Ville Koskiniemi

VBA-OHJELMOINNIN MAHDOLLISUUDET RAPORTOINNIN JA ETÄHUOLLON KEHITYKSESSÄ

Tietotekniikan pro gradu -tutkielma

Sulautettujen järjestelmien linja

5.12.2005

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Ville Koskiniemi

Yhteystiedot: vilkoski@cc.jyu.fi

Työn nimi: VBA-ohjelmoinnin mahdollisuudet raportoinnin ja etähuollon kehityksessä

Title in English: Possibilities of VBA programming with developing reporting and remote diagnostics

Työ: Pro gradu -tutkielma

Sivumäärä: 67+25

Linja: Sulautetut järjestelmät

Teettäjä: Jyväskylän yliopisto, tietotekniikan laitos

Avainsanat: tietotekniikka, pro gradu, VBA, Visual Basic, BASIC, ohjelmointi, Excel, Word, raportointi, etähuolto

Keywords: computer science, Master's thesis, VBA, Visual Basic, BASIC, programming Excel, Word, reporting, remote diagnostics

Tiivistelmä: VBA-ohjelmointia käytetään yleisesti MS Office –paketin ohjelmien yhteydessä mitä erilaisimpiin käyttötarkoituksiin. VBA-ohjelmointi on makrojen ominaisuudessa tuttua melkein kenelle tahansa MS Excel –ohjelman käyttäjälle. Tämän tutkielman tarkoituksena on esitellä makrojen takana olevaa ohjelmointikieltä – VBA:ta – ja sen ominaisuuksia sekä kertoa miksi ja miten sitä käytetään raportoinnin ja etähuollon kehittämisessä.

Abstract: VBA programming is generally used for numerous purposes in relation with MS Office applications. VBA programming is familiar as macros to almost anyone who has used MS Excel. This thesis is supposed to introduce the real programming language - VBA

– and it’s features behind the macros. Thesis will also tell why and how VBA is typically used for developing reporting and remote diagnostics.

Esipuhe

Opintoni ovat tätä tutkielmaa kirjoittessani kestäneet hieman yli viisi vuotta. On ollut hienoa viettää tuo aika Jyväskylässä yhdessä parhaimpien ystävieni kanssa harrastaen, keskustellen ja joskus jopa hieman huvitellen. Oman itsensä reflektointi vaatii sosiaalisen verkon toimivuutta ja olen ollut erittäin onnekas saadessani viettää aikaa reilujen, positiivisten ja aktiivisten ihmisten parissa. Kiitos teille olemassaolostanne ja huumorintajustanne – toivottavasti yhteys teihin ei katkea koskaan.

Ilman oikeanlaista ohjausta valitettavan moni ajautuu nyky-yhteiskunnassa jo varhain väärille tai vaikeille urille. Kiitokset siis myös perheelleni, joka on mahdollistanut kasvuni ja kehitykseni juuri tällaiseksi yksilöksi kuin nyt olen.

Tähän tutkielmaan liittyen erityiset kiitokset kuuluvat Metso Paper Oy:n elinkaari- ja etäpalvelujen kehitysryhmälle, jonka työllistävän vaikutuksen ansiosta minulla ollut mahdollisuus tutustua VBA:lla tehtyihin ratkaisuihin ja niihin liittyviin ongelmanratkaisun näkökulmiin.

Jyväskylässä, 5.12.2005, Ville Koskiniemi

Termiluettelo

ActiveX	on toinen nimi Microsoftin Windows-käyttöjärjestelmässä käytetylle Component Object Model (COM) tekniikalle. COM-komponentit ovat uudelleenkäytettäviä ohjelmistokomponentteja.
ADO	(ActiveX Data Object) on eräs Visual Basicin tietokantarajapinnoista. Microsoft suosittelee tämän rajapinnan käyttöä vanhempien DAO:n ja RDO:n sijasta.
ALGOL	(ALGOrithmic Language) on korkean tason ohjelmointikieli, joka kehitettiin kansainväliseksi algoritmien ilmaisukieleksi ihmisten ja koneiden välille. ALGOL 60 oli yksinkertainen ja Euroopassa laajalti käytetty kieli. ALGOL 68 puolestaan oli monimutkaisempi ja vähemmmän käytetty, mutta sen voidaan katsoa olevan yksi Pascal-kielen lähteistä.
API	(Application Program Interface) tarkoittaa sovellusohjelmointirajapintaa.
DAO	(Data Access Objects) on eräs Visual Basicin tietokantarajapinnoista.
Data General Nova	oli suosittu 16-bittinen minitietokone, jota myytiin 50000 kappaletta. Sen valmistaja oli Data General Corporation (Westboro, MA), joka oli yksi ensimmäisistä minitietokoneyrityksistä.
De facto	on latinankielinen käsite, joka tarkoittaa "käytännössä" tai "yleisesti", yleensä de juren ("lain mukaan")

	vastakohtana. Asiat, jotka ovat *de facto* yleensä kehittyvät autonomisesti ilman erillistä standardin, lain, säännön tai sopimuksen tekoa.
DEC	(Digital Equipment Corporation, Maynard, MA, www.digital.com) oli ensimmäinen minitietokonevalmistaja. DEC tunnetaan yleisesti myös nimellä Digital. DEC perustettiin vuonna 1957 Kenneth Olsenin toimesta.
FORTRAN	(FORmula TRANslator) on ensimmäinen korkean tason ohjelmointikieli ja kääntäjä. Sen kehitti IBM vuonna 1954 ja se oli suunniteltu matematiikan ja luonnontieteiden tarpeisiin ja on edelleenkin käytössä näillä aloilla.
IBM	(International Business Machines) on maailman suurin teknologiayritys ja tunnettu erityisesti tietokoneiden kuten suurkoneiden, raskaiden palvelimien ja PC-koneiden valmistajana.
Intellectual Property	(IP) viittaa lailliseen oikeuteen jotain ideaa, muotoa tai aineetonta asiaa kohtaan. Liittyy läheisesti tekijänoikeuteen.
Makro	on automaattisesti toistettava sarja ohjelman omia komentoja.
Metso Paper	on suurin vuonna 1999 Helsingissä perustetun metalliteollisuuskonsernin liiketoiminta-alueista. Metso Paper keskittyy kuitu- ja paperiteknologiaan, johon kuuluu paperi-, kartonki- ja jatkojalostuskoneiden sekä kuitutekniikan laitteiden valmistus.

Microsoft Corporation	(myös MS) on maailman suurin ohjelmistoalan yritys. Bill Gates ja Paul Allen perustivat yhtiön vuonna 1975. Yhtiön päämaja sijaitsee Redmondissa Washingtonissa. Microsoft on useissa tietokoneissa käytettävän Windows-käyttöjärjestelmän valmistaja ja myyjä.
OLE	(Object Linking and Embedding) -tekniikka tarkoittaa objektien linkittämistä ja upottamista kohdesovellukseen.
Palm OS	on Palm Source Inc. –yhtiön valmistama käyttöjärjestelmä PDA-laitteisiin, jota valmistetaan lukuisin eri lisenssein.
PDA	(Personal Digital Assistants) on yleisnimitys kämmenkokoisille tietokoneille.
PDP	(Programmed Data Processor) Digitalin valmistama minitietokoneperhe, jonka ensimmäinen jäsen oli 18-bittinen PDP-1 vuonna 1959. Hinta $120 000 ja myynti 50 kpl.
RCM	(Reliability Centered Maintenance) on menetelmä, jossa kunnossapitoa tehdään luotettavuuskeskeisesti. Menetelmässä kunnossapitotoimenpiteet kohdistetaan kohteen tärkeyden ja vauriotodennäköisyyden mukaan.
RDO	(Remote Data Control) on eräs Visual Basicin tietokantarajapinnoista.
Return On Investment	(ROI) on laskelma, jota käytetään laskemaan, onko ehdotettu sijoitus viisas ja kuinka hyvin se tuottaisi sijoittajalleen.

Sun Microsystems	on Yhdysvalloissa, Kalifornian Piilaaksossa päämajaansa pitävä tietokoneita, puolijohteita ja ohjelmistoja valmistava yritys.Sunin tuotteisiin kuuluvat mm. Solaris-käyttöympäristö ja Java-ohjelmointikieli.
UTF	(Unicode Transformation Format) on eräs kansainvälinen standardi, jonka tarkoituksena on luoda yhteiset pelisäännöt tietokoneille tallennettavan datan koodaukseen liittyen. UTF-16:n tarkempi kuvaus löytyy RFC2781:stä.
VBA	(Visual Basic for Applications) on tulkittu, proseduraalinen ohjelmointikieli. Lähdekoodia suoritetaan samaan aikaan, kuin varsinainen ohjelma päättelee sen tarkoitusta.
XML-kieli	(Extensible Markup Language) on merkkauskieli, jolla tiedon merkitys on kuvattavissa tiedon sekaan. XML muistuttaa HTML-kieltä, mutta XML-kieli ei kuitenkaan ole tarkoitettu sivunkuvauskieleksi kuten HTML vaan sillä kuvataan tiedon rakenne ilman ennalta määrättyjä koodeja.

Sisältö

1 JOHDANTO 1

2 TERMISTÖN JA TOIMINTAYMPÄRISTÖN ESITTELYÄ 3

2.1 MICROSOFT OFFICE 3
2.2 MICROSOFT EXCEL 4
2.3 MICROSOFT WORD 5
2.4 LOTUS NOTES JA ACTIVESYNC 6

3 HISTORIA – BASICISTÄ VBA:HAN 7

3.1 BASIC 7
3.2 VISUAL BASIC 11
3.3 VISUAL BASIC FOR APPLICATIONS 15
3.4 KOKONAISKUVA 16

4 VBA-OHJELMOINTI 17

4.1 HAITAT – MITÄ RAJOITUKSIA VBA:N KÄYTÖLLÄ ON? 17
4.2 EDUT – MIKSI KÄYTTÄÄ VBA:TA? 18
4.3 YLEISKATSAUS ITSE KIELEEN 19
4.3.1 VBA:n toimintaperiaate ja syntaksi 21
4.3.2 Tietotyypit 24
4.3.3 Aliohjelmien rakenne 25
4.3.4 Ohjausrakenteet 26
4.3.5 Virheenkäsittelymekanismit 27
4.3.6 VBA-koodin muodostaminen 29
4.4 TYYPILLINEN KÄYTTÖ 30
4.5 OHJELMOINTIYMPÄRISTÖ 30
4.5.1 VBA:n objektikirjastot 32
4.6 KOODIN OPTIMOINTI 33

5 ESIMERKKEJÄ VBA:N KÄYTÖSTÄ 36

5.1 RCM-TYÖKALU 36
5.1.1 UserFormien koon skaalautuminen käytössä olevan resoluution mukaan 37
5.2 LOMAKETIETOJEN KERÄÄMINEN JA LÄHETTÄMINEN TIETOKANTAAN 38
5.2.1 Lomakekenttien sijoittaminen pohjaan & tiedon hakeminen 38
5.2.2 Merkkijonon tai XML-tiedoston muodostaminen ja tiedon lähettäminen Notes –postikannan avulla 39
5.3 MITTAUSTIETOJA SISÄLTÄVÄN TIEDOSTON LUKEMINEN JA GRAAFIEN PIIRTÄMINEN 40
5.4 ROI-SOVELLUS 40

5.5 HUOMIOITA VBA- JA OFFICE-YMPÄRISTÖISTÄ JA ARVIOINTI TEHDYISTÄ RATKAISUISTA 43
6 VERTAILUKOHTINA C++ JA JAVA 46
6.1 C++ 46
6.2 JAVA 46
6.3 VISUAL BASIC, JAVA JA C++ 47
7 TULEVAISUUS 50
7.1 MITÄ VISUAL BASICIN EPÄVARMA TULEVAISUUS TARKOITTAA VBA:N OSALTA? 50
8 YHTEENVETO 52
LÄHTEET 54
LIITTEET 58
LIITE 1. LUVUN 5.2.2 ONGELMAAN LIITTYVÄ VBA-KOODI KOKONAISUUDESSAAN 58
LIITE 2. LUVUN 5.3 ONGELMAAN LIITTYVÄ VBA-KOODI KOKONAISUUDESSAAN 58
LIITE 3 MICROSOFT EXCELIN WORKSHEETS- JA CHARTS-OBJEKTILUOKKIEN RAKENNE (LUKU 4.3.1) 58

1 Johdanto

Kolmannen opiskeluvuoden kesällä sain töitä Metso Paper Oy:n palveluksessa ja oikeastaan siinä oli alkusykäys tämän tutkielman kirjoittamiselle. Paperikonehuollon etädiagnostiikassa ja sen raportoinnissa käytetään apuvälineenä paljon Microsoft Excel ja Microsoft Word ohjelmistoja. Paperikoneiden ollessa kyseessä, datamäärät ovat varsin suuria, ja näin ollen mahdottomia hallita luotettavasti ilman automatisointia. MS Excel on taulukkolaskentaohjelmana omaa luokkaansa [5, s. 5] ja sanotaan, että on turhaa keksiä jo olemassa olevaa uudelleen. On kuitenkin myös niin, että MS Excel –ohjelmiston valmiit ominaisuudet eivät aina riitä vastaamaan käytännön tarpeita. Tätä varten Microsoft on kehittänyt oman ohjelmointikielen – VBA:n, joka juontaa juurensa Visual Basicistä. VBA mahdollistaa tehtävien automatisoinnin sekä omien toimintojen sisällyttämisen sovellukseen. Tämä avaa uusia mahdollisuuksia tiedon keräämiseen, järjestämiseen ja raportointiin.

En itse ole koskaan ollut, ja tuskin tulen olemaankaan, erityisen kiinnostunut ohjelmoinnista. Työtehtävieni painottuminen erilaisiin VBA-projekteihin on kuitenkin vaikuttanut ajatteluuni siten, että en enää ajattele ohjelmoinnin olevan jotain, mitä en mistään hinnasta tekisi. Itse asiassa olen työtehtävieni innoittamana rakennellut erilaisia hyötysovelluksia myös kotikäyttöön.

Aloitin tietotekniikan opinnot Jyväskylän yliopistossa vuonna 2000 oikeastaan sattuman kautta. On ollut ilo huomata, että ilman harrastuneisuutta alalle olen kuitenkin onnistunut suoriutumaan hyvin ja osin jopa erinomaisesti minulle annetuissa tietoteknisissä opiskelu- ja työtehtävissä. Kiittäminen tästä on tietotekniikan perusopetusta, joka antaa opiskelijalleen avaimet omatoimiseen ongelmanratkaisuun. Loogisen matemaattisen ajattelutavan sisäistäminen ja peräänantamaton työnteko ovat ainakin omalla kohdallani toimineet niin, että voin todeta olevani tilanteessa, jossa kyky oppia, omaksua ja ymmärtää uutta on vähintään aikamme teknisen yhteiskunnan vaatimalla tasolla.

VBA itsessään on melko tuntematon ohjelmointikieli siksi, että sillä ei ohjelmoida käännettäviä ohjelmia. VBA:ta käytetään lähinnä MS Office –ohjelmistojen kanssa automatisoimaan työtehtäviä tai lisäämään ohjelmien ominaisuuksia. Lähes kaikki MS

Excel –sovellusta joskus käyttäneet tunnistavat kyllä makrot, mutta se, että makrojen kirjoituskielenä on VBA, on useimmille uutta tai hyödytöntä tietoa. Sen sijaan VBA:n edeltäjät BASIC ja Visual Basic ovat tuttuja jossain yhteydessä ainakin suurimmalle osalle kehittyneitä käyttäjiä.

Tässä tutkielmassa pyrin avaamaan verhoa makrojen ja niiden takana olevan ohjelmointikielen välillä esittelemällä kielen historiaa, tekniikkaa ja esimerkkejä. Tutkielma rakentuu siten, että sen toisessa luvussa esittelen lukijalle tyypilliseen tekniseen toimintaympäristöön sekä käytettävään nimistöön liittyviä asioita. Kolmannessa luvussa käsittelen VBA:n kehityskulkua esitellen varsinkin sitä edeltäviä ohjelmointikieliä. Neljännessä luvussa esittelen VBA-ohjelmointiin liittyviä etuja sekä rajoitteita sekä kerron enemmän varsinaisen VBA-ohjelmointikielen ominaisuuksista ja sen ohjelmointiympäristöstä. Tutkielmani viidennessä luvussa keskityn esimerkkeihin, jotka pohjautuvat Metso Paper Oy:n palveluksessa tekemiini ratkaisuihin erilaisiin raportoinnin ja etähuollon toimenpiteisiin liittyen. Esittelen muutamia tapoja, miten VBA:ta käyttäen on ratkaistu tiedon keräämiseen, kokoamiseen ja visualisointiin liittyvää problematiikkaa. Lisäksi kerron VBA:n hyödyntämisestä triviaalimpien (leikkaa-liimaa, etsi-vertaa) tehtävien yhteydessä. Arvioin myös tehtyjen ratkaisujen hyvyyttä ja pohdin keinoja, joilla ratkaisuista saataisiin tehokkaampia ja mahdollisten virhetilanteiden määrää pystyttäisiin karsimaan. Kuudennessa luvussa teen vertailun toiseen tulkattavaan ohjelmointikieleen – Javaan sekä alkuperäiseen ohjelmoinnin opetuskieleeni C++:aan. Seitsemännessä luvussa arvioin VBA:n tulevaisuutta ja kahdeksannessa luvussa teen yhteenvedon tutkielmassa esiintyvistä asioista.

Useassa kohdassa tutkielmaani olen puhunut Visual Basicista VBA:n sijasta. Kuten tiedämme, on tietotekniikka tieteenalana varsin nuori ja VBA ohjelmointikielenä vieläkin nuorempi. Niinpä soveltuvaa kirjallisuutta oli yllättävänkin hankala löytää varsinkin VBA:n osalta. Tästä johtuen Visual Basic (VB), viitaten VB:n versioihin 1-6, esiintyy tutkielmassa välillä synonyyminä VBA:lle.

2 Termistön ja toimintaympäristön esittelyä

Esittelen seuraavassa useimmille jo ennestään tuttuja toimisto- ja työryhmäohjelmia, joita esimerkiksi Metso Paper Oy yleisesti käyttää toimistoissaan.

2.1 Microsoft Office

Microsoft Office on Microsoftin valmistama, yksi maailman eniten käytetty toimisto-ohjelmistopaketti Windows- ja Mac OS X -käyttöjärjestelmille. Microsoft Officen kehitystyö aloitettiin vuonna 1983 Apple Macintosh tietokoneille. Myöhemmin vuonna 1993 Microsoft kehitti Office-paketista Windows-version [10].

Toimisto-ohjelmien lisäksi Microsoft Office sisältää nykyään myös palvelinohjelmiston ja verkkopohjaisia palveluja. Niinpä sen nykyinen nimitys kuuluukin Microsoft Suite [10]. Tässä tutkielmassa puhutaan kuitenkin enimmäkseen Microsoft Officesta (myös MS Office).

MS Office ilmestyi ensimmäisen kerran 90-luvun alkupuolella ja oli alunperin ainoastaan markkinointitermi sovelluksille, joita aikaisemmin oli myyty ja markkinoitu erikseen. Päämyyntiargumentti oli se, että paketin ostaminen tulisi merkittävästi halvemmaksi kuin jokaisen sovelluksen ostaminen erikseen. Ensimmäinen versio MS Officesta sisälsi Word, Excel, ja Powerpoint -sovellukset. Lisäksi Pro-versio Officesta sisältsi myös MS Access –sovelluksen. Vuosien mittaan Office-sovellukset ovat kasvaneet selvästi lähemmäksi toisiaan teknisestä näkökulmasta katsoen. Jaettuja ominaisuuksia ovat muun muassa oikeinkirjoituksen tarkistus, OLE dataintegraatio-ominaisuus ja Microsoft VBA-ohjelmointikieli [10].

MS Office –ohjelmistoja pidetään yleisesti *de facto* standardina toimisto-ohjelmille. MS Office –ohjelmissa on paljon ominaisuuksia, joita toisista paketeista puuttuu – tämä tietysti pätee myös toiseen suuntaan [10].

2.2 Microsoft Excel

Microsoft Excel (myös MS Excel tai Excel) on taulukkolaskentaohjelma, jonka peruskäyttöliittymä on yksinkertainen taulukko, jossa jokaisella solulla on oma osoitteensa rivin ja sarakkeen muodossa. Excelin ominaisuuksiin kuuluvat intuitiivisen käyttöliittymän lisäksi, hyvät laskentaominaisuudet ja piirtotyökalut. Tähän kun lisätään vielä Microsoftin harjoittama aggressiiviinen markkinointistrategia, niin ollaankin mainittu suurimman syyt sille, että Excel–sovelluksella on dominoiva markkinaosuus omassa ohjelmistoluokassaan [10].

