Toissijaisia toimintoja ovat tarkat puhelutiedot, niiden tallentaminen ja selailu, puhelinluettelo ja puhelimen asetusten muokkaaminen, kuten soittoääni ja äänenvoimakkuus. Puhelutietojen tallentamiseen liittyy myös vastaamattomien puheluiden tallentaminen.

Tarkkoihin puhelutietoihin kuuluu puhelun keston määritys, soittajan nimen esitys ja puhelun mahdollinen katkaisun syy. Puhelun kesto saadaan tallentamalla kellonaika järjestelmän kellosta puhelun katketessa ja laskemalla aloitusajan ja lopetusajan erotus. Soittajan nimen esittämiseen tarvitaan puhelinluettelo, josta numeroa vastaava tunniste voidaan hakea. Puhelun katkeamisen syy voidaan toisinaan saada TAPI:lta, mutta näihin ei välttämättä aina voi luottaa. Lisäksi ei välttämättä ole tarpeellista tallentaa epäonnistuneiden puheluiden puhelutietoja, jos syy on verkossa eikä vastapuolessa.

Puhelinluettelo vaatii tietokannan ja rajapinnan. Tietokannaksi riittää yksinkertainen kahden tietueen taulu, johon tallennetaan puhelinnumero ja tunniste. Tunniste on käyttäjän määrittelemä, kuten nimi tai nimimerkki. Puhelinluetteloa selatessa riittää kun käyttäjälle esitetään numeroiden tunnisteet. Saapuvan puhelun numeroa verrataan tietokantaan tallennettuihin numeroihin ja vastaavuuden löytyessä sen tunniste esittää käyttäjälle. Puhelinluettelon ylläpitäminen on oletuksena käyttäjän tehtävä.

Puhelinluetteloa voidaan laajentaa ryhmillä, muilla yhteystiedoilla, pikavalinnalla, puheluestoilla ja keskitetyllä hallinnalla. Puhelinnumerot voisi sijoittaa johonkin ryhmään, jonka jälkeen numeroita voi selata ja hakea ryhmän perusteella. Lisättäviä yhteystietoja voisi olla esimerkiksi lähiosoite tai sähköpostiosoite. Pikavalinta voitaisiin toteuttaa yhdistämällä pienet numerot, esimerkiksi yhdestä kymmeneen, johonkin puhelinluettelon nimeen tai numeroon.

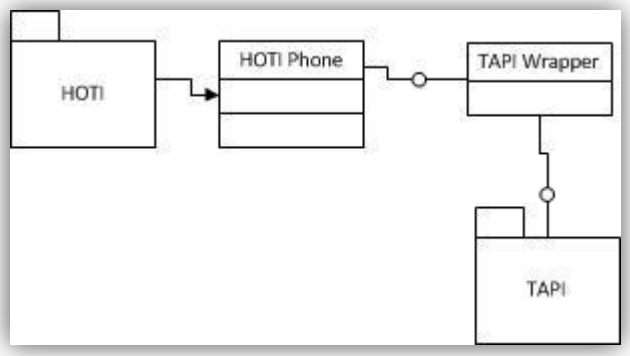
Puhelinluettelon yhteyteen voisi myös luoda sallittujen tai kiellettyhen numeroiden listan. Kiellettyjen numeroiden listalla voitaisiin estää yksittäisiä numeroita tai muutamalla algoritmilla esimerkiksi 0800-alkuiset numerot jo ohjelmassa eikä operaattorin numeronestoa tarvittaisi. Haittapuoleni kuitenkin on olettamus, kaikki mainitsemattomat numerot ovat sallittuja. Sallittujen numeroiden listalla voitaisiin helposti estää paljon numeroita ja sallia vain muutamia numeroita, kuten oman tiimin tai yrityksen numerot. Haittapuolena on korkea ylläpito, sillä jokainen sallittu numero täytyy erikseen lisätä ennen kuin siihen voi soittaa ja tämä laskee puhelimen käytettävyyttä jos käyttäjän pitää soittaa usein uusiin numeroihin. Keskitetyllä puhelinluettelon hallinnalla voitaisiin muokata kaikkien puhelimien puhelinluetteloja kerralla. Tämä mahdollistaisi esimerkiksi tietyn numeron kieltämisen koko laitekannalta.

Puhelutietojen tallentamiseen ja seilailuun tarvitaan tietokanta ja rajapinta. Tietokanta koostuisi yhdestä taulusta, joka koostuu kolmesta tietueesta: vastapuolen puhelinnumero, puhelun aloitus- ja lopetusaika. Mahdollinen tunniste voidaan hakea puhelinluettelosta puhelinnumeron perusteella ja puhelun kesto laskea aloitus- ja lopetusajasta. Lisätietoina voidaan tallentaa puhelun suunta, vastattiinko puheluun ja puhelun katkeamisen syy. Selailua varten käyttäjälle joko näytetään kaikki tiedot kaikista puheluista kerralla tai puheluista voidaan ottaa muutama tieto ja käyttäjä voi halutessaan avata yksittäisen puhelun kaikki tiedot.

Puhelimen asetukset ovat hyvin pieni osa sovellusta ja kattavat lähinnä puhelimen soittoäänen ja äänenvoimakkuuden. Soittoäänen vaihtaminen onnistuu antamalla käyttäjän valita soitettava äänitiedosto ja soittamalla tätä tiedostoa toistuvasta puhelun saapuessa. Äänenvoimakkuuden säätöön kuuluu kaksi eri osaa, järjestelmän äänet, johon kuuluu esimerkiksi soittoääni ja mahdolliset käyttöliittymän äänet, ja luurin. Luuria varten moni laite sisältää äänensäädön laitepuolella, joten tähän ei ole syytä kajota ohjelmallisesti. Olisi kömpelöä kesken puhelun todeta vastapuolella "hetkinen" ja navigoida tiensä asetusten uumeniin luurin voimakkuutta säätämään. Järjestelmän äänien säätämiseenkin Windows Mobile sisältää oman käyttöliittymäelementin, joten tämä osa sovellusta on lähinnä hienostelua.

7.1.3 Ohjelman rakenne

Kehitettävässä sovelluksessa on kaksi osaa: HOTI:n puhelin plugin ja TAPI wrapper.



Kuva 18 Sovelluksen rakenne

HOTI on sovelluskokonaisuus johon kuuluu useita luokkia ja toimintoja erilaisia toimintoja ja tehtäviä varten. HOTI Phone on vain yksi osa tätä kokonaisuutta ja sen tehtävänä on mahdollistaa puhelintoiminnot sovelluksen sisällä.

HOTI Phone on C# kielellä kirjoitettu luokka, joka ohjaa puhelimen käyttöliittymää, halinnoi puhelinluetteloa ja puhelutietokantaa. Puhelintoimintoja varten HOTI Phone tuo wrapperin ulkoiset funktiot, sisältää callback- funktion wrapperin viestien tulkintaan ja esittää wrapperin viestit käyttäjälle. Käyttöliittymän ohjaukseen kuuluu siirtyminen puhelimen eri osien välillä: puhelin, puhelinluettelo ja puhelutiedot.

TAPI Wrapper on C/C++ kielellä kirjoitettu kirjasto, joka käyttää TAPI:n funktioita suoraan, käsittelee TAPI:n viestejä, tulkitsee virhekoodeja ja ohjaa TAPI:n toimintaa. Lisäksi wrapper välittää sitä käyttävälle ohjelmalle sellaisia viestejä, jotka ovat käyttäjälle olennaisia. Wrapper paljastaa funktioita ja tietotyyppejä, joilla wrapperia käytetään. Funktiot eivät ole yksi yhteen vastineita TAPI:n funktioille vaan kokonaisen toiminnon suorittavia kokonaisuuksia, kuten linjalaitteen ja linjojen käynnistys, tai yksinkertaistuksia TAPI:n funktiosta,

kuten puhelun soittaminen. Viestinvälitykseen wrapperilta asiakasohjelmaan käytetään callback- funktiota. Funktiolla on muutama parametri, jotka kertovat viestin tyypin ja sisällön. Viestin tyyppi välitetään bittilipulla ja viestit ovat merkkijonoja tai kokonaislukuja.

7.2 Alpha versio

Alpha version on ensimmäinen toimiva versio sovelluksesta, jossa on mukana kaikki ohjelman kriittiset toiminnot. Alphan tavoitteena on luoda yksinkertaisin toimiva versio lopullisesta ohjelmasta, jonka toimintaa on vielä helppo muokata. Tässä vaiheessa toimintoja voidaan vielä lisätä tai poistaa pienellä vaivalla ja niissä mahdollisesti piileksivät virheet on helpompaa ja nopeampaa löytää ja korjata.

Tässä työssä kehitettävän sovelluksen alpha versio sisältää puheluiden soiton ja vastaanoton sekä näihin tarvittava yksinkertainen käyttöliittymä. Käyttöliittymä on tekstikenttä johon soitettava numero kirjoitetaan, "soita" nappi sekä kaksi ponnahdusikkunaa. Wrapper tehdään näyttämään valmiilta; kaikki wrapperin käyttämät ja paljastamat funktiot, muuttujat ja luettelot esitellään, mutta funktoiden toiminta on vain minimaalinen.

Sovelluksen toiminnan seuraamiseksi käyttöliittymän osaksi tehdään tapahtumalogia kirjoittava funktio. Funktio ottaa vastaan merkkijonon ja kirjoittaa kellonajan ja merkkijonon uudeksi riviksi tekstitiedoston loppuun. Debug tagien avulla kutsut tähän funktioon eivät päädy julkaisuversioon.

7.2.1 Alustava TAPI wrapper

Wrapperin otsikkotiedostossa esitellään kaikki mikä asiakasohjelman tietoon halutaan antaa. Näitä ovat viestityyppien numeerinen määrittely, callback- funktion prototyyppi, TAPI versioiden numeeriset määrittelyt, alhaisin tuettu TAPI versio ja paljastettavat funktiot. Lähdekooditiedostossa esitellään kaikki kirjaston sisäiset osat, kuten muuttujat, funktiot ja määrittelyt. Lisäksi lähdekooditiedosto sisältää kirjaston kaikkien funktioiden rungot. Kahteen tiedos-

toon jaolla on tarkoitus piilottaa kirjaston toimintaa ja estää ulkopuolista ohjelmaa käyttämästä wrapperin muuttujia ja sisäiseen käyttöön tarkoitettuja funktioita.