Alunperin Microsoft markkinoi taulukkolaskentaohjelmaa nimeltä Multiplan, joka oli erittäin suosittu CP/M-käyttöjärjestelmään pohjautuvissa Intelin ja Zilogin laitteissa lähinnä 1970-luvulla. MS-DOS:n aikakaudella Multiplan kuitenkin menetti mahtiasemansa Lotus 1-2-3:lle, mikä oikeastaan antoi tarkoituksen Excelin kehittämiselle. Ensimmäinen Excelin version julkaistiin Mac OS:lle vuonna 1985 ja ensimmäinen Windows-versio ohjelmasta saatiin 1987. Lotuksen kehitystyö Windowsiin siirtyessä ontui ja niinpä Excel ohitti sen myyntimäärissä vuoteen 1988 mennessä ja samalla auttoi Microsoftia saavuttamaan asemansa johtavana PC-ohjelmistojen kehittäjänä. Nykyinen Excelin versionumero on 11, joka tunnetaan paremmin nimellä Microsoft office Excel 2003 [10].

Kun Excel ensimmäistä kertaa paketoitiin mukaan Officeen vuonna 1993, uudelleensuunniteltiin Wordin ja Powerpointin käyttöliittymät vastaamaan Exceliä, joka oli tuon ajan kehittynein PC-sovellus. VBA:n ensimmäinen kehitysversio esiintyi jo vuoden 1993 Excelissä, mutta varsinaisesti VBA esiteltiin vasta vuoden 1997 Office-paketin yhteydessä [10].

Nykyään Excel on *de facto* standardin asemassa. Excel on saatavilla Windows ja Macintosh alustoille. Sen pääkilpailijat ovat OpenOffice.org Calc, Lotus 1-2-3, StarOffice ja Corel Quattro Pro [10].

2.3 Microsoft Word

Microsoft Word (MS Word, Word) on Microsoftin kehittämä maailmanlaajuisessa käytössä oleva tekstinkäsittelyohjelma, joka on saatavina useina erikielisinä versioina. Ohjelman uusimmat versiot on julkaistu osana Microsoft Office -ohjelmistoa Microsoft Windows 2000-, XP- ja Apple Mac OS X -käyttöjärjestelmille. Uusin versio on Microsoft Office Word 2003 Windowsille ja Word 2004 Macille. Vanhemmat versiot ovat toimineet MS-DOS:ssa ja SCO UNIX:ssa sekä tietenkin Windowsin ja MacOS:n vanhemmissa versioissa. Wordin kanssa samoista markkinoista kilpailevat mm. StarOffice, Corel WordPerfect, OpenOffice.org ja Apple Pages [10].

Monet Wordin konseptit ja ideat ovat lähtöisin Bravosta, joka oli Xerox PARC:n kehittämä ensimmäinen alkeellisella graafisella käyttöliittymällä varustettu tekstinkäsittelyohjelma. Bravon luoja Charles Simonyi siirtyi Microsoftin palvelukseen vuonna 1981 ja aloitti Wordin kehitystyön, joka johti ensimmäisen MS-DOS:lle suunnitellun version julkaisuun vuonna 1983. Vastaanotto ei kuitenkaan ollut mitenkään positiivinen ja WordPerfect säilytti helposti tuon ajan valta-asemansa suosituimpana tekstinkäsittelijänä. Macintoshin puolella Word kuitenkin saavutti laajan hyväksynnän ensimmäisen julkaisunsa (1985) jälkeen [10].

Yksi Wordin kohtaamista suurimmista hankaluuksista oli yritysten haluttomuus vaihtaa tuttua tekstinkäsittelyohjelmaa uuteen sen takia, että MS-DOS –käyttöjärjestelmään usein liittyivät sovelluskohtaiset, usein monimutkaiset, käskyt, joilla ohjelmaa ohjattiin [10].

Ensimmäinen Wordin versio Windowsille julkaistiin vuonna 1989. Vuonna 1990 julkaistun Windows 3.0:n ilmestymisen jälkeen Wordin myynti alkoi kohota, sillä WordPerfect ei onnistunut tuottamaan omasta tuotteestaan Windowsiin soveltuvaa versiota. Wordin versio 2 (1991) ankkuroi MS Wordin markkinajohtajaksi omalla sarallaan. Wordin myöhemmät versiot sisältävät tekstinkäsittelyn lisäksi monia uusia ominaisuuksia, kuten grafiikan tuomisen dokumentteihin ja tuen monikielisyydelle [10].

Kuten muutkin Office-paketin ohjelmat, on myös Word räätälöitävissä VBA-ohjelmointikielen avulla, vaikkakin alunperin Word käytti omaa WordBasic-kieltä, joka kuitenkin vaihdettiin VBA:han vuonna 1997.

Word–sovelluksella on dominoiva markkinaosuus tekstinkäsittelyohjelmien saralla ja sen DOC-formaattia pidetään *de facto* standardina, vaikkakin viimeisin versio sisältää myös XML-formaatin, johon Microsof on ilmoittanut jatkossa panostavansa [10].

2.4 Lotus Notes ja ActiveSync

Lotus Notes on patentoitu asiakas-palvelin ohjelmisto ja sähköpostijärjestelmä, jonka omistaa Lotus Software, joka on osa IBM Software Groupia. Se sisältää mm. ryhmäkalenterin, jaetut dokumentit ja keskustelukanavat, SMTP-pohjaisen sähköpostin sekä sisäänrakennetut turvallisuustyökalut. Sen lisäksi, että Notes on ryhmätyöohjelmisto, on se myös räätälöityjen asiakas-palvelin- ja web-sovellusten kehitysalusta [11].

ActiveSync on Microsoftin suunnittelema synkronointiohjelma Windows Mobile- ja Windows CE -käyttöjärjestelmille perustuviin laitteisiin. Versiot 3.0 ja 3.1 laajensivat ohjelmaa myös Palm OS käyttöjärjestelmälle [16].

3 Historia – Basicistä VBA:han

Ennen 1960-lukua tietokoneet olivat erittäin kalliita työkaluja, joita käytettiin ainoastaan yksittäisiin erikoistehtäviin. 60-luvulla tietokoneiden hinnat alkoivat kuitenkin pudota ja niinpä myös pienyrityksillä oli varaa niihin. Samalla myös koneteho kasvoi niin, että prosessorit olivat osan aikaa joutilaina. Tuon ajan ohjelmointikielet olivat usein kullekin koneelle ja käyttötarkoitukselle räätälöityjä, eikä niiden käyttö ollut erityisen helppoa. [18].

Seuraavassa esittelen lyhyesti VBA-ohjelmointikielen syntyä kulkien aina Basic-kielestä Visual Basicin kautta VBA-ohjelmointikieleen, joka on lähes sama kieli kuin Visual Basic 6.0.

3.1 BASIC

BASIC (*Beginner's All-purpose Symbolic Instruction Code*) on perhe korkean tason ohjelmointikieliä. Alunperin Basic laadittiin helppokäyttöiseksi ohjelmointikieleksi, joka levisi laajalle mikrotietokoneissa 1980-luvulla ja on tänä päivänä edelleen suosittua erilaisten murteidensa muodossa. Basicin kehittivät vuonna 1963 Darthmouth Gollege:n professorit John G. Kemeny ja Thomas E. Kurtz [13].

BASIC suunniteltiin opiskelijoiden tarpeita silmällä pitäen siten, että sillä olisi mahdollista kirjoittaa ohjelmia keskuskonetyyppiseen ympäristöön, jossa yhden koneen prosessoriaika jaetaan useille käyttäjille. BASIC-kielen tarkoituksena oli helpottaa ohjelmointia varsinkin käyttäjille, jotka eivät olleet niinkään kiinnostuneita nopeudesta vaan pelkästään siitä, että tietokoneen käyttö tuli mahdolliseksi [18], [8, s. 1-4], [20].

Kurtz ja Kemeny kiittelevät kirjassaan *Back to BASIC* opiskelijoiden panosta kielen alkuvaiheissa. Kemeny jopa sanoo, että heidän onnistumisensa BASICin kehitämisessä johtuu siitä, että muut käyttivät apunaan ammattilaisia, kun he puolestaan luottivat opiskelijoiden kykyihin [8, s. 5].

Kirjallisuudessa esiintyy satoja korkean tason ohjelmointikieliä alkaen 1950-luvulta. Näistä FORTRANin (1957), joka oli suunniteltu pääasiassa luonnontieteellisiin sovelluksiin ja tieteelliseen laskentaan [28, s. 37] [17, s. 81], voidaan katsoa olevan

BASICin (1964) ensimmäinen esi-isä [19, s. 81]. BASICin toiseksi esi-isäksi voidaan katsoa ALGOL 60 (1960) [18], joka oli käytössä lähinnä algoritmien ilmaisukielenä matematiikassa ja luonnontieteissä. [18] [28, s. 38]. BASICin tarkoituksena oli laajentaa FORTRANin ja ALGOLin toimintakenttää prosessoriajan jakamiseen ja myöhemmin myös tekstinkäsittelyyn sekä matriisien aritmeettisiin laskutoimituksiin [18].

Oikeastaan nimitys esi-isä on virheellinen siinä mielessä, että Kemeny ja Kurtz pikemminkin halusivat eroon FORTRANin ja ALGOLin tietyistä piirteistä kuin halusivat pitää niistä kiinni. Joitain kyseisien kielien syntaksiominaisuuksia (For..Next –silmukka, varatut sanat) otettiin kuitenkin mukaan BASICiin, niiden toimivuuden ja valmistuvien opiskelijoiden reaalisen toimintaympäristön vaatimusten takia [8, s. 7-9]

BASIC-kielen kahdeksan periaatetta ovat [8, s. 9]:

1. Kielen tulee olla helposti opittavissa aloittelijalle.
2. Kielen tulee olla yleiskäyttöinen ohjelmointikieli.
3. Kielen tulee sallia kehittyneiden ominaisuuksien lisäämisen kehittyneille käyttäjille, mutta tämä ei saa hankaloittaa noviisien toimintaa.
4. Kielen tulee olla interaktiivinen, jotta kommunikaatio tietokoneen kanssa olisi mahdollista.
5. Kielen tulee antaa selvät ja ystävälliset virheilmoitukset käyttäjälle.
6. Kielen tulee toimia nopeasti pienten ohjelmien tarpeita silmällä pitäen.
7. Kielen ei tule vaatia ymmärrystä tietokoneen raudasta (HW).
8. Kielen tulee eristää käyttäjä käyttöjärjestelmältä.

Kielen alkuvaiheissa Darthmouthin yhteistyö laitevalmistaja GE:n kanssa mahdollisti laitehankinnat, joihin heillä muuten ei olisi ollut varaa. GE puolestaan oli kiinnostunut ajanjakamisjärjestelmästä, joka sisältyi BASICin perusajatukseen [8, s. 21-23].

Kielen suunnittelijat markkinoivat tuotostaan myös Darthmouthin ympäristössä sijaitseville kouluille sillä seurauksella, että tieto BASICin olemassaolosta levisi melko laajalle ja niinpä sitä implementoitiin useiden valmistajien pieniin tietokoneisiin (DEC PDP, Data General Nova) lähinnä tulkin muodossa [18].

Vuonna 1968 ilmestyneessä artikkelissa hollantilainen Edsger Dijkstra kirjoitti GOTO-lausetta käyttävien ohjelmointikielien olevan haitallisia niin ohjelmoijan tuottavuudelle kuin koodin laadulle [9]. Vaikkakaan tässä artikkelissa ei mainita erikseen BASIC-kieltä, sekoitetaan se usein vuonna 1975 ilmestyneeseen saman kirjoittajan artikkeliin, jossa hän kritisoi useita senaikaisia ohjelmointikieliä, kuten BASIC, PL/I, COBOL ja APL [18].

BASICin todellinen läpimurto tapahtui 1975, kun Altair 8800 mikrotietokone julkaistiin. Useimmat ohjelmointikielet olivat liian isoja mahtuakseen silloisiin pieniin muisteihin, joihin käyttäjillä oli varaa. Näin ollen BASIC, joka oli hyvin tunnettu suunnitelijoiden keskuudessa, kiitos Kemenyn ja Kurtzin, sai yhä enemmän jalansijaa pienen kokonsa ansiosta. Myös Microsoft, joka siihen aikaan oli kahden miehen (Bill Gates ja Paul Allen) yritys julkisti BASICistä oman versionsa ja neuvotteli useiden tietokonevalmistajien, kuten IBM, kanssa BASIC-tulkin lisensioimisesta heidän tietokoneisiinsa. Yhtäkkiä BASIC olikin hallitsevassa asemassa oleva ohjelmointikieli, ja löytyi miljoonista koneista ympäri maailmaa [18].

Kehityksen kulkiessa vauhdilla eteenpäin luotiin myös BASICistä useita eri versioita sekä täysin uusia kieliä, jotka muistuttivat BASICiä ainostaan syntaksiltaan. Tästä ilmiöstä Kemeny ja Kurtz eivät erityisesti pitäneet ja ovat kyllä tuoneet sen myös esille kirjassaan ”Back to BASIC: the history, corruption and future of the language” [8]. 1980-luvun lopulle tultaessa tietokoneet olivat jo huomattavasti monimutkaisempia ja sisälsivät useimmiten myös graafisen käyttöliittymän. Tämä johti siihen, että BASIC kävi vähemmän sopivaksi ohjelmointiin. Samaan aikaan tietokoneiden käyttö oli siirtynyt enemmänkin tavallisten ihmisten kuin alan harrastajien käsiin ja niinpä myös ohjelmointi yleisesti tuli vähemmän tärkeäksi painopisteen siirtyessä valmiiksi kirjoitettuihin ohjelmiin. BASIC alkoi kadota kuvasta, vaikka useita eri versioina jatkoikin olemassa olemistaan [18].

BASIC sai uutta tuulta purjeisiinsa, kun Microsoft esitteli Visual Basicin (VB) vuonna 1991. Uusi kieli muistutti BASICiä enää etäisesti tuttujen avainsanojen muodossa, mutta sitä pidetään silti BASICin jälkeläisenä. VB on tällä hetkellä eräs käytetyimmistä kielistä Windows-alustalle. On arvioitu, että 70-80% kaupallisesta kehityksestä tehdään VB:llä. Microsoft loi lisäksi muunnelman Visual Basicistä, nimeltään WordBasic, ja käytti sitä MS Wordin versioissa ennen vuotta 97. VBA:n Microsoft lisäsi Exceliin vuonna 1993, Accessiin vuonna 1995 ja koko Office-pakettiin vuonna 1997 [18].

Syntaksiltaan BASIC on varsin yksinkertainen kieli. Lauseet päättyvät rivinvaihtoon ja minimaalinen syntaksi vaatii ainoastaan LET, PRINT, IF ja GOTO -komentoja. BASIC-kielen ominaispiirteisiin kuuluu myös FORTRANin kaltainen rivinumerointi, joka saattoi aiheuttaa ohjelmoijille pientä päänvaivaa alkeellisissa ohjelmointiympäristöissä. Nykyään lähes kaikki BASICin murteet ovat luopuneet rivinumeroinnista ja tunnistavat lisäksi myös uudemmista ohjelmointikielistä tutut standardirakenteet, kuten silmukat (DO, LOOP, WHILE, UNTIL, EXIT) ja valintarakenteet (ON x GOTO / GOSUB, ja SWITCH & CASE). Viimeaikaiset muunnelmat, esimerkiksi VB, sisältävät myös oliosuuntautuneita ominaisuuksia, kuten For Each –silmukkarakenteen ja jopa ominaisuuksien perintää. Muistinhallinta sujuu helposti automaattisen (vrt. Java) roskankeruun avulla, mistä tosin maksetaan veroa suoritusajoissa. [18].

BASICissä ei ole ulkoisia standardikirjastoja, kuten C:ssä tai C++:ssa. Sen sijaan tulkki sisältää laajan sisäänrakennetun kirjaston proseduureja, jotka käsittävät suurimman osan ohjelmoijan kaipaamista työkaluista perussovellusten kirjoittamiseen. BASIC eroaa monista ohjelmointikielistä siinä, että se erottelee aliohjelmat sen mukaan palauttavatko ne arvon (function) vai eivät (subroutine tai sub). BASIC ei siis tunne esimerkiksi C:stä tuttua void–tyyppiä [18].

BASIC on alusta asti ollut tunnettu hyvistä merkkijonojen käsittelyyn soveltuvista funktioista (LEFT; MID RIGHT). Tämä oli myöskin yksi suurimmista eduista verrattuna muihin varhaisiin korkean tason kieliin. Alkuperäinen Dartmouth BASIC tuki ainoastaan liukuluku- ja merkkijonotyyppejä – kokonaislukutyyppiä ei ollut olemassa. Nykyään BASICin eri murteet tukevat kaikkia yleisimpiä tietotyyppejä mukaanlukien C-tyyliset, itse

määritellyt tietuetyypit sekä erilaiset taulukot. Indeksinumeroinnin alku vaihtelee hieman murteen mukaan (joko 0 tai 1) [18].

Loogiset ja suhteelliset operaattorit, jotka on esitelty taulukossa (Taulukko 1), ovat BASIC:in murteissa varsin tyypillisiä [18] (VBA:n operaattoreista tarkemmin esimerkiksi [1, s. 92-93].):

Operaattori	Merkitys	Operaattori	Merkitys
=	yhtä kuin (vertailu) tai sijoitus	NOT	looginen negaatio
<>	eri suuri kuin	AND	looginen kertolasku
<	pienempi kuin	OR	looginen yhteenlasku
>	suurempi kuin		
<=	pienempi tai yhtä suuri kuin		
>=	suurempi tai yhtä suuri kuin		

Taulukko 1. BASICin loogiset operaattorit [18].

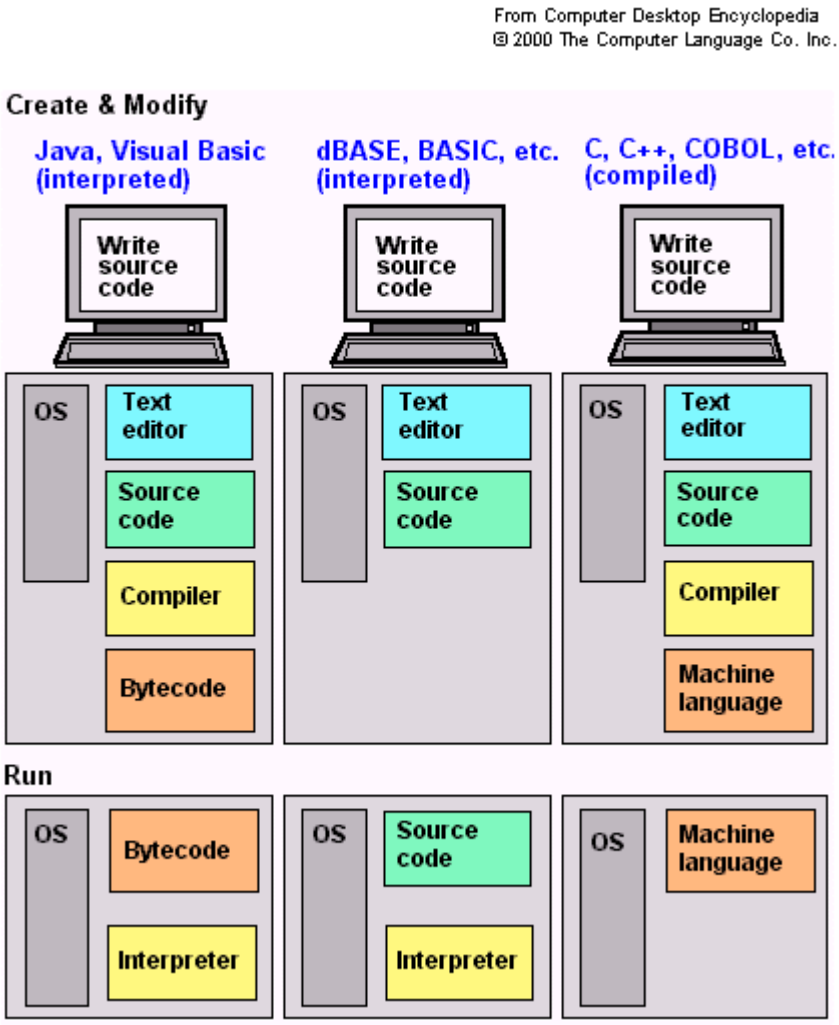
3.2 Visual Basic

Visual Basic (VB) on Microsoftin kehittämä tapahtumapohjainen ohjelmointikieli ja siihen yhdistetty kehitysympäristö. Käyttäjämäärältään VB on eräs eniten käytetyistä ohjelmointikielistä liiketoimintakäytössä. VB juontaa voimakkaasti BASIC-ohjelmointikielestä ja mahdollistaa graafisten käyttöliittymien nopean kehittämisen, nopean pääsyn tietokantoihin käyttäen joko DAO, RDO tai ADO rajapintaa sekä ActiveX

kontrollien luomisen. Ohjelmoijalla on näin mahdollisuus rakentaa sovellus suoraan VB:n itsensä luomilla komponenteilla. [14].