Wrapperin toiminta keskittyy linjalaitteen viestinkäsittelijään. Kaikki TAPI:n funktiokutsut aiheuttavat jonkinlaisen viestin ja näin wrapper saa tiedon kutsun vaikutuksesta. Esimerkiksi saapuva puhelu herättää viestin ja puhelua soittaessa voidaan varmistua puhelinnumeron ynnä muun kelvollisuudesta.

Alustukset

Alustusfunktion tehtävä on alustaa wrapper ja linjalaite. Parametreina funkiolla on ohjelman kahva, funktio-osoitin callback funktiolle ja ohjelman nimi. Näistä ainoa pakollinen parametri on callback- funktion osoitin, koska ohjelmaan voidaan viitata TAPI:n luomalla kahvalla ja ilman ohjelman kahvaa TAPI käyttää juurisovelluksen kahvaa. Callback- funktion osoitin ja ohjelman nimi tallennetaan wrapperiin sisäisiin muuttujiin ja ohjelman kahva annetaan linjalaitteen alustuksen parametriksi. Linjalaitteen alustus vaatii parametreiksi lähinnä muistiosoitteita muuttujiin, joihin se voi tallentaa tietoja. Tärkeää oikean toiminnan kannalta on funktion viimeinen parametri, LINEINITIALIZEEXPARAMS- tyyppinen tietue, dwOptions-jäseneen asetetaan arvo LINEINITIALIZEEXOPTION_USEEVENT jolloin linjalaite käyttää haluttua Event Handle- viestinvälitystapaa.

Linjalaitteen alustuksen jälkeen wrapper tutkii käytössä olevat linjalaitteet. Puhelinlinjoja tarvitaan vain yksi, joten ensimmäisen kelvollisen linjan löytyessä sen kahva ja ominaisuudet tallennetaan globaaliin tietueeseen, linja avataan ja ohjelma siirtyy eteenpäin. Tätä avattua linjaa tarvitaan ainoastaan puheluita soittaessa. Saapuvan puhelun linja on automaattisesti avoin ja linjan kahva saadaan saapuvan puhelun viestissä.

Viestinkäsittelijä

Wrapper on suunniteltu käyttämään viestikeskusta loputtomalla odotusajalla. Tämä vaatii wrapperia luomaan erillisen säikeen viestinkäsittelyä varten. Jos viestinkäsittelijä ajettaisiin ensisijaisessa säikeessä, koko kirjasto ja sitä käyttävä ohjelma jäisi odottamaan viestejä TAPI:lta. Tämä toteutus pakottaa wrapperin monisäikeisyyteen eikä käyttäjä voi vaikuttaa

wrapperin sisällä luotuihin säikeisiin. Vaihtoehtoja tälle toteutuksella olisi paljastaa viestinkäsittelijä, jolloin käyttäjä itse käynnistäisi viestinkäsittelijän uuteen säikeeseen tai viestinkäsittelijöitä voisi kirjoittaa useamman, joista käyttäjä voisi valita sopivan.

Viestinkäsittelijä on sisäkkäinen switch-case rakenne, joka käy asteittain läpi viestin parametrit. Joistain viestityypeistä lähdetään heti suorittamaan jatkotoimia, mutta suurin osa vaatii useita switch-case rakenteita, jotka tutkivat viestin dwParam1, dwParam2 ja dwParam3 muuttujia. Esimerkiksi LINE_CALLSTATE tapauksessa dwParam1 kertoo mikä tieto puhelussa on muuttunut ja dwParam2 sisältää tiedon uuden arvon. DwParam3 sisältää yleensä tarkentavaa tietoa, esimerkiksi LINE_APPNEWCALL viestissä se kertoo sovelluksen oikeuden puheluun.

Puheluiden soitto

Puheluiden soittaminen tapahtuu TAPI:n lineMakeCall- funktiolla, joka vaatii parametreiksi käytettävän linjan kahva, puhelun kahvalla täytettävän osoittimen, soitettavan numeron, maakoodin ja puheluparametrit. Kahvat ovat yksinkertaisia, mutta puhelinnumero, maakoodi ja puheluparametrit täytyy rakentaa. Wrapper yksinkertaistaa tätä prosessia paljastamalla MakeCall- funktion, joka vaatii vain soitettavan numeron ja maakoodin.

Puheluihin vastaaminen

TAPI ilmoittaa saapuvasta puhelusta viestillä. Viestinkäsittelijä huomaa viestin, kerää puhelun tiedot muistiin ja ilmoittaa asiakasohjelmalle puhelusta callback- funktiolla. Asiakasohjelma voi hyväksyä tai hylätä puhelun wrapperin paljastamilla funktioilla. Hyväksyttäessä wrapper kutsuu TAPI:n lineAnswer- funktioa, joka kytkee puhelinlinjan.

Puhelun lopetus

Wrapper hyväksyy vain yhden puhelun kerrallaan ja tallentaa aktiivisen puhelun kahvan. Puhelu voidaan katkaista milloin tahansa wrapperin paljastamalla funktiolla, joka kutsuu TAPI:n lineDrop- funktiota aktiiviselle puhelulle ja poistaa puhelua koskevat tiedot.

Wrapperin sulkeminen

Sulkeminen koostuu globaalien osoittimien vapauttamisesta ja säikeiden, linjojen ja laitteiden sulkemisesta. Itse TAPI:a ei tarvitse sammuttaa, mutta alustetut laitteet kyllä. Säikeet suljetaan windowsin CloseHandle- funktiolla. Mahdollinen käynnissä oleva puhelu katkaistaan lineDrop- funktiolla ja linjat suljetaan lineClose- funktiolla. Mahdollinen puhelinlaite sammutetaan phoneClose- funktiolla. Linjalaite sammutetaan lineShutdown- funktiolla.

7.2.2 Alustava käyttöliittymä

Käyttöliittymää ei vielä alpha versiossa kehitetä merkittävästi. Riittää kun kaikki kriittiset elementit ovat olemassa. Näillä elementeillä ohjataan ohjelman kriittisiä toimintoja, tässä tapauksessa puhelinnumeron syöttökenttä ja soita -nappi, käyttöliittymän navigointi ja viestinvälitys käyttäjälle. Alphaversiossa käytettävien elementtien ei myöskään tarvitse olla ulkonäöltään tai kaikilta toiminnoiltaan lopullisia, kunhan ne toteuttavat niihin sidotun kriittisen toiminnon. Esimerkiksi käyttöliittymään voi kuulua tekstikenttä, jonka tärkein toiminto on näyttää käyttäjän syöttämä puhelinnumero. Alpha versiossa riittää .NET:in peruselementti EditBox tai Label, mutta lopullisessa versiossa voidaan käyttää jotain johdettua versiota, jolla voi olla muokattu ulkonäkö tai sisältää erikoisia viestinkäsittelyjä. Viestikenttä voisi vaikka kuunnella soita -nappia, jonka painaminen aiheuttaisi viestikentän sisällön tallentamisen väliaikaiseen muuttujaan, joka puhelun loputtua tallennetaan puhelutietona.

Puhelimen valmiiseen käyttöliittymään kuuluu puhelin sivu, puhelinluettelosivu ja puheluhistoriasivu. Näiden välillä liikutaan näytön ylälaidassa sijaitsevalla palkilla, jolla voidaan siirtyä vasemmalle tai oikealle. Alpha versioon luodaan nämä sivut, mutta vain puhelin sivu sisältää elementtejä. Puhelimen käyttöliittymä koostuu tekstikentästä sekä tavallisesta puhelimen kolme kertaa neljä numeronäppäimistöstä. Numeronäppäimet luovat mielikuvan puhelimesta ja näin helpottaa sovelluksen käyttöä. Minimaalisinta ulkoasua haluttaessa olisi numeronäppäimistö voitu jättää pois ja käyttää laitteen omaa virtuaalinäppäimistöä numeron syöttämiseen.

Puheluviestien ilmoitukseen käytetään kahta uutta elementtiä, kuvan viesti-ikkunat mukaisia viesti-ikkunoita. Ikkunoilla on alustavan suunnittelun mukainen toiminta, niiden sisältämää puhelun kestoa ei vielä tässä vaiheessa toteuteta. Soittajan puhelunnumero saadaan wrapperilta.

Käyttöliittymän toteutus oli suoraviivainen toimenpide. Kaikki tarvittavat elementit oli olemassa joko .NET:in tai HOTI:n kirjastoissa. Toteutus käsitti siis lähinnä olioiden luontia ja palikoiden liittämistä toisiinsa.

7.2.3 Alustava yhteistoiminta

Alphaversiossa yhteistoiminta käyttöliittymän ja wrapperin välillä koostuu yksittäisistä funktiokutsuista, kuten alusta wrapper tai soita puhelu. Tärkeimmät viestit kuten uudet puhelut ja suorituksen epäonnistumiset tulee välittää asiakasohjelmalle, mutta tiedonjakoa, monipuolista viestientulkintaa tai jokaista pikkuvirhettä ei välitetä. Wrapper paljastaa jo alphavaiheessa kaikki funktiot, joten ne voidaan tuoda käyttöliittymään. Kaikki funktiot eivät välttämättä toimi täysin, mutta toteuttavat kriittisimmät toimintonsa.

```
extern "C" LUOVAPHONEDLL_API void expInitPhoneApp(
        HINSTANCE hwndMain,                 //C# application handle
        LPWSTR lpwsAppName,                 //C# application name
        CALLBACKINTERFACE interfacehandle   //C# application callback
);
```

Kuva 19 Esimerkki paljastetusta funktiosta C kielellä

Kuvan 19funktio eroaa normaalista funktion määrittelystä pelkästään otsikon alkuun kirjoitetuilla extern "C" ja LUOVAPHONEDLL_API __declspec(dllimport) määrityksillä. Extern sanalla funktio määritetään ulkoiseksi ja __declspec(dllimport) funktion symbolit viedään ulkoiseen moduuliin. Nämä eivät ole pakollisia, mutta tehostavat ohjelman toimintaa. Yleensä nämä kätketään jonkin sanan taakse C/C++ kielen #define makrolla.