VB on siis versio BASIC-kielestä ja suunniteltu erityisesti Windows-sovellusten kehittämiseen. VB julkaistiin vuonna 1991 ja siitä tuli nopeasti varsin suosittua. Vuonna 1999 julkaistu VB 6.0 on viimeinen yksittäisenä kehitystuotteena julkaistu VB:n versio. Tämän jälkeen VB:stä tuli osa Visual Studio.NET:iä [21].

VB:n 5.0 –versioon saakka VB kääntäjä ainoastaan konvertoi ohjelmoijan kirjoittaman lähdekoodin eräänlaiselle välikielelle (kuten Java), jota kutsutaan tavukoodiksi (bytecode). Alkaen VB:n versiosta 5 pystytään luomaan myös suoritettavia alkuperäisohjelmia. VB-ohjelmien käyttämiseen vaaditaan versiosta riippumatta VB runtime module DLL-tiedosto, joka sisältää tarvittavat ajonaikaiset kirjastot sekä konvertoi tavukoodin konekieliseksi [21]. (Kuva 1)



Kuva 1. Tavukoodin suorittaminen tietokoneissa [21].

Microsoft on kehittänyt useita versioita Visual Basicistä sovelluskäyttöön. Näistä tunnetuin on VBA, joka sisältyy useisiin Microsoftin sovelluksiin, kuten MS Officeen. VBA on suureksi osaksi sama kieli kuin VB 6.0 [21].

Microsoftin alkaessa kehittää uutta ohjelmointityökalua, he päätyivät ratkaisuun, jossa Visual Basic .NET (versio 7.0) liitettiin osaksi Microsoftin .NET –alustaa. VB.NET-kieli on varsin erilainen verrattuna VB 6:een, mutta alustassa on import-ominaisuus VB 6 –tiedostojen sisällyttämistä ja konvertointia varten [21].

VB.NET on taaksepäin yhteensopimaton päivitys Visual Basicistä Microsoftin .NET-alustalle. Pääosin aikaisemmin esitettyyn kritiikkiin (enemmän luvussa 4.1) on nyt vastattu

ja kaivattuja ominaisuuksia on lisätty tähän versioon. Esimerkiksi säikeet, olio-ominaisuudet, tyypillinen virheenkäsittely ja ensimmäisen indeksin kiinnittäminen nollaksi löytyvät VB.NETistä. Myös muita .NET-tyypppisiä ominaisuuksia on tullut mukaan. Lisäksi VB.NET on täysin käännettävä kieli ja näin ollen ohjelmien nopea suorittaminen ja editointi kehitysympäristössä ilman täydellistä käännöstä ei enää ole mahdollista (Tämä ominaisuus on tosin tulossa takaisin VB.NET 2005:een) [21].

Monet Visual Basicin alkuperäisistä kriitikoista ovat nyt kiitelleet VB.NETiä kokonaisen ohjelmointikielen luomisesta, vaikkakin osa VB:n kannattajista arvostelee VB.NETiä siitä, että se on tehnyt kielen liian monimutkaiseksi. Lisäksi syntaksin ja joidenkin ominaisuuksien muuttuminen (esim. Variant –muuttujatyypin poistuminen) aiheuttavat suuritöisiä päivitysoperaatioita, mikäli sovellukset halutaan siirtää uudelle alustalle. VB.NET-ympäristössä kirjoitettu koodi ei ole helposti siirrettävissä taaksepäin esimerkiksi VB 6:een. VB.NETin tuen uskotaan vähenevän sillä C#-kieli tuntuu olevan vahvin .NET-alustaisista uusista kielistä, vaikka molemmat käännetäänkin samalle välikielelle ja näin ollen kysymys on oikeastaan vain ohjelmoijien syntaksimieltymyksistä [21].

VB:n kehitys [21]:

- Visual Basic 1.0, toukokuussa 1991, Windowsille.
- Visual Basic 1.0 DOS:lle syyskuussa 1992.
- Visual Basic 2.0 marraskuussa 1992.
- Visual Basic 3.0 kesäkuussa 1993.
- Visual Basic 4.0, elokuussa 1995, oli ensimmäinen versio, joka pystyi luomaan sekä 32- että 16-bittisiä Windows-ohjelmia.
- VB 5.0, helmikuussa 1997, oli ensimmäinen täysin 32-bittinen versio.
- Visual Basic 6.0 ilmestyi kesäkuussa 1998. VB6:n tuki loppuu maaliskuussa 2008.
- Visual Basic .NET ilmestyi 2001 yhdessä .NET Frameworkin kanssa.

- Visual Basic .NET 2003 ilmestyi 2003 yhdessä .NET Frameworkin version 1.1 kanssa.
- 2004 Microsoft julkaisi beta-version Visual Studio.NET 2005:stä ja beta 2.0 –version VB.NETistä. Samana vuonna Microsoft ilmoitti julkaisevansa myös Visual Studion riisutun version, joka keskittyy Visual Basiciin.
- Huhtikuussa 2005 Microsoft ilmoitti, että ei-.NET –pohjaisten VB-versioiden tuki päättyy muutaman vuoden kuluessa.

Mainittakoon Visual Basicin kehityksen yhteydessä myös HyperCard (1987), joka oli Applen Visual Basicia vastaavaa raahaa- ja pudota –sovelluskehitysympäristö. HyperCardin suurena miinuksena oli se, siinä missä VB tuotti aitoja Windows sovelluksen näköisiä sovelluksia, HyperCard tuotti HyperCard-pinoja, jotka eivät olleet oikeita sovelluksia. Apple ei panostanut HC:n kehitystyöhön ja virallisesti HC haudattiin 2004 [21].

3.3 Visual Basic for Applications

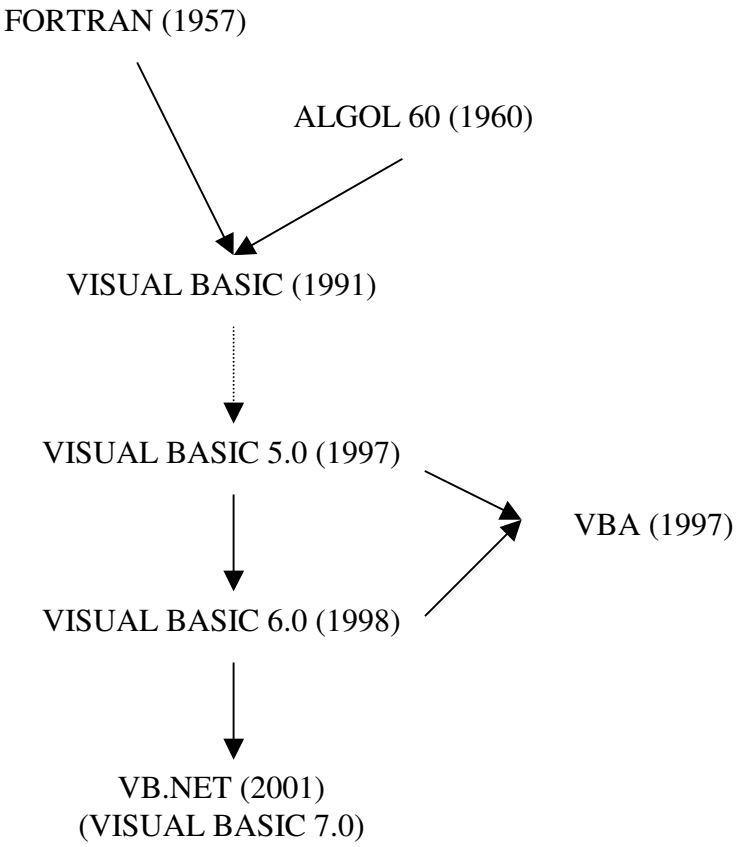
Visual Basic for Applications (VBA) on Microsoftin kehittämä implementaatio Visual Basic –ohjelmointikielestä, joka on sisäänrakennettuna kaikkiin Microsoft Office sovelluksiin. Lisäksi VBA on implementoitu mm. MS Visioon ja osittain implementoitu esimerkiksi AutoCAD ja WordPerfect –ohjelmistoihin. VBA laajentaa aikaisempien makro-ohjelmointikielien kuten WordBasicin ominaisuuksia ja sen avulla on mahdollista kontrolloida lähes kaikkia isäntäsovelluksen ominaisuuksia valikoista ja menuista aina räätälöityihin lomakkeisiin ja dialogeihin [12].

Nimensä mukaisesti VBA on läheistä sukua Visual Basicille, mutta voi normaalisti toimia ainoastaan isäntäsovelluksen yhteydessä, eikä itsenäisenä ohjelmana. VBA:lla on kuitenkin mahdollista ohjata toista sovellusta toisen avulla (Esimerkiksi automaattisten raporttien luominen Wordiin Excel-datasta tai päinvastoin.) [12].

VBA on funktionaalisesti erittäin rikas ja joustava, mutta sillä on myös eräitä merkittäviä rajoitteita, joista enemmän luvussa 4.1. [12].

3.4 Kokonaiskuva

Edellä olen kuvannut Basicin kehityskulkua Visual Basic –kieleksi ja sen erään version muuntumista VBA:ksi. Alla esitän yksinkertaistetun kuvan (Kuva 2) asiasta vuosilukuineen, jotka vähän lähteestä riippuen saattavat olla vuoden tai kaksi pielessä. Erikoisinta oli huomata erilaiset käsitykset siitä, miten joku kieli – tässä tapauksessa BASIC – on syntynyt. Esimerkiksi tietotekniikan tutkija Eric Levenez pitää Basic-kieltä FORTRANin ja Algolin jälkeläisenä [17]. Koikkalainen ja Orponen puolestaan katsovat BASICin juontavan suoraan FORTRANista [19, s. 83]. Edelleen Sethi [25, s. 12-16] ei mainitse BASICiä ollenkaan, kun taas Louden [28, s. 42], Tucker [23, s. 6] ja Mayer [24, s. 5-6] sisältävät ainakin maininnan BASICistä, vaikkakin sen suhde esimerkiksi FORTRANiin jää hämärän peittoon.



Kuva 2. Näkemys BASIC-sukuisten kielten kehityksestä.

4 VBA-ohjelmointi

Luvussa 3 on kerrottu BASIC ohjelmointikielen kehityksestä aina VB.NET (VB 7.0) versioon saakka. VBA on lähes sama kieli kuin VB 6.0. Näin ollen myös kielen ominaisuudet ovat samankaltaisia sen kanssa. Esittelen seuraavassa VBA-ohjelmointikielelle (ja VB:lle) ominaisia piirteitä.

Visual Basic suunniteltiin käytettäväksi riippumatta ohjelmoijan tasosta. Lomakkeiden luominen, kontrollien lisääminen, ominaisuuksien asettaminen ja muuttaminen sekä tapahtumankäsittelijöiden käyttöönotto onnistuu helposti [21].

Sovellus voi koostua yhdestä tai useammasta ikkunasta – käyttöliittymä voi myös puuttua kokonaan, mikäli komponentti ainostastaan lisää ohjelman toiminnallisuutta. Kielessä on automaattinen roskien keruu, kuten BASICissäkin. Mikä erottaa VB:n useista kielistä on se, että VB ei ole yleisesti herkkä isojen ja pienien kirjaimien erolle (case sensitivity). Nykyään saatavilla on paljon uudelleenkäytettäviä komponentteja ja yleensä niiden käyttö on varsin helppoa, mutta kuitenkin lomakkeiden käyttö oman sovelluksensa ulkopuolella on ongelmallista kielen globaalin luonteen takia [21].

4.1 Haitat – mitä rajoituksia VBA:n käytöllä on?

Visual Basic herättää ristiriitaisia tunteita ohjelmoijien keskuudessa. Se suunniteltiin alunperin yksinkertaiseksi kieleksi ja monia esimerkiksi Javalle ja C++:lle tyypillisiä ominaisuuksia ei ole saatavilla VB:ssä. Monet kriitikot sanovatkin, että VB:n haitallisuus piilee nimenomaan sen yksinkertaisuudessa – monia tärkeitä ohjelmoinnin peruasioita jää huomioimatta (muuttujien esittely, tyyppitarkistukset), mikäli VB:tä käytetään opetuskielenä. Tämä johtaa jatkossa usein epä-älykkääseen koodiin ja huonoihin ratkaisuihin. Lisäksi bugien löytäminen saattaa, varsinkin kokemattomilla käyttäjillä, hankaloitua sen seurauksena, että varoitukset ja tarkistukset on mahdollista asettaa pois päältä. Suuri osa VB:n kohtaamasta kritiikistä on peräisin jo BASICin ja Dijkstran ajoilta [21].

Jos sitten mietitään ohjelmointirakenteita, joita Visual Basicissä (versiot 1-6) ja VBA:ssa ei ole, niin ensimmäisenä huomataan, että olio-ohjelmoinnin tuki puuttuu lähes kokonaan (esim. perintä) VB:n .NET-versiota aikaisemmista versioista. Visual Basicin terminologia ei vastaa esimerkiksi Javan ja C++:n vastaavaa (vrt. property - attribute). VB ei myöskään tue säikeitä ja siitä puuttuu niin virheenkäsittelymekanismi (Try-Catch-Finally) kuin myös sisäänrakennettu bittiaritmetiikka. VB:n puutteisiin lukeutuu myös se, että siinä ei ole osoittimia. Lisäksi VB:n kokonaislukutyyppi (signed 8-32 bit) on varsin rajoittunut. Sama on todettava myös VB:n (32 ja 64 bittinen) sisäisestä rajoittuneisuudesta UTF-16 merkkijonoihin [21].

Lisäkritiikkinä VB:lle on esitetty, että VB ei ole kovin siirrettävä – se on saatavilla oikeastaan vain Windows-ympäristössä. Tosin koodia pystytään ajamaan myös VBA-ympäristöissä (myös Mac OS). On myös arvosteltu sitä, että Visual Studion koko ja resurssitarve on liian suuri. Sen lisäksi kehitysympäristössä on todettu paljon ohjelmointivirheitä, joista osa on tosin korjattu service packeilla. Hyvä asia ei ole sekään, että VB:n ja Windows API:n integraation heikkous vaati ohjelmoijilta manuaalista käyttöjärjestelmäfunktioiden määrittelyä sekä matalan tason muistikikkailua, jotka ovat monimutkaisempia kuin C:ssä. Erityisesti VBA:han liittyvä haittapuoli on sen tulkattavuus, jonka takia IP:n piilottaminen ei onnistu MS Office –sovellusten yhteydessä [21].

4.2 Edut – miksi käyttää VBA:ta?

VB:n puolestapuhujat perustelevat VB:n hyvyyttä samalla argumentilla. VB:n yksinkertaisuus sallii erittäin nopean sovelluskehityksen ja siirtyminen toisen kielen parista VB:hen sujuu helposti. Lisäksi VB-sovellusten integrointi tietokantojen kanssa onnistuu helposti [21].

Ominaisuuksia, jotka VB:stä löytyvät [21]:

- Option Base –lauseen käyttö mahdollistaa ensimmäisen indeksin määrittelyn ykköseksi tai nollaksi.

- Sovelluksen ajaminen ja korjaaminen ajon aikana on mahdollista. Täyttä käännöstä ei vaadita.
- Vahva integraatio Windows-käyttöjärjestelmän ja MS Officen kanssa.
- VBA:ssa on lisäksi ominaisuutena nk. makron äänitys, josta voi olla apua ainakin kokemattomille ohjelmoijille (luku 4.3.6).

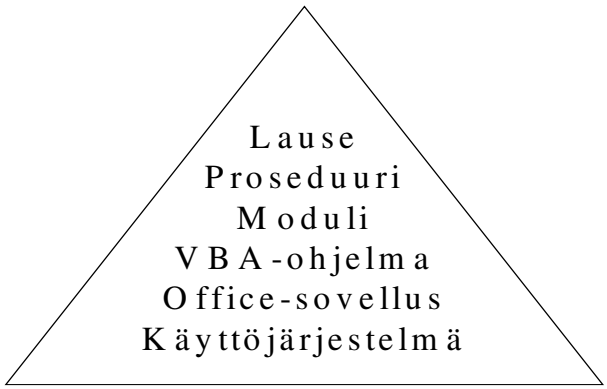
VB on kielenä vähemmän johdonmukainen, mutta sallivampi, kuin useat nykyiset ohjelmointikielet. Kädenvääntö sen hyvyydestä tai huonoudesta ei lopu koskaan [21].

4.3 Yleiskatsaus itse kieleen

Visual Basic for Applications on ohjelmointikieli, jolla käyttäjät voivat mukauttaa Officen toimintaa. VBA mahdollistaa kaikentyyppisen asiakirjaan, laskenta-arkkiin tai tietokantaan rakennetun loogisen toiminnallisuuden ja mitä tahansa Office-sovelluksissa tehtyjä toiminnallisia muutoksia [1, s. xxix].

Nykyisin yrityksissä käytetään paljon mukautettuja ja itse valmistettuja ohjelmistoja. Microsoftin tuotteet ovat käytössä valtaosassa toimistoja ja niiden integraatio omien järjestelmien kanssa on näin väistämätöntä [1, s. xxxi].

VBA on ohjelmointikieli – se ei ole pelkästään makrokieli. VBA-ohjelma on sarja ohjeita, jotka ottavat Office-sovellukselta ohjat käsiinsä ja lisäävät sen toiminnallisuutta tavalla, jota siinä ei aikaisemmin ollut. VBA-ohjelma koostuu moduleista ja modulit puolestaan proseduureista [1, s. xxxii]. Seuraavassa kuvassa (Kuva 3) on esitetty VBA-ohjelman sijoittuminen suhteessa käyttöjärjestelmään.



Kuva 3. VBA-ohjelman sijoittuminen suhteessa käyttöjärjestelmään.

VBA:lla voidaan haluttaessa luoda hyvin tehokkaita ratkaisuja, muta sitä ennen on omaksuttava itse ohjelmointikieli ja toisaalta eri objektimallit, joita eri sovellusohjelmiin liittyy [2, s. 5].

VBA-ohjelmien tyypit:

- **Laajentava komento** nimensä mukaisesti antaa mahdollisuuden suorittaa tehtävä, johon sovellus ei sellaisenaan kykene. Pienimmillään tämä tarkoittaa yhtä VBA-proseduuria.

- **Upotettu toiminto**, jota voi käyttää suoraan vaikkapa työkirjasta. Esimerkiksi painikeryhmä.

- **Asiakirjapohjat** – toiminnallisuus liitetään osaksi kaikkia kyseisen luokan asiakirjoja tai laskentalomakkeita sijoittamalla toiminnallisuus asiakirjapohjaan.

- **Integroidut sovellusapuvälineet**. Esim. wordin ja excelin yhdistäminen VBA:n avulla.

Office-paketti on maailman käytetyin toimisto-ohjelmisto PC-tietokoneissa, ja niinpä jokaisen yrityksen pitää pystyä mukauttamaan Office juuri omiin tarpeisiinsa sopivaksi. VBA on työkalu, jolla tällainen mukauttaminen on mahdollista [1, s. 5].

VBA on tulkittu, proseduraalinen ohjelmointikieli. Office-ympäristössä tapahtuva VBA-ohjelmointi perustuu Officen oliomalleihin, joihin sisältyvät kaikki Officen oliokirjastot [1, s. 6-7].

Muutamia VBA:n piirteitä [1, s. 9-10]:

- Muutujat ovat muistiyksiköitä esittäviä symboleja.
- Oliokirjasto sisältää sovelluksen sanaston (vrt. include C++:ssa).
- Lausekkeet rajaavat ehdollisia ja toistettavia ohjeita.

4.3.1 VBA:n toimintaperiaate ja syntaksi

Esittelen seuraavassa VBA:n toimintaperiaatteeseen ja syntaksiin liittyviä asioita, joista jokaisella aikaisempaa ohjelmointikokemusta omaavalla henkilöllä on jo jonkinlainen kuva, sillä VBA muistuttaa hyvin pitkälle muita tuntemiamme kieliä.

VBA-ohjelmointi, kuten Windows ohjelmointi yleensä muutenkin, on tapahtumapohjaista ja olioperusteista. Tapahtumilla tarkoitetaan käyttäjän (näppäimistö, hiiri), ohjelmaprosessin (Windows) tai laitteiston (esimerkiksi tietoliikenneportit) aiheuttamia keskeytyksiä. Ohjelmoijan tehtävä on ratkaista, mihin keskeytyksiin reagoidaan ja miten [4, s. 19-20].

VBA sisältää ohjelmointikielille tyypilliset operaattorit (loogiset, vertailu, aritmetiikka) ja muuttujatyypit, joista osaa on esitelty tarkemmin tässä tutkielmassa (Taulukko 1, Taulukko 2). VBA:n kommenttimerkkinä toimii heittomerkki (’), jonka jälkeen tulevan tekstin VBA-tulkki jättää huomiotta [6, s. 123-124].

VBA käsittelee objekteja, joita Office-ohjelmissa on yhteensä useita satoja. Objektit järjestyvät hierarkkisesti ja ne voivat toimia varastopaikkana muille objekteille. Esimerkiksi Excel voi olla objekti, jonka nimi on `Application` ja se sisältää `Workbook`-objekteja, jotka puolestaan sisältävät `Worksheet`-objekteja, jotka sisältävät

`Range`-objekteja [5, s. 765-768]. Itse koodissa tällainen voisi esiintyä vaikkapa seuraavalla tavalla:

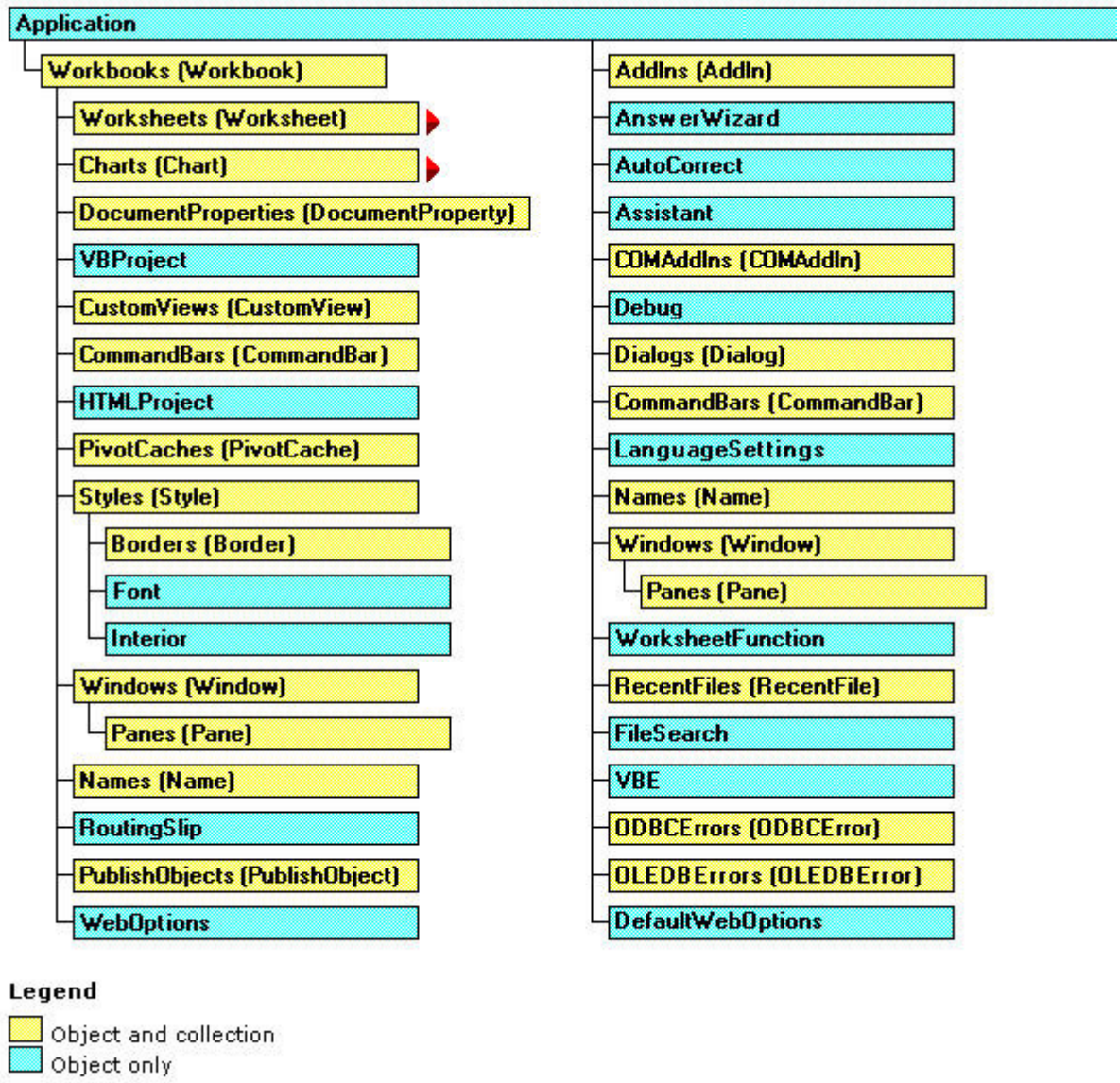
```
Application.Workbooks(FileName).Sheets("Asetukset").Range("A1").Select
```

Tässä esimerkissä valitaan sovelluksen tietynnimisen työkirjan Asetukset-lehden solu A1. Objektiin siis viitataan VBA-koodissa määrittämällä sen sijainti hierarkiassa käyttämällä erottelijana pistettä. Mikäli määrättyjä viittauksia jätetään pois, käytetään aktiivisia objekteja. Esimerkiksi lause:

```
Sheets("Sheet1").Range("A1").Select
```

viittaa aktiivisen työkirjan ensimmäisen välilehden soluun A1. Samanlaiset objektit muodostavat kokoelmia [5, s. 765-768]. Edellä olevassa koodissa esimerkiksi viitataan `Sheets`-kokoelmaan, josta sitten valitaan haluttu olio (`Sheet1`).

Esimerkiksi MS Excel 2000 sisältää seuraavan kuvan (Kuva 4) kaltaisen objektirakenteen.



Kuva 4. Microsoft Excelin objektikartta (Lähde: MS Excel Help).

Worksheets- ja Charts-objektikokoelmien perässä oleva nuoli on merkkinä siitä, että nämä itsessään sisältävät vielä varsin laajan objektirakenteen, joka on kuvattu tarkemmin liitteessä 3.

VBA-objektien ominaispiirteitä ovat [1, s. 11-12]:

- **Ominaisuudet (properties)**, jotka ovat muutettavan olion luonteenpiirteitä.
- **Metodit (methods)**, jotka ovat toimenpiteitä, joita olioon voidaan liittää.

- **Tapahtumat (events)**, jotka edustavat olion rekisteröimiä tapahtumia.

VBA:n olioperusteisuus näkyy hyvin sen varsinaisessa syntaksissa, joka tyypillisesti on seuraavankaltaista [4, s. 21-22]:

`olio.ominaisuus = arvo` (ominaisuudet) tai

`olio.metodi argumentit` (metodit) tai

```
Private Sub UserForm_Click()
  Tähän ohjelmoija määrittelee, mitä tehdään tietyn
tapahtuman sattuessa.
End Sub
```
(tapahtumat).

Kaikki VBA:n oliot kuuluvat siis johonkin luokkaan, joka määrittelee ominaisuudet niille olioille, jotka siihen kuuluvat. Luokkakirjastot puolestaan sisältävät olioluokkia. VBA rakentuu joukosta kirjastoja ja ohjelmointikielen sisältämät käskyt ovat suureksi osaksi kielen sisäisissä kirjastoissa [4, s. 22].

4.3.2 Tietotyypit

Seuraavassa taulukossa (Taulukko 2) on esitelty VBA:n perustietotyypit ja niiden ominaisuudet. VBA:ssa ei vaadita muuttujan tyypin eksplisiittistä määrittelyä automaattisesti. Mikäli muuttuja ei esitellä tietyntyyppiseksi, tulkitaan se `Variant`-tyyppiseksi. Muuttujan näkyvyys riippuu siitä, onko se esitelty jonkin aliohjelman sisällä vai. Mikäli näin on tehty, kyseessä oleva muuttuja on käytössä ainoastaan tässä aliohjelmassa. Mikäli esittely on tehty modulitasolla, voidaan ko. muuttujaa käyttää kaikissa modulin aliohjelmissa ja jos muuttuja on esitelty `Public`-tyyppiseksi, voidaan sitä käyttää kaikissa projektin moduleissa.

Tyyppi/avainsana	Muistin kulutus	Arvoalue
Integer	2 tavua	–32,768 - 32,767

Long	4 tavua	–2,147,483,648 - 2,147,486,647
Single	4 tavua	–3402823E38 - –1.401298E–45 tai 1.401298E–45 - 3.402823E38
Double	8 tavua	–1.79769313486232E308 - –4.94065645841247E–324 tai 1.79769313486232E308 - 4.94065645841247E–324
Currency	8 tavua	–922,337,203,477.5808 - 922,337,203,685,477.5807
Date	8 tavua	1.1.100 – 31.12.9999
Object	4 tavua	Mikä tahansa objektiviittaus.
Boolean	2 tavua	-1(true) tai 0(false)
Variant	16 tavua + arvon tallennuskoko	Tilanneriippuva. Sama kuin Double, String tai Date; voi sisältää myös arvon Error tai Null.
Byte	1 byte	0 – 255

Taulukko 2. VBA:n muuttujatyypit [27], [1, s. 74], [3, s. 17].

4.3.3 Aliohjelmien rakenne

VBA:ssa on kahdenlaisia aliohjelmia – lausealiohjelmia (`Sub`) ja funktioaliohjelmia (`Function`). `Sub`-tyyppinen aliohjelma suorittaa jonkin toiminnan objekteille, eikä palauta mitään. `Function`-tyypin aliohjelma puolestaan palauttaa yhden arvon, jota voidaan sen jälkeen käyttää hyväksi muun toiminnan tai virheenkäsittelyn ohjaamisessa [2, s. 61], [3, s. 8].

Aliohjelmien rakenne on tyypillisesti seuraavanlainen [2, s. 61]:

1. Otsikko
2. Käskyosat
3. Lopetus

Esimerkiksi:

```
Sub Esimerkki () 'Otsikko
  MsgBox("Hei maailma!") 'Käskyt
End Sub 'Lopetus
```

Aliohjelmien näkyvyyttä voi säädellä avainsanoilla `Private`, `Public` ja `Static`. `Private` avainsanalla määritellyt aliohjelmat ovat käytettävissä ainoastaan sen modulin sisällä, mihin se on kirjoitettu. `Public` avainsanalla määriteltyjä aliohjelmia voi puolestaan kutsua mistä tahansa kyseisen sovelluksen modulista. `Static` avainsanalla määriteltyjen aliohjelmien kaikista paikallisista muuttujista tulee staattisia, mikä tarkoittaa sitä, että niiden arvo säilyy koko ohjelman suorituksen ajan [2, s. 62-67], [3, s. 8].

4.3.4 Ohjausrakenteet

VBA:n ohjausrakenteet ovat nekin varsin tyypillisiä ja esitelen ne siksi vain suppeasti.

Ehdolliset rakenteet [3, s. 21-28], [6, s. 141-145]:

- `If...Then` ehtorakenne soveltuu yhden ehdon ja lauseen tai useamman lauseen suorittamiseen.
- `If...Then...Else` ehtorakenne sopii yhden ehdon ja kahden vaihtoehtoisen koodiosan suorittamisen valintaan.
- `If...Then...ElseIf` mahdollistaa useammat ehdot ja useat vaihtoehtoiset koodiosat.

- `Select Case` käy yhden ehdon ja usean vaihtoehtoisen koodiosan suorittamiseen.

Silmukkarakenteet [3, s. 21-28], [6, s. 141-145]:

- `Do...Loop` soveltuu, kun käskyjoukko on suoritettava ennalta tuntematon määrä kertoja.

- `Do While...Loop` –rakenteessa ehto testataan silmukan alussa. Silmukka suoritetaan ehdon ollessa tosi.

- `Do...Loop While` –silmukka suoritetaan vähintään kerran, ehdon testaus silmukan lopussa. Silmukka jatkuu ehdon ollessa tosi.

- `Do...Loop Until` - silmukka suoritetaan vähintään kerran, ehdon testaus silmukan lopussa. Silmukka jatkuu ehdon ollessa epätosi.

- `While...Wend` on sama kuin `Do...While Loop`

- `For...Next` –silmukka suoritetaan ennalta tietty määrä kertoja laskurin avulla.

- `For Each...Next` –silmukkaa suoritetaan ryhmän alkioiden lukumäärän osoittama määrä kertoja.

4.3.5 Virheenkäsittelymekanismit

Ideaalisessa maailmassa virheenkäsittelykoodia ei tarvittaisi ollenkaan, mutta sehän ei aivan ole todellisuutta. Laiteongelmat, käyttäjän kannalta arvaamattomat toimenpiteet ja tietysti varsinaiset virheet koodissa saattavat aiheuttaa ajonaikaisia virhetilanteita, jotka pysäyttävät ohjelman suorituksen tai saavat ohjelman toimimaan ei-toivotulla tavalla [3, s. 344].

Virheenkäsittelykoodi on koodiosa, joka reagoi sovelluksessa tapahtuneeseen virhetilanteeseen. Ohjelmoijan tulee ennakoida virhetilanteiden syntyminen ja lisätä koodiin käsittelymekanismit potentiaalisille virheille. VBA:ssa virheenkäsittely sisältää

kolme vaihetta: virheenkäsittelyn käyttöönoton, varsinaisen virheenkäsittelijän ja virheenkäsittelystä poistumisen [3, s. 345].

Virheenkäsittelyn käyttöönotto tarkoittaa käytännössä VBA-tulkin suorittamaa `On Error` –lausetta, joka määrittelee virheenkäsittelykoodin. Virheenkäsittely on aktiivisena niin kauan kuin aliohjelma, jossa koodi sijaitsee, on aktiivisena. Virheenkäsittelyn käyttöönotto ilmoitetaan seuraavalla tavalla [3, s. 347].

```
On Error GoTo ErrorHandler
'Tässä määritellään virheenkäsittelijän alkukohdaksi rivi,
'joka on nimetty ErrorHandler-nimellä.
'Virheenkäsittelijä voidaan myös poistaa käytöstä
'kirjoittamalla:
On Error GoTo 0
```

Virheenkäsittelijä alkaa nimetyltä riviltä, joka on tavallisesti sijoitettu aliohjelman loppuun, jolloin sitä edeltää Exit-lause, joka estää virheenkäsittelykoodin suorittamisen, mikäli virhettä ei synny [3, s. 347-348].

```
  Exit Sub 'to avoid the handler
ErrorHandler: 'virheenkäsittelijän nimetty aloitusrivi
  MsgBox ("Virhe: " & Err) 'informoidaan käyttäjää virheestä
  Exit Sub 'poistutaan aliohjelmasta
End Sub 'aliohjelman varsinainen loppu
```

Hyvässä virheenkäsittelyssä on kuitenkin yleensä mietittävä vielä tarkemmin mahdollisia virhetilanteiden aiheuttajia ja niiden käsittelyä hallitusti esimerkiksi `Select...Case`-rakenteen avulla [3, s. 348].

```
ErrorHandler:
   Select Case Err.Number 'otetaan virheen numero
      Case AnticipatedError#1
         ' Hoidetaan virhe #1.
      Case AnticipatedError#2
         ' Hoidetaan virhe #2.
      Case UnAnticipatedErrors
         ' Hoidetaan odottamaton virhe.
      Case Else
         ' Varaudutaan kaikkeen.
   End Select
```

Virheenkäsittelystä poistumiseen on useita vaihtoehtoja [5, s. 348]:

- `Resume(0)` palauttaa ohjelman suorituksen lauseeseen, jossa virhe tapahtui. Tätä kannattaa siis käyttää, mikäli toiminto halutaan korjauksen jälkeen toistaa.

- `Resume Next` jatkaa suoritusta lauseesta, joka seuraa virheen aiheuttanutta lausetta.

- `Resume` *`rivi`* –komennolla ohjelman suoritus jatkuu ennalta määrätystä *`rivi`*-kohdasta, joka sijaitsee samassa aliohjelmassa virheenkäsittelykoodin kanssa.

- `Err.Raise Number:=`*`luku`* synnyttää suorituksenaikaisen virheen ja aiheuttaa toiseen virheenkäsittelijään siirtymisen VBA:n kutsupinon kautta (ks. 5, s.351-352).

Virheenkäsittelyn suunnittelu ja toteutus, mikäli sellainen yleensä on, on vaativa tehtävä varsinkin isommissa sovelluksissa ja niinpä on syytä muistaa kirjoittaa varmistava virheenkäsittelijä, jota muut virheenkäsittelijät voivat kutsua, kun ne eivät itse pysty hallitsemaan virhetilannetta.. Tämän ”varmistajan” tehtävänä on keskeyttää suoritus ja mahdollistaa tietojen tallentaminen ja ohjelman hallittu alasajo [3, s. 353].

4.3.6 VBA-koodin muodostaminen

Edellä on esitelty VBA:n yleisominaisuudet ja niinpä seuraavaksi on syytä perehtyä VBA-koodin luomiseen. Siihen on kaksi tapaa:

- Käynnistää makronauhoittaja ja nauhoittaa toiminnot tai

- kirjoittaa koodit suoraan VBA-moduliin.

Makron nauhoittaminen onnistuu valitsemalla Tools/Macro/Record New Macro. Tällöin avautuvaan ikkunaan voit asettaa makron nimen, mahdollisen pikanäppäimen, makron tallennuspaikan ja kuvauksen makrosta [2, s. 8-10], [7, s. 46-47] ja [5, s. 753-763]. Tässä tutkielmassa en aio kertoa tästä tavasta sen enempää – jokainen pystyy itse kokeilemalla tutustumaan makronauhoittimen hyviin ja huonoihin puoliin.

Yksinkertaisten tehtävien automatisointi on helppoa nauhoituksen avulla. On kuitenkin niin, että tällöin syntyvä koodi ei ole kovinkaan tehokasta tai nopeaa. Niinpä monimutkaisempien makrojen luomiseksi on VBA-koodit kirjoitettava konkreettisesti käsin. Tämä tapahtuu lisäämällä koodimoduuli, VBA-ohjelman varastopaikka, työkirjaan VBA-editorissa, jossa varsinainen koodin kirjoitus tapahtuu. Ohjelmointiprosessi itsessään ei eroa normaaleista käytännöistä (määrittely, suunnittelu, implementointi, testaus, korjaus, ylläpito).

4.4 Tyypillinen käyttö

Luvussa 5 tulen esittämään muutamia esimerkkejä siitä, miten VBA:ta hyödynnetään työssä. Tyypillisimmin VBA:ta käytetään yhdessä Excelin kanssa seuraaviin tarkoituksiin [5, s. 748]:

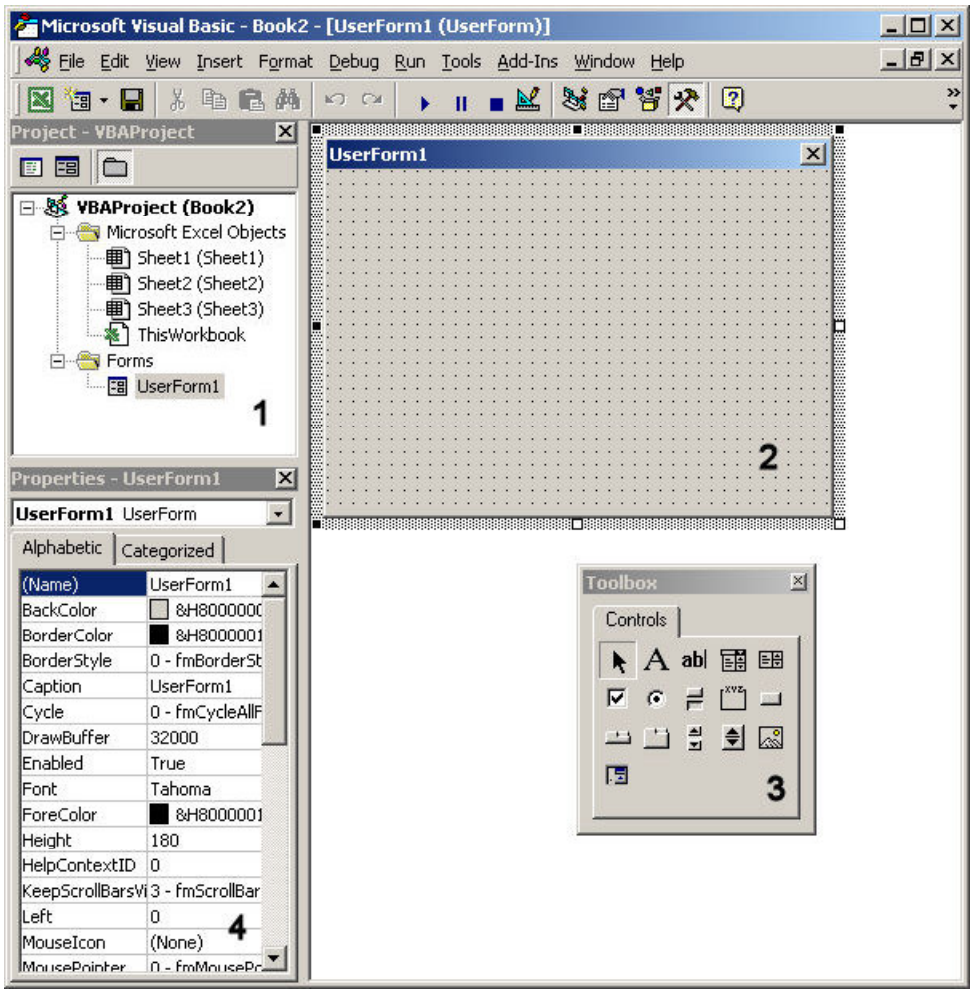
- Merkkijonon tai kaavan lisäämiseen,
- jatkuvasti tai toistuvasti suoritettavan toimenpiteen automatisointiin,
- oman komennon luomiseen,
- oman työkalurivipainikkeen luomiseen,
- tiedonsyöttöpohjien luomiseen (virheiden eliminointi),
- uuden taulukkolaskentafunktion kehittämiseen,
- täydellisen makrokäyttöisen sovelluksen tai Excelin apuohjelman luomiseen.