Funktion ensimmäisen ja toisen parametrin tyypit HINSTANCE ja LPWSTR on määritelty windows.h tiedostossa. HINSTANCE on moduulin kahva ja LPWSTR on leveä merkkijono. Kolmannen parametrin tyyppi on itse luotu funktio-osoitintyyppi ja sen arvoksi annetaan funktio-osoitin kirjastoa käyttävästä ohjelmasta.

```
[DllImport("Luovaphoneapi.dll", EntryPoint="ExpInitPhoneApp")]
static extern void InitPhone(
    IntPtr Apphandle,
    [MarshalAs(UnmanagedType.LPWStr)]
    string AppName,
    [MarshalAs(UnmanagedType.FunctionPtr)]
    CallbackDelegate CallbackInstance
);
```

s

Kuva 20 ExpInitPhone funktio tuonti C# ohjelmaan

Funktiota tuodessa C# ohjelmaan wrapperin funktion parametrien tyypit eivät aina täysin vastaa C#:n tyyppejä. Jokainen parametri vaatii tyyppimäärityksen jotta tiedetään millaista tietoa muuttuja sisältää. Asian hankaloittamiseksi C:ssä on monia eri kokoisia versioita samasta tietotyypistä, kuten kokonaisluvun short, int ja long versiot, sekä näiden 32- ja 64-bittiset versiot.

Kaikkia kahvoja voidaan käyttää IntPtr tyyppisinä, mutta merkkijonoja on monenlaisia eikä funktio-osoitin käänny suoraan delegaatiksi. Tuotavat funktiot esitellään sellaisiksi, miltä ne halutaan näyttävän C#- ohjelmassa, .NET:in InteropServices olettaa muuttujien tyypit esittelyn mukaisiksi ja järjestelee ne automaattisesti. Oletuksesta poikkeaville tietotyypeille annetaan ominaisuus, MarshalAs, joka kertoo mitä tyyppiä saapuva parametri on. Näin ohjelma osaa muuttaa saapuvan tyypin oikein. Ilman näitä ominaisuuksia osa välitetystä tiedosta voi pudota pois. Esimerkiksi merkkijonot on C:ssä toteutettu merkkiosoittimella, char*, ja InteropServices olettaa kaikkien C kielestä saapuvien merkkijonojen olevan tätä tyyppiä. TAPI puolestaan käyttää merkkijonoille tyyppimääritettyä LPWSTR, joka on leveämerkkiosoitin, wchar_t*. Tuloksena jotkut LPWSTR merkeistä vaihtuu tai katoaa kokonaan ja merkkijono ei enää ole oikea. Wrapperin on mahdollista välittää vain näitä oletustyypin merkkijonoja,

mutta koska TAPI käyttää vain LPWSTR tyypin merkkijonoja, täytyisi jokainen merkkijono järjestelmmä joka tapauksessa, koska C:ssä char on kahdeksan bittiä ja wchar_t 16 bittiä.

Suunniteltu viestinvälitys- ja käsittelytapa aiheuttaa ongelman käyttöliittymän puolella. Wrapper luo uuden säikeen CLR:n hallitseman muistialueen ulkopuolelle ja tämä säie kutsuu käyttöliittymän funktiota. CLR estää kutsun näihin funktioihin, koska kutsu tulee hallitsemattomasta koodista. Ongelma kierretään Invoke- funktion avulla, jolla ajettavalle funktiolla luodaan uusi hallittu säie. Suorituksen jälkeen mahdolliset palautusarvot annetaan wrapperille ja säie lopetetaan. Hallitsematon koodi siis kutsuu hallitun koodin säikeenluotifunktiota ja antaa tälle parametrina funktion, jonka se haluaa ajaa.

7.2.4 Alphan testaus

Suurin ongelma ohjelmaa käyttäessä oli taustalla kummitteleva käyttöjärjestelmän oma puhelinsovellus cprog ja sen vankkumaton tahto omia kaikki laitteeseen saapuvat puhelut. TAPI:lle voi määrittää eri sovelluksien prioriteetin saapuviin puheluihin, mutta tämä ei vaikuttanut cprogin kykyyn poimia saapuva puhelu ennen kuin viesti saapuvasta puhelusta edes päätyi wrapperille asti. Karkein ja nopeimmin toteutettava ratkaisu on etsiä ohjelman kahva prosessi listasta sen nimen perusteella ja sammuttaa se. Järjestelmä ei kuitenkaan halua cprogin pysyvän suljettuna ja käynnistää sen uudestaan lyhyen ajan kuluttua, joka tarkoittaa jatkuvaa prosessilistan uudelleen tarkistusta ja cprogin sammutusta. Jatkuva ohjelman uudelleenkäynnistys ja mahdollisesti pitkänkin prosessilistan läpikäynti syö turhaan mobiililaitteen rajallisia tehoja, vaikkakin tämän vaikutusta HOTI:n muuhun toimintaan ei tutkittu. Parempi tilapäinen ratkaisu ongelmaan on etsiä cprogin ikkunan kahva ja piilottaa ikkuna. Poissa näkyvistä, poissa mielestä ratkaisu siis. Ikkunan kahva voidaan etsiä CLR:n avulla sen nimen perusteella. Ikkunan kahva välitetään wrapperille, joka piilottaa ikkunan ShowWindow- funktiolla. Parametreina ikkunan kahva HWND ja piilotusparametri SW_HIDE.