4.5 Ohjelmointiympäristö

Visual Basic –editori käynnistetään mistä tahansa Office-ohjelmasta Komennolla Tools-Macro-Visual Basic tai näppäinyhdistelmällä Alt+F11.

VBA:n ohjelmointiympäristö on varsin tyypillinen ominaisuuksiltaan. Siitä löytyy project explorer –ikkuna (Kuva 5, kohta 1), joka esittää ohjelmointiprojekin rakenteen

visuaalisella tavalla. Lomakkeiden muokkausikkuna (Kuva 5, kohta 2) työkaluvalikkoineen (Kuva 5, kohta 4) ja properties-ominaisuusikkuna (Kuva 5, kohta 3) ovat varsinaisten lomakkeiden muokkausta ja niiden ominaisuuksien asettamista varten. Lisäksi löytyvät myös varsinainen koodieditori ja ohjelman debuggaukseen tarvittavat toiminnot sekä help-toiminnot.



Kuva 5. Microsoft Visual Basic for Applications ohjelmointiympäristö.

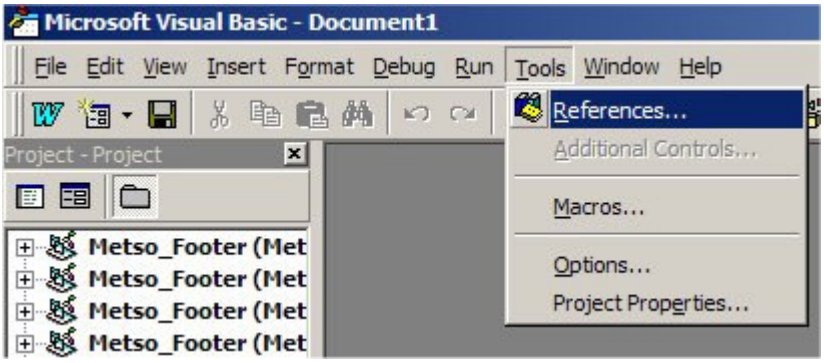
Koodieditori avautuu samaan ikkunaan lomakkeiden muokkausikkunan kanssa ja sen voi valita painamalla F7-näppäintä tai valikosta View/Code. Mikäli projektissa ei ole visuaalisia objekteja, näkyy koodieditori oletuksena, kun koodimoduli on lisätty (Insert/Module).

Ohjelmoitaessa monimutkaisempia sovelluksia tarvitaan usein lisätietoja objektien ominaisuuksista. Tätä varten VBA-editorissa on varsin käyttökelpoinen help-järjestelmä, jossa on yksityiskohtaisia tietoja niin objekteista, menetelmistä kuin aliohjelmistakin. Help on perusteellinen ja sitä pääsee käyttämään helposti VBA-editorissa siirtämällä hiiren kohdistin ominaisuuteen tai menetelmään ja painamalla F1. Toinen vaihtoehto on etsiä neuvoja indeksoidun haun avulla.

VBA:n ohjelmointiympäristö muistuttaa todella suurelta osin Microsoftin Visual Basic ohjelmointiympäristöä, jota on esitelty esimerkiksi Leppämaan kirjassa [4]. Valikot, näppäinkomennot, työkalurivit, pikavalikot ja ikkunat ovat lähes identtisiä VBA:n ohjelmointiympäristössä verrattuna erikseen hankittavaan VB ohjelmointiympäristöön. Koodieditorissa ja ohjelmointiympäristön tiloissa (design, run, break) puolestaan ei ole lainkaan eroa näiden kahden ympäristön välillä. Oikeastaan ainoat merkittävät erot näiden kahden ympäristön välillä ovat ne, että VB:n ympäristö antaa mahdollisuuden koodin kääntämiseen ja näin ollen ajokelpoisen EXE-tiedoston tekemiseen, lisäksi VB tarjoaa vahvemmat työkalut tietokantojen käsitelyyn [4, s. 130-144]. VB-ympäristön projektin tallentamiseen liittyvät toiminnot ovat myös huomattavasti monipuolisemmat [4, s. 24-29] kuin Officen mukana automaattisesti tulevan VBA-ympäristön vastaavat.

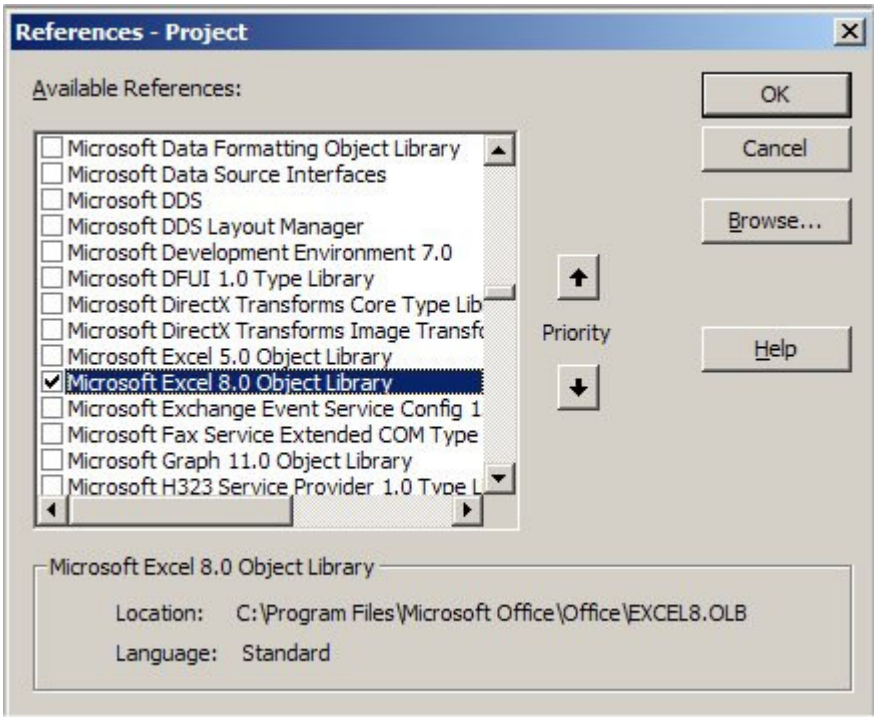
4.5.1 VBA:n objektikirjastot

Oliotermit, jotka kuuluvat muuhun sovellukseen kuin mihin VBA-editori sillä hetkellä kuuluu (esimerkiksi haluat saada Excelin objektit käyttöön Word-asiakirjapohjasta), eivät ole oletuksena ko. editorin käytössä [1, s. 41]. Nämä oliot saa helposti käyttöön VBA-editorissa lisäämällä haluttu kirjasto valitsemalla valikosta Tools-References (Kuva 6)



Kuva 6. Objektikirjaston lisääminen VBA:n käyttöön 1/2.

ja sen jälkeen asettamalla rastit haluttuihin kohtiin, minkä jälkeen painetaan OK (vrt. include C++:ssa) (Kuva 7).



Kuva 7. Objektikirjaston lisääminen VBA:n käyttöön 2/2.

4.6 Koodin optimointi

VBA-koodia voi optimoida nopeampaan suoritukseen yksinkertaisten periaatteiden avulla. Pääperiaatteena voitaneen pitää menetelmää, jossa lasketaan koodissa olevien pisteiden lukumäärä. Kuulostaa erikoiselta, mutta jokainen piste on siirtyminen eteenpäin

oliohierarkiassa ja näin ollen jokainen piste vie aikaa. Kutsuja voi minimoida monellakin eri tavalla. Esimerkiksi objektimuuttujan käyttäminen objektinimen toistuvan käyttämisen sijaan vähentää OLE-kutsuja. Esimerkiksi edellä käyttämäni esimerkki

```
Application.Workbooks(FileName).Sheets("Asetukset").Range("A1
").Select
```

on sellaisenaan aivan käypä, mutta jos se toistuu useasti on syytä miettiä sen toteuttamista objektimuuttujan avulla:

```
Dim SelectedCell as Object
Set SelectedCell =
Application.Workbooks(FileName).Sheets("Asetukset").Range("A1
")
```

Nyt aikaisemman esimerkin lause voidaan kirjoittaa muodossa:

```
SelectedCell.Select
```

Samoin `With`-lausetta voidaan käyttää samassa tarkoituksessa:

```
With Application.Workbooks(Filename).Sheets(“Asetukset”)
  .Range(“A1”).Value = 1
  .Range(“A2”).Value = 2
End With
```

Nyt vältetään toistuvat `Application`-, `Workbooks`-, ja `Sheets`-objektien kutsut.

Silmukoissa on tärkeää pitää ominaisuudet ja menetelmät silmukan ulkopuolella, sillä ne pystytään lukemaan muuttujista nopeammin kuin ominaisuuksista, sekä mahdollisuuksien mukaan välttää indeksoitujen silmukkamuuttujien käyttöä.

```
For i = 2 to 200
  Cells(i, 1).Value = Cells(1,1).Value
Next
```

Seuraavassa on siirretty sijoitettava arvo muuttujaan ennen varsinaista silmukkaa:

```
cv = Cells(1,1).Value
For i = 2 to 200
  Cells(i, 1).Value = cv
```

```
Next
```

Indeksistä ei tässä tapauksessa kannata kuitenkaan luopua, sillä se toisi yhden objektikutsun lisää silmukan sisälle, sillä For...Each iteraattoria voi käyttää ainoastaan kokoelmien ja taulukoiden läpikäymiseen.

```
cv = Cells(1, 1).Value
For Each i In Range("A2:A200").Rows
  Cells(i.Row, 1).Value = cv
Next
```

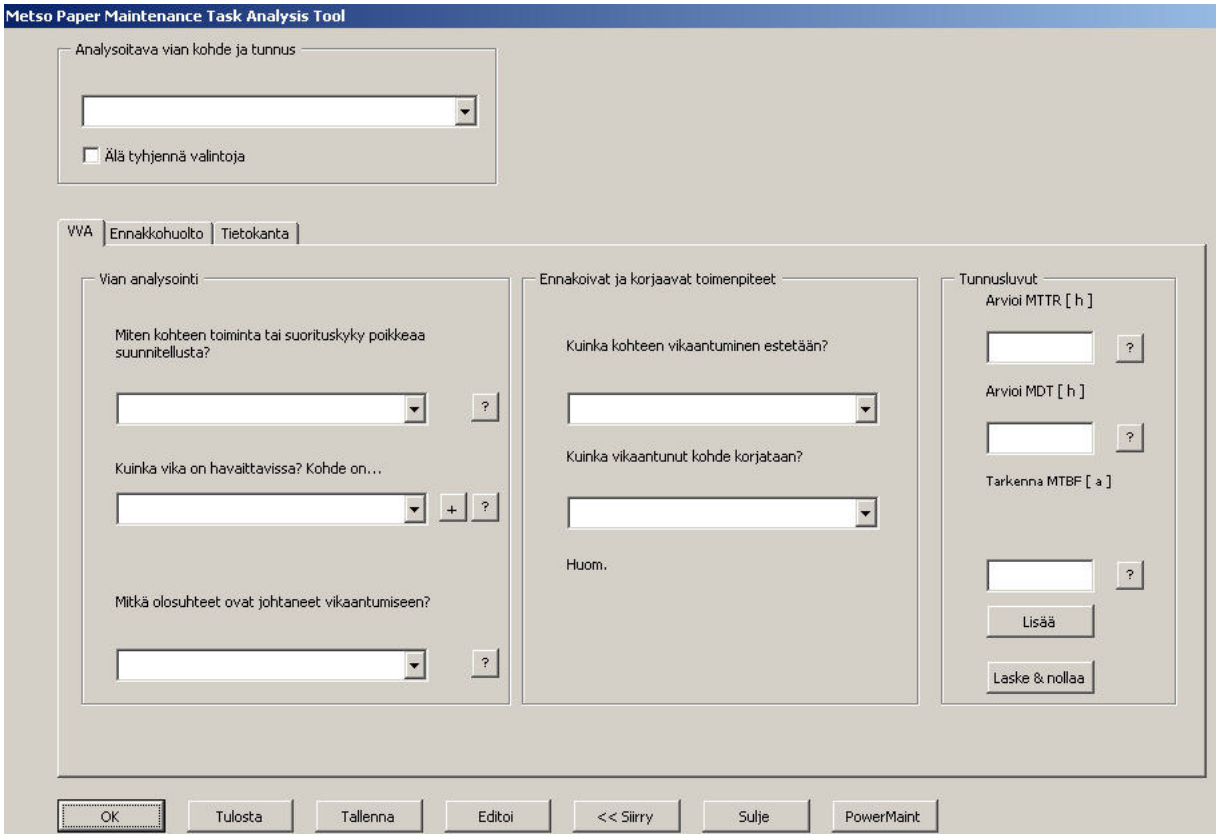
Muita keinoja koodin nopeuttamiseksi ovat `Variant`-muuttujien käytön minimoiminen, määritettyjen objektityyppien ja vakioiden käyttäminen sekä näytön päivityksen ja varoitusvietien estäminen [3, s. 333-342], [5, s. 849-852].

5 Esimerkkejä VBA:n käytöstä

Seuraavassa esittelen muutamia tapauksia, joissa ratkaisuun on käytetty VBA-ohjelmointikieltä.

5.1 RCM-työkalu

Metso Paper on kehittänyt sisäiseen käyttöönsä työkalun (Kuva 8), jolla voidaan helpottaa luotettavuuskeskeisen kunnossapitoanalyysin toteuttamista asiakkaalle (Reliability Centered Maintenance). Alkuperäisen sovelluksen kehitti Toni Mäcklin, joka on tehnyt aiheesta lopputyön Jyväskylän ammattikorkeakoululle. Itse olen ollut mukana kehittämässä sovellusta niin käyttöliittymän kuin toiminnallisuudenkin osalta. Seuraavassa esimerkki eräästä ongelmasta ja sen ratkaisusta.



Kuva 8. RCM-työkalun käyttöliittymä.

5.1.1 UserFormien koon skaalautuminen käytössä olevan resoluution mukaan

Ongelmana oli se, että haluttiin piilottaa Excelin valikot ja työkalurivit käyttäjän näkymättömiin siksi aikaa, kun hän täyttää tietoja sovelluksessa olevalle lomakkeelle (UserForm). UserFormien koko voitiin suunnitteluvaiheessa kiinnittää, mutta koska sovellusta käytettiin erilaisia näyttöresoluutioita käyttävissä tietokoneissa, seurasi tästä se, että vaadittiin dynaamista lomake- ja komponenttikoon sekä sijainnin hallintaa. Excel-sovelluksiin liittyviä keskustelupalstoja lukiessa huomaa, että tämä ongelma on itse asiassa muodossa tai toisessa varsin yleinen. Ratkaisu ei tällä kertaa löydy varsinaisesti VBA:sta vaan avuksi täytyy ottaa muutamia WIN32 API-funktioita.

Määritellään vakiot ikkunakoon määrittämiseksi:

```
'API constants
Const SW_HIDE = 0
Const SW_SHOWNORMAL = 1
Const SW_SHOWMINIMIZED = 2
Const SW_SHOWMAXIMIZED = 3
```

Esitellään API-funktiot, jotta päästään käsiksi käyttöjärjestelmältä saataviin tietoihin:

```
'API functions
Private Declare Function apiShowWindow Lib "user32" _
    Alias "ShowWindow" (ByVal hwnd As Long, ByVal nCmdShow As
Long) As Long
Private Declare Function FindWindow Lib "user32" _
    Alias "FindWindowA" (ByVal lpClassName As String, ByVal
lpWindowName As String) As Long
```

Haetaan kahva kyseessä olevalle ikkunalle käyttäen hyväksi yllä esiteltyä API-funktioita.

```
'Get the userform's window handle
If Val(Application.Version) < 9 Then
  mhWndForm=FindWindow("ThunderXFrame",oForm.Caption)'XL97
Else
  mhWndForm=FindWindow("ThunderDFrame",oForm.Caption)'XL2000+
End If
```

Sen jälkeen voidaankin kutsua aliohjelmaa, joka yksinkertaisesti muuttaa parametrina annetun ikkunakahvan omistajan halutun kokoiseksi.

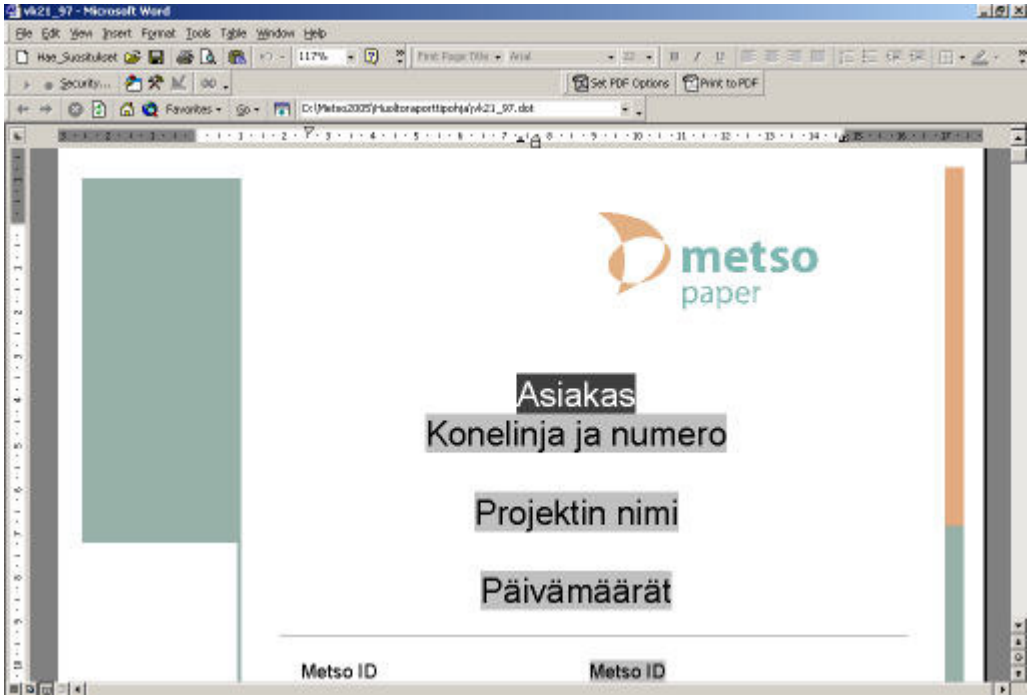
```
Sub MaximizeExcelForm()
  Dim loX  As Long
  loX = apiShowWindow(mhWndForm, 3) 'formhandle and
window_state as arguments
End Sub
```

5.2 Lomaketietojen kerääminen ja lähettäminen tietokantaan

Metson asiantuntija kirjaa asiakkaan kunnossapitoraportteihin suosituksia paperikoneelle tehtävistä toimenpiteistä. Liian usein käy niin, että suosituksia ei saada myytyä asiakkaalle, koska myyntihenkilöillä ei ole tietoa niistä. Niinpä kehitimme Word-lomakepohjan, joka napin painalluksella ottaa talteen asiantuntijan tekemät suositukset, tallentaa ne seurantaa varten Excel-tiedostoon, parsii niistä merkkijonon ja lähettää sen Notes-postiohjelman avulla Notes-tietokantakäsittelijälle, joka puolestaan järjestää ja esittää tiedon niin, että se on myyntimiesten käytössä helposti tietokantareplikaation jälkeen.

5.2.1 Lomakekenttien sijoittaminen pohjaan & tiedon hakeminen

Jotta Word-lomakepohjaa voidaan hyödyntää automaattisesti, tulee siihen olla upotettuna lomake-elementtejä, joihin voidaan sitten viitata VBA-ohjelmassa. Tässä raportissa olemme käyttäneet runsaasti TextFormField-komponentteja (Kuva 9). Lomake-elementit vaativat toimiakseen dokumentin suojauksen, joka on syytä suunnitella etukäteen, sillä yleensä dokumentissa esiintyy tarvetta myös suojaamattomille osille, joiden sisältöä dokumentin käyttäjällä on oikeus muuttaa.



Kuva 9. Raporttipohja, jossa on käytetty TextFormField –lomakekenttiä (näkyvät kuvassa harmaalla).

Tiedon hakeminen tehdään nyt yksinkertaisesti silmukalla:

```
For i = 1 To 10
  Recommendation = ActiveDocument.FormFields("Suositus" & i).Result
  If Len(Recommendation) > 10 Then
    xlapp.Workbooks(fname).Sheets(1).Range("G" & datarows + 6).Value =
Recommendation
    datarows = xlapp.Workbooks(fname).Sheets(1).Range("j1").Value
  End If
Next
```

5.2.2 Merkkijonon tai XML-tiedoston muodostaminen ja tiedon lähettäminen Notes –postikannan avulla

Tieto halutaan usein lähetettää jonkinlaisena merkkijonona tai standardoituna notaationa esimerkiksi tietokantaan automaattista jatkokäsittelyä varten. Word- ja Excel-sovelluksissa on nykyään (versio 2003) valmiina ominaisuus, jolla tiedostot voidaan tallettaa XML-formaattiin, mutta valitettavasti tuo formaatti sisältää varsin paljon turhaa tietoa ja turhia tageja. Sen lisäksi otettiin huomioon se, että Metso Paper käyttää suurimmaksi osaksi Officen versiota 97, jossa XML-tallennusominaisuutta ei ole. Näin ollen kirjoitimme

VBA:lla pienen parserin, joka hoitaa tehtävän varsin tehokkaasti ja ilman turhaa täytettä datassa.

Kuten on aikaisemmin mainittu, on VBA:lla mahdollista ohjata muita sovelluksia isäntäsovelluksesta käsin. Tässä tapauksessa otamme Notes-sähköpostiohjelman haltuun käyttäen Exceliin kirjoitettua VBA-ohjelmaa. Käytännössä tämä vaatii sen, että luodaan VBA:lla uusi Notes-objekti, jolle sitten annetaan kunkin käyttäjän henkilökohtaista postikantaa koskevat tiedot (userID, postikannan nimi). Tämän jälkeen huomioitava on vielä se, että Notes-clientin tulee olla aktiivisessa tilassa, ennen kuin makro voidaan ajaa niin, että posti lähtee haluttuun paikkaan.

Esimerkki, joka ottaa Notesin haltuun ja parsii merkkijonon lähettämistä varten on nähtävissä liitteessä (Liite 1).

5.3 Mittaustietoja sisältävän tiedoston lukeminen ja graafien piirtäminen

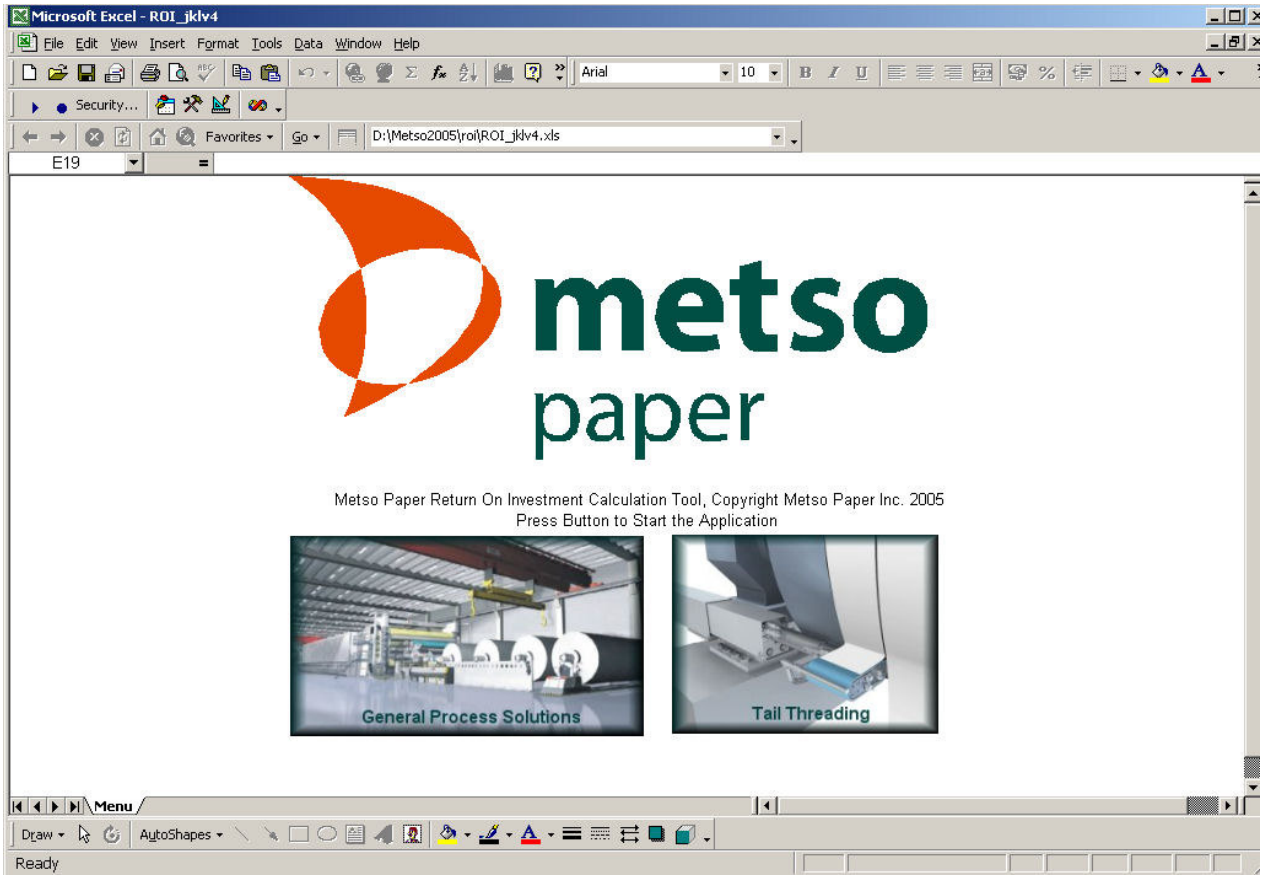
Paperikoneiden ennakkohuollossa käytetään hyväksi käsitietokoneita mittaustietojen tallentamisessa. Mittauskierroksen jälkeen kannettava laite telakoidaan ja tieto siirretään PC:lle MS ActiveSync –ohjelmiston avulla. Käyttöjärjestelmään ajastettu Excel-sovellus käy lukemassa uuden tiedoston säännöllisesti, purkaa tiedoston XML-tagituksen, tallentaa mittaustiedot ja piirtää niistä graafit. Ohjelma luo tarvittaessa uuden graafin tai graafeja, mikäli kohteita onkin enemmän kuin valmiina olevia graafeja.

Esimerkki on sen verran pitkä, että se on luettavissa liitteestä (Liite 2).

5.4 ROI-sovellus

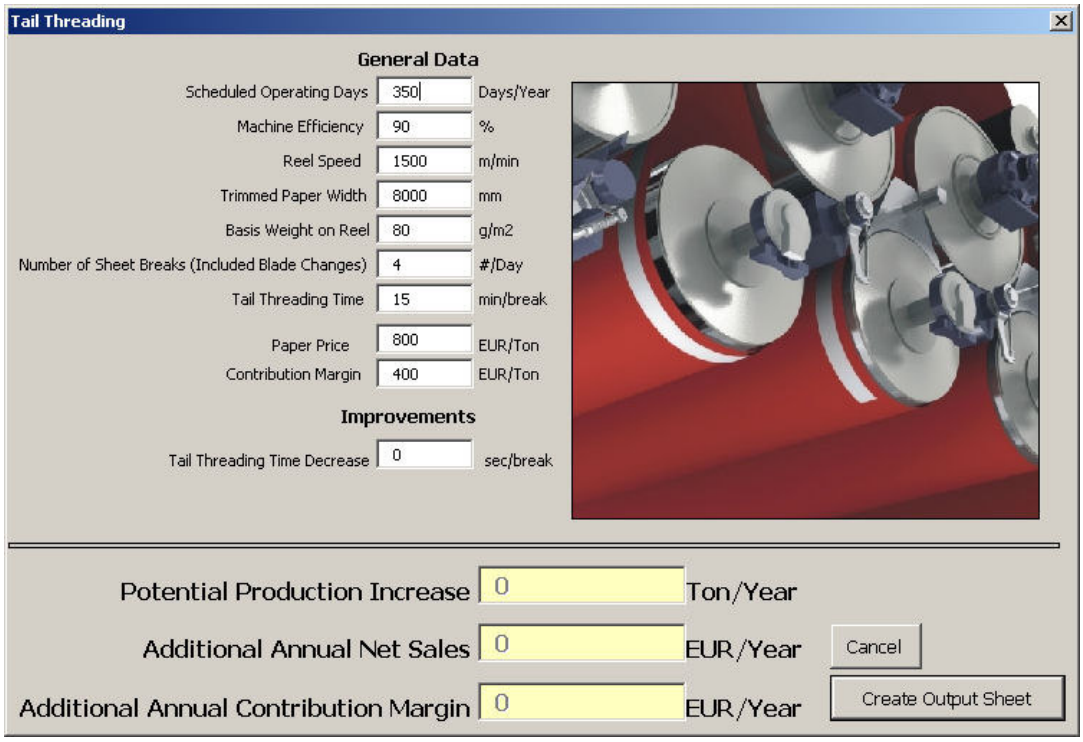
Usein on tarpeen laskea asiakkaalle, mitä parannukset paperilinjan tietyn osa-alueen tehokkuudessa vaikuttavat koko linjan tuottavuuteen. Tätä varten on käytössä useita erilaisia ROI-sovelluksia (Return On Investment). Itse olin mukana kehittämässä käyttöliittymää ja laskentaa liittyen papelinjan yleisprosessiin ja päänvientiin. Tärkeää tässä tapauksessa oli visuaalinen ulkoasu, yksinkertainen käyttöliittymä, laskukaavojen piilotus sekä mittayksikkömuunnosten huomioon ottaminen.

Kuvassa (Kuva 10) on esitetty ROI-sovelluksen alkuvalikko, joka fyysisesti sijaitseen Excelin laskentalomakkeella. Painamalla nappia Excel minimoituu ja



Kuva 10. ROI-sovelluksen käynnistyssivu.

kuvassa (Kuva 11) olevan kaltainen varsinainen sovelluksen käyttöliittymänä toimiva lomake tulee näkyviin. Tähän lomakkeeseen asiakas yhdessä asiantuntijan kanssa syöttää halutut laskentaparametrit ja itse laskenta tapahtuu samanaikaisesti Excelin taulukkolaskentaa ja VBA-funktioita hyödyntäen.



Kuva 11. ROI-sovelluksen laskentalomake.

Seuraavassa on esitetty, miten yksikkömuunnokset saadaan aikaan:

```
Private Sub UserForm_Initialize()
    'Units
    Me.lblReelSpeed.Caption = Units_Speed(UnitID)
    Me.lblTrimmedPaperWidth.Caption = Units_Width(UnitID)
    Me.lblAACM.Caption = Units_Currency(UnitID) & "/Year"
    Me.lblBasisWeight.Caption = Units_Weight(UnitID)
    Me.lblPPI.Caption = Units_Mass(UnitID) & "/Year"
'...
End Sub
```

Yllä olevassa, karsitussa aliohjelmassa, lomakkeen Initialize-tapahtuman käsittelijä (tapahtuu aina, kun lomake luodaan) muuttaa yksikkökenttien sisällön vastaamaan UnitID:n mukaista yksikköä, joka valitaan ohjelman käynnistyessä, käyttäen apunaan yksikkömuunnoksia varten kirjoitettuja funktioita. Esimerkiksi:

```
Function Units_Speed(UnitID As Integer)
'Returns unit for Speed in different unit system
  Select Case UnitID
    Case 1
```

```
        Units_Speed = "m/min"
    Case 2
        Units_Speed = "ft/min"
  End Select
End Function
```

Varsinaisten lomakkeelle syötettävien lukujen muunnokset tehdään tarvittaessa sisäisesti Excelissä, jossa laskenta suoritetaan aina ensin kymmenjärjestelmän yksiköillä. Kutakin tarvittavaa yksikköä vastaava muunnosfunktio kertoimineen on implementoitu Exceliin VBA:ta käyttäen.

```
Function k_SpeedToUnit1(UnitID As Integer)
'Returns a factor for speed from other unit to [m/min]

    Select Case UnitID
    Case 1
        k_SpeedToUnit1 = 1                  '[m/min]
    Case 2
        k_SpeedToUnit1 = 0.3048             '[ft/min]
    End Select
End Function
```

5.5 Huomioita VBA- ja Office-ympäristöistä ja arviointi tehdyistä ratkaisuista

Edellä olen esitellyt VBA:lla tehtyjä ratkaisuja, joita on tarkoitus käyttää useammin kuin kerran. Varsin usein syntyy kuitenkin tarvetta kirjoittaa VBA-ohjelmia, joita käytetään ainoastaan kerran. Hyvän esimerkkiryhmän muodostavat ohjelmat, joiden tehtävänä on etsiä ja korvata tiettyjä tietoja isoista tiedostoista, joissa kyseessä oleva tietoalkio saattaa esiintyä satoja tai jopa tuhansia kertoja. Näissä tapauksissa VBA on tehokas apuväline, yhdessä Excelin valmiiden ominaisuuksien kanssa, millä aikaansaadaan tehokkaita etsi ja korvaa automaatteja. Muita esimerkkejä tällaisista kertakäyttöisistä ohjelmista ovat tiedostojen sisältöjen vertailut tai tietyn sisällön siirtäminen toisessa muodossa uuteen paikkaan.

Jälkikäteen on helppoa huomata, että tekemissäni sovelluksissa viimeistely, optimointi ja virheidenkäsittely on jäänyt sivuseikaksi varsinaisten algoritmien suunnittelun ja

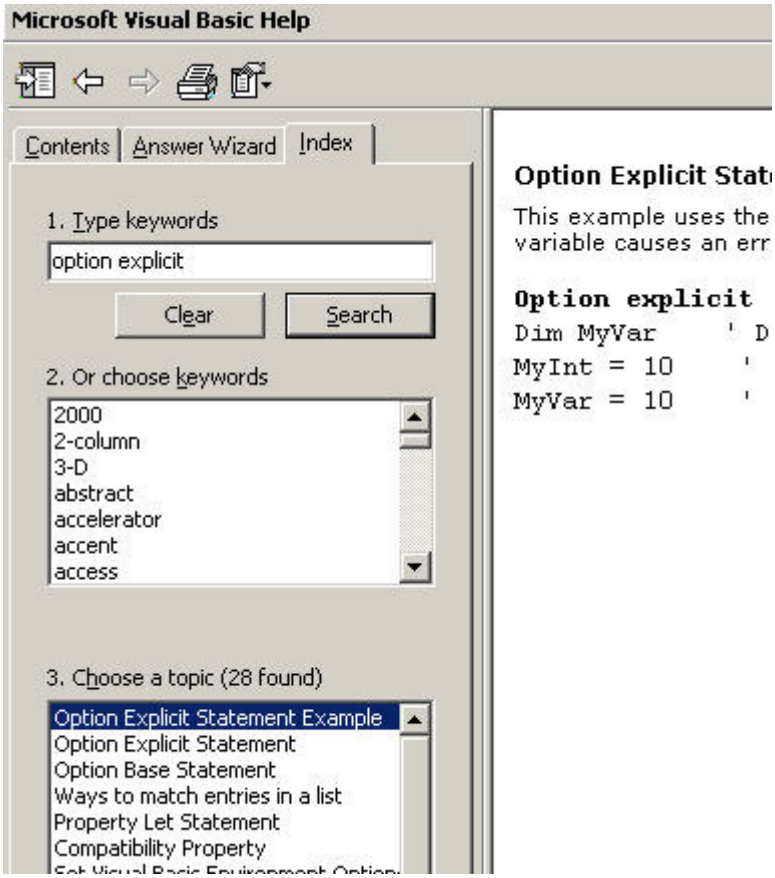
toiminnallisuuden implementoinnin viedessä suurimman osan huomiosta. Koodin siisteyttä, tehokkuutta ja suoritusvarmuutta saataisiin lisättyä melko vähällä vaivalla lisäämällä muutama objektimuuttuja (Katso liitteet 1 ja 2.), yksilöity virheenkäsittelijä sekä miettimällä enemmän muuttujien esittelyyn ja nimeämiseen sekä liittyviä säännönmukaisuuksia.

Vaikkakin VBA on implementoitu Officeen ensimmäisen kerran jo vuonna 1997 ja sen yhteensopivuuden Windowsin kanssa sanotaan olevan sen etu, on todettava, että käytännön tilanteissa ongelmia ei seuraa niinkään Officen ja Windowsin välisestä suhteesta kuin Officen eri versioiden välisestä yhteensopivuudesta. Olen töissä törmännyt lukuisiin tilanteisiin, joissa eri Office-versioiden välillä toimiminen aiheuttaa odottamattomia ongelmatilanteita ja pahimmassa tapauksessa jopa tiedostoversioiden katoamista. Joissakin tapauksissa virheet ovat itse VBA:n toiminnassa ja toisissa tapauksissa esimerkiksi lomake-elementtien ominaisuuksiin liittyvissä asioissa. Yhteensopimattomuustilanteita ilmeni ainakin ympäristössä, jossa käytettiin Windows XP –käyttöjärjestelmää yhdessä Officen versioiden 97 ja 2003 kanssa. VBA-ohjelmien suunnittelussa on selkeästi huomioitava se, että kehitystyö on syytä tehdä kokonaan tietyllä Officen versiolla ja sen jälkeen miettiä, miten halutaan toimia mahdollisissa yhteensopimattomuustilanteissa. Yleensähän tilanteen ratkaisee yritysmaailmassa se, että mikä on yrityksen virallinen ohjelmaversio. Ongelmalliseksi tilanne kehittyy vasta sitten, kun vanhoja sovelluksia pitäisi päivittää toimimaan uuden version alaisuudessa. Itse asiassa kohtaamani ongelmat olivat ainakin Internetin keskustelupalstoilla yllättävän huonosti tunnettuja tai ainakin niihin liittyvät ratkaisut olivat vaikeasti löydettävissä, jos niitä ylipäätään oli saatavilla.

Keskustelupalstat ja spesialistifoorumit tarjosivat VBA-ohjelmoinnin ongelmatilanteissa joitakin apukeinoja. Muut ongelmanratkaisulähteet olivat VBA-kirjallisuus, työtovereiden kokemus ja VBA-editorin oma help-toiminto, jonka avulla yllättävän moneen ongelmaan löytyi ratkaisu. Käytän tässä tilaisuutta hyväkseni ja moitin vähän Officen help-järjestelmän kehitystä Office-assistant –suuntaan (Kuva 12),

Kuva 12. Office-assistant.

joka mielestäni ainoastaan hankaloittaa ja hidastaa aputoimintojen tehokasta käyttöä. Office-avustajan saa pois käytöstä valitsemalla Options ja poistamalla rastin Use Office Assistant –kohdasta. Näin saa käyttöönsä perinteisemmän indeksoidun aputoiminnon (Kuva 13).



Kuva 13. Indeksoitu, perinteinen aputoiminto.

6 Vertailukohtina C++ ja Java

Seuraavaksi vertailen VB:tä toiseen tulkattavaan kieleen – Javaan ja opiskelujeni alkuhetken perusopetuskieleen C++:aan. Tarkoituksenani on ainoastaan tehdä vertailu siitä, miten nämä kielet eroavat toisistaan. En siis ainakaan tahallisesti tee mitään paremmuusehdotuksia.

Ensin kuitenkin esittelen vertailukohtina olevat kielet yleisesti.

6.1 C++

C++ on Bjarne Stroustrupin kehittämä olio-ohjelmointikieli, joka on kehitetty C-kielen pohjalta. C++ on lisäksi saanut vaikutteita Algolista, Simulasta, Adasta ja CLU:sta. Alunperin C++ kehitettiin lähes pelkästään Stroustrupin ja hänen ystäviensä käyttöön. Tarkoituksena oli pystyä kirjoittamaan hyviä ohjelmia aiempaa helpommin. [35, s. 16-19].

C++ laajentaa C-kieltä, johon on tehty joitain lisäyksiä sekä laajennus, joka sisältää olio-ohjelmointimekanismit. Olio-ohjelmointiin liittyviä ominaisuuksia ovat esimerkiksi kapselointi, olioiden luonti, moniperiytyminen, polymorfismi sekä STL-kirjasto. Muita C++:n ominaisuuksia ovat mm. viittausmuuttujat, aliohjelmien kuormitus sekä poikkeuskäsittely [35, s. 16-19].

C-kieli valittiin C++:n perustaksi monipuolisuutensa, selkeytensä sekä monipuolisen käyttösoveltuvuutensa takia. C++:n ensimmäisen version katsotaan syntyneen vuonna 1980, jolloin Stroustrup kehitti kielen, jonka nimenä silloin oli C with Classes. Tämän jälkeen C++:aa ovat kehittäneet useat henkilöt ennen kuin vuonna 1997 hyväksyttiin C++-kielen standardi ja sen standardikirjasto [35, s. 16-19].

6.2 Java

Java on James Goslingin ja Sun Microsystemsin vuonna 1991 kehittämä olio-ohjelmointikieli, jonka työnimenä oli Oak, Goslingin toimiston ulkopuolella kasvaneiden tammipuiden mukaisesti. Javan tarkoituksena oli korvata C++, vaikkakin sen ominaisuudet muistuttavatkin ehkä enemmän Objective-C -kieltä.