7.3 Beta versio

Beta version on ensimmäinen valmis versio ohjelmasta, joka sisältää kaikki lopulliset toiminnot, mutta vaatii testausta. Betassa voi vielä olla bugeja ja toiminnot voivat vaatia pientä hienosäätöä, mutta mitään uutta sovellukseen ei enää pitäisi tulla. Tässä vaiheessa projektin tarvemäärittelyn muuttaminen tai virheiden korjaaminen voi olla hyvin kallista.

7.3.1 Sovellussuunnitelman tarkistus

Viestinvälitys wrapperin ja käyttöliittymän välillä toteutetaan bittiflagien sijasta luetteloilla ja merkkijonoilla. Viestejä varten wrapperiin luodaan kaksi luetteloa, yksi viestin tyypille ja yksi puhelun tiloille. C ja C# käyttävät samaa syntaksia luetteloille, joten ne voidaan kopioida suoraan otsikkotiedostosta. Viestien tyypit ovat yksiselitteisiä puhelun uutta tilaa lukuun ottamatta, jota tarkentamaan tarvitaan toinen muuttuja. Viestin yhteydessä välitetään myös selkokielinen merkkijono, joka kertoo käyttäjälle selkokielisesti mitä on tapahtunut.

Puhelutietojen esittämiseen tuli muutos TAPI:n toiminnan vuoksi. Puhelun tiloihin ei voi luottaa ja viesti puhelun kytkemisestä voi saapua vaikka puhelu vasta hälyttää. Vastapuolen vastatessa puheluun puhelu vain kytkeytyy ilman mitään viestiä. Puhelun hylkääminen kuitenkin herättää viestin. Tästä puutteesta johtuen puheluiden kestoa ei pystytä tarkasti laskemaan. TAPI kuitenkin antaa puhelu kytketty-viestin soiton alussa jolloin tiedetään puhelun oikeasti hälyttävän. Laskenta voidaan aloittaa tästä, mutta puhelun kestossa on hälytysajan pituinen virhe. Tästä puutteesta johtuen puheluiden kesto siirtyy ylimääräiseksi ominaisuudeksi.

7.3.2 Lopullinen TAPI wrapper

Beta versiossa wrapperiin lisätään linjalaitteen lisäksi puhelinlaite ja siihen liittyvä viestin- ja virheenkäsittelijät. Toiminnaltaan nämä ovat samanlaisia kuin linjalaitteen vastaavat. Puhelinlaitetta koskevat viestit haetaan TAPI:lta phoneGetMessage- funktiolla ja sen palauttamasta PHONEMESSAGE tietueesta tutkitaan dwMessageID, joka kertoo viestin tyypin. Virheen

sattuessa virhekoodi lähetetään puhelinlaitteen omalle virheenkäsittelijälle, joka lähettää käyttöliittymälle selityksen virheestä merkkijonona ja tekee tarvittavat toiminnot virheestä palautumiseksi.

7.3.3 Lopullinen Käyttöliittymä

Beta versiossa uusia toteutettavia käyttöliittymän elementtejä ovat puhelinluettelo ja puhelutietojen selain. Molemmat käyttävät HOTI:n omaa listaa. Lista ottaa vastaan listausalkiosta perittyjä luokkia, joka sisältää kuvan ja kaksi tekstikenttää. Lista näyttää kuvan elementin vasemmassa laidassa ja tekstirivit päällekkäin kuvan oikealla puolen.

Puhelinluettelossa käytettävistä elementeistä näytetään puhelinnumero ja siihen liitetty tunnus. Elementin kuvaa painamalla puhelin soittaa elementin numeroon. Puhelutiedoissa puhelusta tallennetaan puhelinnumero, soiton alkuaika ja kesto. Luettelo sisältää myös kolme suodinta näytettävien puhelinnumeroiden suodattamiseen: Asiakkaat, tiimi ja omat. Asiakkaat lista sisältää puhelinta käyttävän työntekijän asiakkaiden puhelinnumerot, tiimi-listalla on työntekijän tiimin numerot ja omat listaan voi itse lisätä tarpeelliseksi katsomiaan numeroita.

Puhelutietolistassa näytetään soittajan puhelinnumero tai puhelinluettelosta haettu tunniste ja soittoaika. Tekstikenttiä painamalla soittoaika vaihtuu soittajan numeroksi ja takaisin. Kuvana käytetään punaista tai vihreää nuolta, joka osoittaa puhelimeen tai siitä poispäin. Nämä kertovat puhelun suunnan ja vastattiinko siihen. Kuvaa painamalla puhelin soittaa elementtiin liitettyyn numeroon, kuten puhelinluettelossakin.

7.3.4 Lopullinen yhteistoiminta

Wrapperin ja käyttöliittymän välistä yhteistoimintaa ei kehitetä pidemmälle. Wrapperin jo paljastamat funktiot ja viestejä palauttava callback ovat riittävät lopulliseen toteutukseen. On turhaa kehittää monipuolisempaa ja hienostuneempaa toimintaa kun siitä ei ole selvää hyötyä. Yksinkertaisempi malli on helpompi hallita ja vähemmän virhealtis. Lisäksi vanha toteu-

tus ei vaadi ulkoista moduulia osallistumaan wrapperin toimintaan, jolloin se voi toimia itsenäisesti.

Asiakas ei käsittele mitään wrapperin sisäisiä muuttujia eikä wrapper oleta asiakkaalta toimia minkään viestin suhteen. Asiakas käsittelee pelkästään viestit ja toimii niiden pohjalta parhaalla katsomallaan tavalla.

8 YHTEENVETO

Puhelintoiminnot ovat hyvin rajattu alue sovelluskehityksessä. Niitä tarvitaan vain matkapuhelimissa ja jokainen puhelin sisältää jo valmistajan taholta nämä toiminnot. Aiheen rajoittuneisuudesta kertoo myös tiedon vähyys internetissä. Tässä työssä käytetty ohjelmointirajapinta on Microsoftin kehittämä ja silti sille löytyi projektin aikana vain kaksi yhteisöä, MSDN foorumit ja Microsoftin ylläpitämä postituslista. Tiedon hakua hankaloitti myös rajapinnan yleinen nimi, jota myös muut toimittajat käyttävät.

Tulevaisuus näkymiä tälle alueelle voisi arvioida hieman valoisammiksi kuin mitä ne olivat kehitystyön aikaan vuonna 2009. Uutisten mukaan vuonna 2011 Microsoftin osuus älypuhelinkaupasta on vain kaksi prosenttia, mutta osuuden odotetaan kasvavan Nokian uusien Windows Phone 7 laitteiden myötä. Kehitystä käyttöjärjestelmissä on kuitenkin tapahtunut paljon kahdessa vuodessa ja uudet versiot voivat tarjota paremmat työkalut puhelintoimintojen käyttöön kolmannen osapuolen ohjelmissa, eikä tässä työssä kehitettyä wrapperia enää tarvita. Nykyversiot ovat myös paljon lähempänä työpöytäjärjestelmiä, joten ne varmasti jossain vaiheessa mahdollistavat COM objektien käytön. Tällaisen järjestelmän myötä ei tarvitse enää käyttää TAPI 2 kirjastoja ja kaikki voidaan tehdä Microsoftin suosimalla C# kielellä.

Jälkikäteen katsoen sovellus olisi voitu suunnitella ja toteuttaa paremmin. Nämä parannukset kärjistyvät kahteen asiaan: TAPI:sta käyttetty versio ja viestinkäsittelyyn. Suunnittelussa hosuttiin tiedonhaun suhteen ja päädyttiin toteamukseen, jossa .NET Compact ei tukisi COM yhteistyötä vaikka sen hetkinen versio sitä jo tuki. Tämä seikka pakotti erillisen wrapperin kehitykseen ja yhteistoimintaan kahden eri ohjelmointikielen välillä. Yhteistoiminta puolestaan vaati uusien ohjelmointimenetelmien opettelua ja osittain hankaloitti kehitystyötä. Viestinvälityksen kanssa käytetty menetelmä ymmärrettiin myös väärin, jonka vuoksi ohjelmasta tehtiin monisäikeinen. Monisäikeisyys pakotti jälleen uusien menetelmien ja teknologioiden opetteluun ja loi uusia ongelmia ja haasteita kehitystyölle.

Toisaalta, vaikka toteutus ei ollutkaan optimaalinen, antoi se myös enemmän eväitä tulevaisuutta varten. Ilman näitä ratkaisuja kehitystyö olisi mahdollisesti ollut yksinkertaisesti eikä niin opettavainen. Tehdyt ratkaisut pakottivat tutustumaan ja opettelemaan sellaisia mene-

telmiä kuin delegaatit, yhteistoiminta, järjestely, rinnakkaisohjelmointi, Win32 API:en käyttö ja muistinhallinta.

Työharjoittelu päättyi kun sain kehitystyön valmiiksi. Vaikka moduuli liitettiinkin HOTI:in projektin aikana, en ehtinyt näkemään sitä varsinaisessa käytössä. Kävin puoli vuotta harjoittelun jälkeen käymässä Luovaliikkeellä ja kysyin myös kehittämäni moduulin nykytilasta. Moduuliin oli tehty pieniä muutoksia, mutta oli käytössä pääosin sellaisena kuin olin sen itse toimittanut.