Java on tietoverkkoihin suunnattu olio-ohjelmointikieli, joka on rakennettu C++:n pohjalta ja jolle tyypillisiä ominaisuuksia ovat puhdas oliopohjaisuus, vahva tyypitys, automaattinen roskien keruu, tulkittavuus ja WWW-ohjelmointituki. Java kehitettiin alunperin sulautettujen järjestelmien tarpeita silmällä pitäen (ohjelmien kääntäminen uudelleen prosessorialustaa vaihdettaessa), mutta graafisten käyttöliittymien tulo Internet-päätteisiin avasi Javalle sen viimeiset vuodet dominoiman markkina-alueen [26, s.17].

Javaa ei pidä sekoittaa JavaScriptiin, jonka kanssa sillä on yhteistä ainoastaan nimi ja C-tyyppinen syntaksi. Sun Microsystems ylläpitää ja kehittää Javaa säännöllisesti. Javan spesifikaatiot, Java Virtual Machine (JVM) ja Java API ovat kaikki Sun Microsystemsin johtaman Java Community Process -yhteisön ylläpitämiä tuotteita [15].

6.3 Visual Basic, Java ja C++

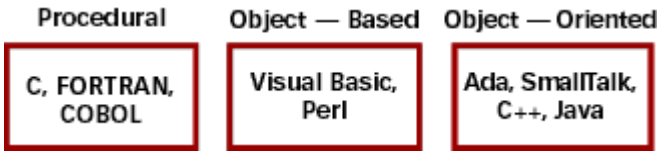
Riippuen hieman lähteestä ja siitä, mitä on tutkittu, saadaan otsikossa mainitut kolme kieltä suosituimpien listan kärkipäähän. Esimerkiksi vuonna 1999 tehdyn tutkimuksen mukaan 35 % ohjelmoijista käytti Visual Basicia, 20 % puolestaan valitsi C:n ja C++:n, Java oli tuolloin vertailun kolmas 9 %:n osuudellaan. Uudempia WWW-tilastoja [30, 31] selatessa on huomattavaa se, että vaikka yleisesti oletetaan Javan nousseen suosituimmaksi kieleksi, näin ei välttämättä olekaan [30]. C, C++ ja Visual Basic näyttävät säilyttäneen suositun asemansa myöskin tutkimuksessa, jossa Java on kärjessä [31].

Visual Basicin, Javan ja C++:n eroja ohjelmointikielinä voi lähestyä monesta eri näkökulmasta. Eräs tapa on jaotella ohjelmointikielet joihinkin ohjelmointiparadigmoihin. Kuvassa (Kuva 14) on esitetty eras jaottelu proseduraalisiin (procedural), oliopohjaisiin (object-based) ja oliosuuntautuneisiin (object-oriented) ohjelmointikieliin.

Proseduraalinen ohjelmointikieli tarkoittaa nimensä mukaisesti ohjelmointia, jonka keskeisenä osana ovat proseduurien, aliohjelmien tai funktioiden (procedures, subroutines, functions). kutsut. Nämä kutsuttavat osat sisältävät yksinkertaisia, yksitellen, tietyssä järjestyksessä suoritettavia askelia [34].

Oliosuuntautunut (object-oriented) ohjelmointi (tai tutummin olio-ohjelmointi) tarkoittaa ohjelmointikieltä, jossa ohjelma rakentuu itsenäisistä yksiköistä – olioista – toisin, kuin perinteinen proseduraalinen tapa, jossa ohjelma on ainoastaan lista ohjeita tietokoneelle. Olio-ohjelmoinnissa jokainen objekti pystyy vastaanottamaan ja lähettämään viestejä sekä prosessoimaan dataa [33].

Oliopohjaisella (object-based) ohjelmoinnilla tarkoitetaan yleisesti jollain tavoin rajoittunutta versiota oliosuuntautuneisuudesta. Tälläisia rajoitteita ovat implisiittisen perinnän ja polymorfismin puuttuminen sekä oliopohjaisuuden esiintyminen esimerkiksi ainoastaan GUI-komponenttien yhteydessä [32].



Kuva 14. Ohjelmointikielien eräs luokittelu [22].

Oliopohjaisuus näkyy VB:ssä varsin selvästi esimerkiksi lomakkeiden rakentamisessa. Vaikkakin tällaiseen objektiin pystyy viittaamaan ja sitä pystytään uudelleenkäyttämään, niin se ei silti pysty yksilöitymään tai siirtymään toiseen paikkaan. Kun objekti on kerran luotu Visual Basicissa, ohjelmoijalla on varsin rajoittuneet mahdollisuudet käyttää sitä hyväksi esimerkiksi perinnässä. Tässä ovatkin syyt, joiden pohjalta on perusteltua sijoittaa VB oliopohjaisten kielien ryhmään [22].

VB:n puutteista on jo puhuttu luvussa 4.1, mutta mainittakoon tässä vielä kertauksen vuoksi muutama ominaisuus, jotka ovat löydettävissä Javasta, mutta puuttuvat VB:stä.

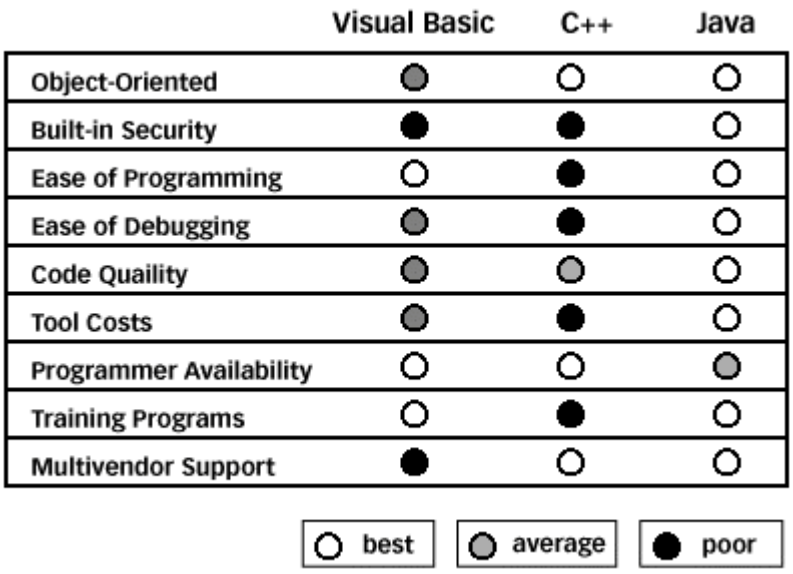
- Säikeet,
- verkko-ominaisuudet (http ja soketit) ja
- poikkeusten käsittely [22].

Ilman mainittuja ominaisuuksia VB itsessään ei pysty tarjoamaan nykyaikaisten Internet-pohjaisten sovellusten vaatimia asioita. Näin ollen vaadittaisiin ulkoinen rajapinta nämä

ominaisuudet tarjoavaan ympäristöön ja se puolestaan tarkoittaisi liiketoimintanäkökulmasta kasvavia kuluja vaadittavien taitojen, ohjelmointityökalujen ja debuggaus-työkalujen osalta. Tämä, Javan puolesta puhuva, oli tilanne vielä vuonna 1999 [22]. Sittemmin VB.NET on ottanut kritiikistä opikseen ja sisällyttänyt kieleen niin säikeet, poikkeusten käsittelyn kuin verkko-ominaisuudetkin.

Liiketoimintanäkökulmasta katsoen VB:n yhteydessä tulee ottaa huomioon myös Microsoftin tuomat vaikutukset esimerkiksi lisensseihin, käyttöjärjestelmä- ja laitevaatimuksiin sekä muihin mahdollisesti muuttuviin ennustamattomiin tekijöihin. Javan ja C++:n kohdalla sidonnaisuus yhteen tiettyyn toimijaan ei ole niin vahva [22] Tämä on jälleen kerran piirre, jonka positiivisuus tai negatiivisuus on punnittava tapauskohtaisesti. Pienyrittäjälle ilmainen on tosin aina ilmaista.

Kuvassa (Kuva 15) on tehty vertailua näiden kielien ja niiden tiettyjen ominaisuuksien välillä. Kuva on vuodelta 1999 ja kuvastaa sen aikaista tilannetta. Sanoisin, että nykyään tilanne ei ole lainkaan näin selkeä Javan eduksi – tosin kuvakin on jawaworldin sivuilta.



Kuva 15. Ohjelmointikielien vertailua [22].

7 Tulevaisuus

Seuraavassa pohditaan VBA:n tulevaisuudennäkymiä erilaisista näkökulmista lähtien.

7.1 Mitä Visual Basicin epävarma tulevaisuus tarkoittaa VBA:n osalta?

Oikeastaan ei yhtään mitään. VBA:n tulevaisuutta säätelee enemmänkin MS Officen tulevaisuus. VBA on ollut osa Officea vuodesta 1997, eikä se näytä poistuvan ainakaan vielä vuonna 2006. Itse asiassa VBA:han tuodaan lisää ominaisuuksia mm. IP:n (Intellectual property) suojaamiseksi. Oikeastaan on siis niin, että VB käy tällä hetkellä omaa selviytymistaisteluaan muutamia ohjelmointikieliä (lähinnä Java, C#) vastaan, kun taas VBA elää omaa elämäänsä kielenä, jolla valmiita toimistotuotteita on mahdollista personoida aina kullekin käyttötarkoitukselle sopivaksi. Näin ollen voitaneen katsoa VB:n ja VBA:n eriytyneen lopullisesti erillisiksi kielikseen, sillä en usko VB:n uudempien versioiden implementoituvan Office-alustaan. Tosin Microsoftin tapauksessa täytynee olla jonkin verran varovainen arvioissaan. VBA:n tulevaisuus on vielä hieman hämärän peitossa, mutta tällaisen räätälöintikielen poistumista Officesta ei enää voitane pitää mahdollisena.

Varsinaisen VB:n tulevaisuus sen sijaan näyttää epävarmalta Microsoftin .NET-rungon ja C#:n vallatessa jalansijaa markkinoilla. Microsoft julkaisi uuden sukupolven Windows-ympäristöön suunnitellun ohjelmointikielensä C#:n helmikuussa 2002 yhdessä uuden Visual Studio .NET:in kanssa [29].

Julkaisun jälkeen pessimistit ovat odottaneet Visual Basicin tuhoa, mutta vaikkakin tulevaisuus näyttää olevan C#:n, ei Visual Basic poistu suosiolla. Visual Basicin tilasta puhuttaessa on edelleenkin huomioitava, että se on yksi maailman suosituimmista ohjelmointikielistä liiketoimintakäytössä asiakas-palvelin sovelluksissa [29].

C# on kasvava kieli, ja se saavuttaa Visual Basicia, mutta vielä tällä hetkellä ero on varsin selvä VB:n hyväksi [29, 30, 31].

Mitä C#:lla sitten on tarjota, mitä VB.NET:llä ei? C# on kehittäjiensä mukaan suunniteltu alusta alkaen olio-ohjelmointikieleksi toisin kuin VB, johon olio-ominaisuudet ovat

tavallaan ajan mittaan käytännön tarpeista ja kritiikistä johtuen integroituneet. Näin ollen ongelmat, joita ohjelmoija pyrkii ratkaisemaan C#:ssa ovat enemmänkin liitoksissa itse olioihin kuin niitä käsittelevään logiikkaan [29].

Vaikka tällä hetkellä Visual Studio .NET tukeekin molempia kieliä, on odotettavissa, että tulevaisuus on C#:n, sillä se on suunniteltu olio-ohjelmoinnin tarpeita silmällä pitäen, mitä VB puolestaan ei ole. Näyttää siis siltä, että ollaan väistämättä tulossa pisteeseen, jossa VB ja sillä tehtävä ohjelmistokehitys on tullut tiensä päähän [29]. Yritysten näkökulmasta tämä merkitsee jälleen isoja ratkaisuja sen suhteen, milloin tulisi vaihtaa uuteen kieleen. Yksiselitteistä vastausta ei ole olemassa, mutta osaavat ihmiset ovat aina varma sijoitus tulevaisuuteen.

Toinen huolenaihe liittyy tietysti olemassa oleviin sovelluksiin, joiden päivittämisestä aiheutuva kustannuserä on sitä luokkaa, että asia on harkittava erittäin tarkasti. Ainakin C#:n koulutustarve on lisääntynyt akuutisti viimeisten kahden vuoden aikana. Javan mahdollinen rooli entisten VB:n käyttäjien kaappaajana on vielä arvoitus, mutta todennäköisintä on, että VB-käyttäjät siirtyvät C#:iin. Tämä siirtymä ei kuitenkaan tule tapahtumaan yhdessä yössä, vaan pitkän ajan kuluessa, joten VB ei tule poistumaan ainakaan seuraavien 4-5 vuoden aikana. Varmaa tekniikan maailmassa tosin ei ole juuri mikään [29].

8 Yhteenveto

Olen tässä tutkielmassa yrittänyt kertoa, mikä oikeastaan on VBA-ohjelmointikieli, miten se on syntynyt, miten sitä käytetään ja millainen on sen tulevaisuus.

VBA:n alkutaival nivoutuu kiinteästi yhteen Visual Basicin, joka puolestaan juontaa juurensa BASICistä, kanssa (Luku 3). Tästä se sitten on eriytynyt omaksi erityisesti toimistosovellusten räätälöintiin soveltuvaksi ohjelmointikielekseen.

Microsoft lupasi vuonna 1995, ennen Windows 95:n julkistamista, että ”Kaikessa toiminnassa pitää olla jotain järkeä”. Oikeastaan VBA on syntynyt tämän lupauksen pohjalta ja mahdollistanut Office-paketin sovellusten toiminnallisuuden lisäämisen vuodesta 1997 alkaen. VBA:n tutuin ilmentymä tavalliselle käyttäjälle ovat Excelin makrot, joita useat peruskäyttäjät itsekin tekevät käyttäen hyväksi Excelin makronauhoitinta. Tässä tutkielmassa olen perehtynyt kuitenkin enemmän VBA:n tietoteknisempään puoleen kirjoitettuna ohjelmointikielenä ja niinpä olen tutkielmani luvuissa esitellyt VBA:n hyviä ja huonoja puolia sekä VBA:n perusrakenteita ja ohjelmointiympäristöä, jonka avulla varsinainen koodin kirjoittaminen suoritetaan (Etenkin luku 4).

VBA:lla on paljon todettuja heikkouksia, mutta sen ehkä suurin vahvuus onkin juuri siinä, että sen käyttö on äärimmäisen helppoa ja nopeaa, ja siinä, että sen integroituvuus Officeen ja Windowsiin on tavallaan sisäänrakennettuna. Ohjelmointirakenteidensa puolesta VBA muistuttaa suuresti VB:tä tai oikeastaan mitä tahansa nykyaikaista ohjelmointikieltä. Suurin ero muihin kieliin tulee siinä, että VBA ei varsinaisesti tuota itsenäisiä sovelluksia vaan ohjelmia, jotka vaativat toimiakseen jonkin Office-ohjelman tuen.

VBA:ta käytetään työmaailmassa paljon. Siellä, missä on käytössä Office-ohjelmistoja, on myös tarvetta liiketoimikohtaiseen räätälöintiin ja siispä VBA-ohjelmointiin. Oma kokemukseni VBA:sta tulee paperikonevalmistaja Metso Paper Oy:n palveluksessa tehdyistä kesätöistä opintojeni aikana (Luku 5). Raporttien tuottaminen, datan kerääminen ja järjestäminen pelkästään käsin on käytännössä mahdotonta ja niinpä automaattisilla ratkaisuilla on tilausta.

Olen esitellyt muutamia käytännön ongelmia ja samalla tehdessäni tutkimusta huomannut, että tekemämme ratkaisut olisivat voineet olla vieläkin tehokkaampia kuin tässä tutkielmassa ja sen liitteissä esitetyt VBA-ohjelmat ovat. Tätä olen kommentoinut tarkemmin luvuissa 4.6 ja 5.5.

Jotta VBA säilyisi elinvoimaisena kielenä, on versioyhteensopimattomuuksista aiheutuvat ongelmat otettava paremmin huomioon Officen kehitystyössä kuin nyt on tehty esimerkiksi versioiden 97 ja 2003 välillä (Luku 5.5). Näin sovelluksiin pystytään tuottamaan sitä lisäarvoa, jota yritykset tarvitsevat kukin omassa toimintakentässään – ja opiskelijat sekä muut yksityiskäyttäjät oppilaitoksissa ja kotonaan. Lisäksi on tietysti huolehdittava siitä, että kieleen saadaan lisää käyttäjien kaipaamia ominaisuuksia.

Jää nähtäväksi, mihin suuntaan VBA tulee kehittymään tulevaisuudessa, kun sen tulevaisuus kielenä tuntuu eriytyneen lopullisesti VB:stä ja VB:n puolestaan on ennustettu häviävän kokonaan Javan ja C#:n painostuksessa, vaikka tämä ei vielä tilastoista olekaan luettavissa (Luku 7). On kuitenkin selvää, että VBA-kielen kaltaiselle tehokkaalle ja helppokäyttöiselle työkalulle on tilausta toimistoissa, kodeissa ja oppilaitoksissa ympäri maailman.

Lähteet

[1] Scott D.F., ”MS Office 2000 -sovelluskehitys”, Teknolit Oy, Jyväskylä, 2000.

[2] Ek Jesper, Eriksson Ulrika ja Isanovic Senad, ”VBA-ohjelmointi office 2000:ssa”, Pagina, Helsinki, 2000.

[3] Microsoft Press, ”Microsoft Office 97 Visual Basic -ohjelmointi”, Satku, Helsinki, 1997.

[4] Leppämaa Kirsti, ”Visual Basic 6”, Teknolit, Jyväskylä, 1999.

[5] Walkenbach John, ”Excel 2000”, Teknolit, Jyväskylä, 2000.

[6] Väisänen Kaarlo, Kentala Jussi-Pekka, Ensio Sami ja Laiho Rami, ”Microsoft Office 2000 -opas”, Satku, Helsinki, 2000.

[7] Lammi Outi ja Pulkkinen Heljä, ”Word&Excel MS Officen tehokäyttö”, Teknolit, Jyväskylä, 1999.

[8] Kemeny John G. ja Kurtz Thomas E., ”Back to BASIC: the history, corruption and future of the language”, Addison-Wesley, Reading, Mass., 1985.

[9] Dijkstra Edsger, “*Communications of the ACM”,* Volume 11, 147-148. 1968, saatavilla HTML-muodossa osoitteessa <URL: `http://www.acm.org/classics/oct95/>,` viitattu 14.9.2005.

[10] Wikipedia, Vapaa tietosanakirja, Microsoft Office, Word ja Excel, saatavilla HTML-muodossa osoitteessa <URL:
`http://fi.wikipedia.org/wiki/Microsoft_Office,`
`http://fi.wikipedia.org/wiki/Microsoft_Word,`
`http://en.wikipedia.org/wiki/Microsoft_Office,`
`http://en.wikipedia.org/wiki/Microsoft_Excel>,` viitattu 7.9.2005.

[11] Wikipedia, Vapaa tietosanakirja, Lotus Notes, saatavilla HTML-muodossa osoitteessa <URL: http://en.wikipedia.org/wiki/Lotus_Notes>, viitattu 7.9.2005.

[12] Wikipedia, Vapaa tietosanakirja, VBA, saatavilla HTML-muodossa osoitteessa <URL: http://en.wikipedia.org/wiki/Visual_Basic_for_Applications>, viitattu 7.9.2005

[13] Wikipedia, Vapaa tietosanakirja, BASIC, saatavilla HTML-muodossa osoitteessa <URL: http://en.wikipedia.org/wiki/BASIC_language>, viitattu 7.9.2005.

[14] Wikipedia, Vapaa tietosanakirja, Visual Basic, saatavilla HTML-muodossa osoitteessa <URL: http://en.wikipedia.org/wiki/Visual_BASIC>, viitattu 7.9.2005.

[15] Wikipedia, Vapaa tietosanakirja, Java, saatavilla HTML-muodossa osoitteessa <URL: http://en.wikipedia.org/wiki/Java>, viitattu 7.9.2005.

[16] Wikipedia, Vapaa tietosanakirja, ActiveSync, saatavilla HTML-muodossa osoitteessa <URL: http://en.wikipedia.org/wiki/Activesync>, viitattu 7.9.2005.

[17] Computer Languages History, saatavilla HTML-muodossa osoitteessa <URL: http://www.levenez.com/lang/history.html - 05>, viitattu 29.9.2005.

[18] BASIC programming language, saatavilla HTML-muodossa osoitteessa <URL: http://www.answers.com/topic/basic-programming-language>, viitattu 29.9.2005.

[19] Koikkalainen Pasi ja Orponen Pekka, ”Tietotekniikan perusteet, Luentomoniste 7”, Jyväskylän yliopistopaino, Jyväskylä, 2001.

[20] Kemeny John, Kurtz Thomas, ”BASIC”, Darthmouth Gollege Computation Center, 1964, saatavilla PDF-muodossa osoitteessa <URL:

http://www.bitsavers.org/pdf/dartmouth/BASIC_Oct64.pdf>, viitattu 7.9.2005.