LÄHTEET

CLR via C#, Jeffery Richter. 2006, Microsoft Press.

Concurrent programming, Actors, SALSA, Coordination abstractions. Carlos Varela, 22.3.2007. RPI. Web-dokumentti. Saatavilla: http://www.cs.rpi.edu/academics/courses/spring07/dci/SALSA-Concurrency.pdf

C#, Christopher Wille. 2001.

Akadia AG, 2002. Delegates and Events in C# / .NET. Web-dokumentti Saatavilla: http://www.akadia.com/services/dotnet_delegates_and_events.html.

Olio-ohjelmointi C++ - OOAD, Jesse Liberty, kääntäjä Kai Kurki-Suonio, 2001. ISBN 951-826-117-2.

Microsoft Developer network, MSDN. www.microsoft.com/msdn

Micorsoft. MSDN .NET Compact Framework. Web-dokumentti saatavilla: http://msdn.microsoft.com/en-us/library/aa446497.aspx

TAPI dokumentaatio. http://msdn.microsoft.com/en-us/library/ms734273(v=vs.85).aspx

Marc Clifton, Concurrent programming – a primer. 3.1.2008. Web-dokumentti saatavilla: http://www.codeproject.com/KB/Parallel_Programming/ConcurrentProgrammingI.aspx

Sergey Ignatchenko, 2010 a. Single-Threading: Back to the Future? Overload Journal #97. Web-dokumentti. Saatavilla: http://accu.org/index.php/journals/1634

Sergey Ignatchenko, 2010 b. Single-Threading: Back to the Future? (Part 2). Overload Journal #98,8/2010. http://accu.org/index.php/journals/1679

Richard S. Hall, Dr. Concurrent Programming. 24.4.2008. Web-dokumentti saatavilla: http://www.cs.tufts.edu/comp/150CCP/assignments/exam-solution.pdf

M. Madiajagan, and B. Vijayakumar. Interoperability in Component Based Software Development. World Academy of Science, Engineering and Technology 22. 2006. web-dokumentti saatavilla: http://www.waset.org/journals/waset/v22/v22-13.pdf

Funktiorunkoja

```
lstrcpy(
        szDialablePhoneNum, (LPTSTR)((LPBYTE) lpTransOutput +
        lpTransOutput->dwDialableStringOffset)
);
```
tiedon luku muistista offset arvolla

```
memcpy(
        (LPBYTE) lpCallParams + lpCallParams->dwDisplayableAddressOffset,
        (LPBYTE) lpTransOutput + lpTransOutput->dwDisplayableStringOffset,
        lpTransOutput->dwDisplayableStringSize
);
```
tiedon luku muistista offset-size parilla

```
LONG WINAPI lineInitialize(
        LPHLINEAPP lphLineApp,
        HINSTANCE hInstance,
        LINECALLBACK lpfnCallback,
        LPCSTR lpszAppName,
        LPDWORD lpdwNumDevs
);
```
Vanha TAPI linjalaitteen alustus

```
LONG WINAPI lineInitializeEx(
        LPHLINEAPP lphLineApp,
        HINSTANCE hInstance,
        LINECALLBACK lpfnCallback,
        LPCSTR lpszFriendlyAppName,
        LPDWORD lpdwNumDevs,
        LPDWORD lpdwAPIVersion,
        LPLINEINITIALIZEEXPARAMS lpLineInitializeExParams
);
```
Laajannettu TAPI linjalaitteen alustus

```
LONG WINAPI lineNegotiateAPIVersion(
        HLINEAPP hLineApp,
        DWORD dwDeviceID,
        DWORD dwAPILowVersion,
        DWORD dwAPIHighVersion,
        LPDWORD lpdwAPIVersion,
        LPLINEEXTENSIONID lpExtensionID
);

LONG WINAPI lineGetDevCaps(
        HLINEAPP hLineApp,
        DWORD dwDeviceID,
        DWORD dwAPIVersion,
        DWORD dwExtVersion,
        LPLINEDEVCAPS lpLineDevCaps
);
```

lineNegotiateAPIVersion ja LineGetDevCaps prototyypit.

```
typedef struct linemessage_tag {
        DWORD           hDevice;
        DWORD           dwMessageID;
        DWORD_PTR       dwCallbackInstance;
        DWORD_PTR       dwParam1;
        DWORD_PTR       dwParam2;
        DWORD_PTR       dwParam3;
} LINEMESSAGE, *LPLINEMESSAGE;
```

TAPI:n LINEMESSAGE tietueen määrittely

```
LONG WINAPI lineMakeCall(
        HLINE           hLine,
        LPHCALL         lphCall,
        LPCSTR          lpszDestAddress,
        DWORD           dwCountryCode,
        LPLINECALLPARAMS const lpCallParams);
```

LineMakeCall prototyyppi.

```
DwMessageID = LINE_APPNEWCALL
        hDevice = hLine
        dwCallbackInstance = dwInstanceData
        dwParam1 = dwAddressID
        dwParam2 = hCall
        dwParam3 = dwPrivilege
```

Viesti saapuvasta puhelusta