[21] Visual Basic programming language, saatavilla HTML-muodossa osoitteessa <URL: http://www.answers.com/main/ntquery?method=4&dsid=2222&dekey=Visual+Basic&gwp=8&curtab=2222_1&linktext=Visual%20Basic>, viitattu 29.9.2005.

[22] The business case for Java: A primer for geeks, Part 1, saatavilla HTML-muodossa osoitteessa <URL: http://www.javaworld.com/javaworld/jw-01-2000/jw-01-business-p2.html>, viitattu 29.9.2005.

[23] Tucker Allen B., ”Programming Languages”, McGraw-Hill, New York, 1985.

[24] Mayer Herbert G., “Programming Languages”, Macmillan Publishing Company, New York, 1988.

[25] Sethi Ravi, “Programming Languages: concepts and constructs ”, Addison-Wesley, Reading, Mass., 1997.

[26] Vesterholm Mika ja Kyppö Jorma, “Java-ohjelmointi”, Satku, Helsinki, 2001.

[27] Using Variables, Constants, and Data Types, saatavilla HTML-muodossa osoitteessa <URL: http://www.quepublishing.com/articles/article.asp?p=339477&seqNum=2>, viitattu 14.9.2005.

[28] Louden Kenneth C., ”Programming Languages, Principles and Practice”, Thomson-Brooks/Cole, Pacific Grove, 2003.

[29] Eric B. Parizo, “Visual Basic faces uncertain future in .NET world”, saatavilla HTML-muodossa osoitteessa <URL: http://searchvb.techtarget.com/tip/1,289483,sid8_gci914179,00.html>, 7.10.2003.

[30] David N. Welton, ”Programming Language Popularity”, saatavilla HTML-muodossa osoitteessa <URL: http://www.dedasys.com/articles/language_popularity.html>, 24.9.2004.

[31] Programming Language Popularity: The TCP Index for August, 2004, saatavilla HTML-muodossa osoitteessa <URL: http://www.developer.com/lang/article.php/3390001>, 08/2004.

[32] Wikipedia, Vapaa tietosanakirja, Object-based programming, saatavilla HTML-muodossa osoitteessa <URL: http://en.wikipedia.org/wiki/Object-based_programming>, viitattu 3.10.2005.

[33] Wikipedia, Vapaa tietosanakirja, Object-oriented programming, saatavilla HTML-muodossa osoitteessa <URL: http://en.wikipedia.org/wiki/Object-oriented_programming>, viitattu 3.10.2005.

[34] Wikipedia, Vapaa tietosanakirja, Procedural programming, saatavilla HTML-muodossa osoitteessa <URL: http://en.wikipedia.org/wiki/Procedural_programming>, viitattu 3.10.2005.

[35] Hietanen Päivi, ”C++ ja olio-ohjelmointi”, Teknolit, Jyväskylä, 2000.

Liitteet

Liite 1. Luvun 5.2.2 ongelmaan liittyvä VBA-koodi kokonaisuudessaan

Liite 2. Luvun 5.3 ongelmaan liittyvä VBA-koodi kokonaisuudessaan

Liite 3 Microsoft Excelin Worksheets- ja Charts-objektiluokkien rakenne (luku 4.3.1)

Liite 1. Luvun 5.2.2 ongelmaan liittyvä VBA-koodi kokonaisuudessaan

```
Dim s As Object              's = CreateObject("Notes.Notessession")
                             '-> s on Notessessio-objekti
Dim db As Object             'Set db = s.getdatabase("", ""); Call db.openmail
                             '-> db on postikanta
Dim doc As Object            'Set doc = db.createdocument -> doc on uusi dokumentti
Dim rtitem As Object         'Set rtitem = doc.CreateRichTextItem("Message")
                             '-> ritem on kirjoittamamme viesti
Dim xlapp As Object          'Excel application object
Dim fname As String          'File name
Dim MetsoID As String        'MetsoID
Dim Recommendation As String 'Recommendation
Dim customer As String       'customer name
Dim pm As String             'papermachine and line
Dim performer As String      'analysis performer (metso contact)
Dim ProDate As String        'project date
Dim CustRep As String        'customer representative
Dim headl As String          'headline = customer + pm

Private Sub ParseXML()

' Macro written by Ville Koskiniemi 8.6.2005
' Modified 1.7.2005
' Modified for printing, added comments 23.11.2005, VK

    Dim counter As Integer      'counter for xml recommendations
    Dim UserName As String      '
    Dim msg  As String          'error message

    On Error GoTo errorhandler  'sets the error handler
    msg = "Problem creating Notes Session - Is your Logon active?"

    'create notes session object
    Set s = CreateObject("Notes.Notessession")
    'set maildatabase name #####Muutetaan tilanteen mukaan
```

```
Set db = s.getdatabase("", "usersmaildatabase.nsf")

'checking status
If db.IsOpen = True Then
      'Already open for mail
Else
  Call db.OPENMAIL
End If

'get username
UserName = LCase(s.CommonUserName)
If UserName = "" Then UserName = "METSO"
'replacing space with full stop
UserName = ReplaceCharacters(UserName, " ", ".") + "@metso.com"


' Ratkaisuissa on käytetty useasti muotoa, jossa
' Range-objekteihin viitataan käyttämällä
' xlapp.Workbooks(fname).Sheets(1) -objektiviittausta.
' Kuten tutkielmani luvussa 5.5. olen maininnut, optimaalisempi
' tapa tehdä sama olisi sijoittaa usein käytettävä viittaus
' objektimuuttujaan esimerkiksi näin:
' Dim FrequentlyUsedSheet as Object
' FrequentlyUsedSheet = xlapp.Workbooks(fname).Sheets(1)
' Näin seuraava sijoituslause tulisi muotoon:
' counter = FrequentlyUsedSheet.Range("g1").Value - 1
' ja vähentäisi kaksi siirtymistä objektihierarkiassa.

' Setting counter (to customer row count)
counter = xlapp.Workbooks(fname).Sheets(1).Range("g1").Value - 1

On Error GoTo errorhandler
' Muutetaan virheviestiä - viesti kertoo, missä kohdassa mahdollinen virhe
' on tapahtunut, kun varsinaista virheenkäsittelyä ei korjaavassa mielessä
' ole implementoitu. Ks. tutkielman luku 5.5.
msg = "Problem creating HeaderData"

'calling sub
```

```
Call HeaderData

'####Ensimmäinen/alkuperäinen ratkaisu alkaa
'writing the message (xml notation: VK, TN 06/2005)
Call rtitem.AppendText("<recommendations>")
For i = 1 To counter
   headl = CStr(xlapp.Workbooks(fname).Sheets(1).Range("A" & i + 6).Value + _
   ";" + xlapp.Workbooks(fname).Sheets(1).Range("B" & i + 6).Value)
   MetsoID = CStr(xlapp.Workbooks(fname).Sheets(1).Range("C" & i + 6).Value)
   ProDate = CStr(xlapp.Workbooks(fname).Sheets(1).Range("D" & i + 6).Value)
   performer = CStr(xlapp.Workbooks(fname).Sheets(1).Range("E" & i + 6).Value)
   CustRep = CStr(xlapp.Workbooks(fname).Sheets(1).Range("F" & i + 6).Value)
   recomm = CStr(xlapp.Workbooks(fname).Sheets(1).Range("G" & i + 6).Value)

   Call rtitem.AddNewLine(1)
   On Error GoTo 0
   Call rtitem.AppendText("<rec index=" + Chr(34) + CStr(i) + Chr(34) + ">")
   Call rtitem.AddNewLine(1)
   Call rtitem.AppendText("<headline>")
   Call rtitem.AppendText(headl)
   Call rtitem.AppendText("</headline>")
   Call rtitem.AppendText("<metsoid>")
   Call rtitem.AppendText(MetsoID)
   Call rtitem.AppendText("</metsoid>")
   Call rtitem.AddNewLine(1)
   Call rtitem.AppendText("<date>")
   Call rtitem.AppendText(ProDate)
   Call rtitem.AppendText("</date>")
   Call rtitem.AddNewLine(1)
   Call rtitem.AppendText("<source>")
   Call rtitem.AppendText(performer)
   Call rtitem.AppendText("</source>")
   Call rtitem.AddNewLine(1)
   Call rtitem.AppendText("<custrep>")
   Call rtitem.AppendText(CustRep)
   Call rtitem.AppendText("</custrep>")
   Call rtitem.AddNewLine(1)
   Call rtitem.AppendText("<recbody>")
```

```
   Call rtitem.AddNewLine(1)
   Call rtitem.AppendText(Recommendation)
   Call rtitem.AddNewLine(1)
   Call rtitem.AppendText("</recbody>")
   Call rtitem.AddNewLine(1)
   Call rtitem.AppendText("</rec>")
Next
   Call rtitem.AppendText("<recommendations/>")
'####Ensimmäinen/alkuperäinen ratkaisu loppuu

'XML-notaatio ei tuonut lisäarvoa, joten päädyimme lopulta
'yksinkertaiseen ratkaisuun, jossa data lähetettiin merkkijonona
'puolipisteillä eroteltuna.
'####Toinen ratkaisu alkaa

'Lisätty objektimuuttuja optimointia varten
Dim FrequentlyUsedSheet As Object
FrequentlyUsedSheet = xlapp.Workbooks(fname).Sheets(1)

For i = 1 To counter
     headl = CStr(FrequentlyUsedSheet.Range("A" & i + 6).Value + ";" _
     + FrequentlyUsedSheet.Range("B" & i + 6).Value)
     MetsoID = CStr(FrequentlyUsedSheet.Range("C" & i + 6).Value)
     ProDate = CStr(FrequentlyUsedSheet.Range("D" & i + 6).Value)
     performer = CStr(FrequentlyUsedSheet.Range("E" & i + 6).Value)
     CustRep = CStr(FrequentlyUsedSheet.Range("F" & i + 6).Value)
     recomm = CStr(FrequentlyUsedSheet.Range("G" & i + 6).Value)

     RecommendationRow = CStr(i) + ";" + headl + ";" + MetsoID + ";" + _
     ProDate + ";"
     RecommendationRow = RecommendationRow + performer + ";" + CustRep + _
     ";" + recomm + ";"

     Call rtitem.AppendText(RecommendationRow)
     Call rtitem.AddNewLine(1)
Next
'####Toinen ratkaisu loppuu
```

```
        On Error GoTo errorhandler
        msg = "Problem sending mail"

        'calling send sub
        Call SendMail

    Exit Sub ' avoid errorhandler if code succesfull

errorhandler:
 'Virheenkäsittelijä, joka näyttää viestin ja jatkaa suoritusta virhettä
 'seuranneelta riviltä.
 MsgBox (msg)
 Resume Next

End Sub

Private Sub SendMail()

' Macro written by Ville Koskiniemi 8.6.2005, ville.koskiniemi@metso.com
' Last modified 22.6.2005
' Modified for printing, added comments 23.11.2005, VK

    'set the sender that shows in the "who"-field
    doc.principal = "MSSReportMacro"

    'set the receiving address #######Tähän NOTES-kannan käsittelijän osoite
    Call doc.Send(False, "database.agent@company.com")

    ' cleanup objects
    Set db = Nothing
    Set doc = Nothing
    Set s = Nothing
    Set rtitem = Nothing

End Sub

Private Sub HeaderData()
```

```
' Modified by Ville Koskiniemi 1.7.2005

    Set doc = db.createdocument
    Set rtitem = doc.CREATERICHTEXTITEM("Body")

    doc.Form = "Memo"
    doc.Subject = "MSS Analysis Report"
    doc.NotesMessageType = "TEXT"

End Sub
```

Liite 2. Luvun 5.3 ongelmaan liittyvä VBA-koodi kokonaisuudessaan

```
Option Explicit
Public FileName As String       'name of the current workbook
Sub get_xml()

'Macro written 16.6.2005 by Ville Koskiniemi.
'Modified 23.6.2005, VK
'Modified 19.8.2005, VK, added new features and one level of hierarchy (kohde, target)
'Last modified 22.8.2005, VK, renewed solution for getting the within loops to work properly
'Added comments 23.8.2005, VK
'Modified for printing, added comments 23.11.2005, VK

'Opens the xml-file of specified name, finds specific tags, gets the
'contents and puts it in the excel-cells.
'After that graphics is drawn - each target on individual sheet.
'19.8.2005 added functionality for each target to contain max 6 measurepoints.


    Dim Content As String        'actual information between the xml-tags
    Dim TagToFind As String      'tag name to find
    Dim NumberOfTags As Integer  'number of tags
    Dim i As Integer             'loop index
    Dim j As Integer             'loop index
    Dim k As Integer             'loop index
    Dim l As Integer             'loop index
    Dim MeasurePoints As Integer 'number of mp:s in one target
    Dim Targets As Integer       'number of targets in one measure
    Dim row As Integer           'row index
    Dim col As Integer           'column index
    Dim counter As Integer       'help counter for vba loop calculation
    Dim XMLFileName As String    'name of the XML-file
    Dim InfoRows As Integer      'number of "header-info" rows (tekijä, alkuaika yms.)
    Dim DataRows As Integer      'number of real data rows (kohteet)
    Dim mem1 As Integer          'variable for storing the current loop index
```

```
'Aktivoidaan virheenkäsittelijä, joka vain näyttää virheviestin.
On Error GoTo errorhandler

'get the name of active file
FileName = ActiveWorkbook.Name
'name of the XML-file - adjust according to the current environment
XMLFileName = "writer.xml"

'etsittävien tagien määrä (solusta)
NumberOfTags = Workbooks(FileName).Sheets("Asetukset").Range("ASXML_TagiLKM").Value

'avataan xml-file tekstimuotoisena ja rivitettynä
Workbooks.OpenText FileName:= _
    "C:\vilkoski\work_related\activeSync\" + XMLFileName, Origin:= _
    xlWindows, StartRow:=1, DataType:=xlDelimited, TextQualifier:= _
    xlDoubleQuote, ConsecutiveDelimiter:=False, Tab:=True, Semicolon:=False, _
    Comma:=False, Space:=False, Other:=False, FieldInfo:=Array(1, 1)

'if the settings are empty, macro gets them from file
'tämä kommentteihin, jos halutaan kirjoittaa asetukset käsin!!
If Workbooks(FileName).Sheets("Asetukset").Range("ASXML_GraafiLKM").Value = 0 Then
  Call get_settings
End If

'mittapisteiden määrä (solusta)
Targets = Workbooks(FileName).Sheets("Asetukset").Range("ASXML_GraafiLKM").Value

'seuraavan datarivin numero
row = Workbooks(FileName).Sheets("Asetukset").Range("ASXML_DatariviLKM").Value + 2
'asetetaan muuttujat
col = 1
counter = 0

'jokaiselle tagille tehdään silmukka
'6:sta tagista eteenpäin erityiskäsittely sisäsilmukassa
For i = 1 To NumberOfTags - 5

  Workbooks(XMLFileName).Activate
```

```
'etsii ko. tagin, purkaa sisällön ja siirtää sen soluun;
'parametreina tagin nro, rivi-ind ja sarakeind.
TagToFind = Workbooks(FileName).Sheets("Asetukset").Range("C" & i + 4).Value
Call find_and_store_content(TagToFind, row, col)

'jokaisen kohteen osalta:
If i = NumberOfTags - 5 Then
For j = 1 To Targets

  'otetaan käyttöön sen hetkinen tagin osoittava indeksi
  'mem1 = i + 1 - vanha tapa
  mem1 = 10 'sarake J - uusi tapa
  'uusi rivi aloitetaan sarakkeesta F (kohdenimi)
  col = 6

  'haetaan tagit kohde ja kohdeId
  For l = 1 To 2
    'ei varsinaisen tagiluettelon (c) vaan kohdenimen (j) mukaan, sillä
    'muuten toisesta kierroksesta (luettelon järjestämisen jälkeen) menee pieleen
    'TagToFind = Workbooks(FileName).Sheets("Asetukset").Range("C" & mem1 + 4).Value
    TagToFind = Workbooks(FileName).Sheets("Asetukset").Cells(j + 4, mem1)
    'Range("J" & mem1 + 4).Value
    Call find_and_store_content(TagToFind, row, col)
    mem1 = mem1 + 1
    col = col + 1
  Next

 'haetaan kyseisen kohteen mittauspisteiden lukumäärä
 '(listan tulee olla järjestämätön, koska muuten haku tehdään
 'väärän lukumäärän mukaan ja näin kaikki siitä eteenpäin menee väärin -
 ' tämä mahdollisuus siis 2-n ajokerralla, ensimmäisellä kerralla menee oikein)
 MeasurePoints = Workbooks(FileName).Sheets("Asetukset").Range("L" & j + 4).Value

 'asetetaan mem1 vastaamaan rivinumerointia:
 mem1 = i + 3

  'jokaiselle mittauspisteelle:
  For k = 1 To MeasurePoints * 3 'mp sis. 3 tagia
```

```
      TagToFind = Workbooks(FileName).Sheets("Asetukset").Range("C" & mem1 + 4).Value
      Call find_and_store_content(TagToFind, row, col)
      mem1 = mem1 + 1
      col = col + 1
      counter = counter + 1

      'lasketaan tagi-indeksiä kolmen sykleissä (mittapiste, oletusarvo, tulos)
      If counter = 3 Then
        mem1 = mem1 - 3
        counter = 0
      End If

    Next

    'siirrytään seuraavaan kohteeseen (ja uudelle riville)
    row = row + 1
  Next
  End If

  'seuraava sarake (yleistagit 1-5)
  col = col + 1
Next

'suljetaan xml-file
Workbooks(XMLFileName).Activate
ActiveWindow.Close

' Huomautin jo edellisen liitteen esimerkissä, että
' ratkaisuissa on käytetty useasti muotoa, jossa
' Range-objekteihin viitataan käyttämällä nyt
' Workbooks(FileName).Sheets("Asetukset/Data") -objektiviittausta.
' (Edellisessä esimerkissä jopa
' xlapp.Workbooks(fname).Sheets(1) -objektiviittausta.)
' Kuten tutkielmani luvussa 5.5. olen maininnut, optimaalisempi
' tapa tehdä sama olisi sijoittaa usein käytettävä viittaus
' objektimuuttujaan esimerkiksi näin:
' Dim FrequentlyUsedSheet as Object
' FrequentlyUsedSheet = Workbooks(FileName).Sheets("Asetukset/Data")
```

```
' Näin seuraavat sijoituslauseet tulisivat muotoon:
' muuttuja = FrequentlyUsedSheet.Range("nimetty_alue").Value
' ja vähentäisi siirtymistä objektihierarkiassa.

'muokataan excel kuntoon myös viiden ensimmäisen sarakkeen kohdalta
InfoRows = Workbooks(FileName).Sheets("Asetukset").Range("ASXML_InfoLKM").Value
DataRows = Workbooks(FileName).Sheets("Asetukset").Range("ASXML_DatariviLKM").Value
If InfoRows < DataRows Then 'tehdään, jos lisättiin enemmän kuin yksi rivi
  Workbooks(FileName).Sheets("Data").Range("A" & InfoRows + 1 & ":E" & InfoRows + 1).Copy
  Workbooks(FileName).Sheets("Data").Select
  Range("A" & InfoRows + 2 & ":E" & DataRows + 1).Select
  Workbooks(FileName).ActiveSheet.Paste
End If

'järjestetään positiolista kohdeid:n mukaan järjestykseen (tarvitaan, jotta
'piirretään oikeat arvot oikeaan positioon)
'toinen vaihtoehto on hoitaa piirtämistä vastaavien arvojen etsiminen findilla
'kohdeId:n mukaan, jos tätä järjestystä ei saa muuttaa
'Workbooks(FileName).Activate
Sheets("Asetukset").Select
Range("J5:M" & Targets + 4).Select
Selection.Sort Key1:=Range("K5"), Order1:=xlAscending, Header:=xlGuess, _
    OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
Workbooks(FileName).Sheets("Asetukset").Range("J" & Targets + 5).Select


'Workbooks(FileName).Activate
Sheets("Data").Select
Range("Home").Select

'järjestetään file kohdeId:n ja alkuajan mukaan
'(pitää tehdä, jotta piirtäminen voidaan suorittaa)
Selection.Sort Key1:=Range("G2"), Order1:=xlAscending, Key2:=Range("D2") _
    , Order2:=xlAscending, Header:=xlGuess, OrderCustom:=1, MatchCase:= _
    False, Orientation:=xlTopToBottom

'piirretään graafit
Call draw_graphs
```

```
    'to avoid the handler
    Exit Sub

errorhandler:
    MsgBox ("Virhe: " & Err)
    Exit Sub

End Sub

Sub draw_graphs()

' Macro written 17.6.2004 by Ville Koskiniemi
' Modified 23.6.2005
' Last modified 23.8.2005
' Modified for printing, added comments 23.11.2005, VK

' May need some graphs to be adjusted manually.


Dim SheetName As String     'current sheetname
Dim DataRows As Integer     'count of datarows
Dim GraphCount As Integer   'count of graphs
Dim i, n As Integer         'loop index
Dim j As Integer            'loop index
Dim k As Integer            'loop index
Dim SeriesName  As String   'series name
Dim counter As Integer      'help counter for vba loop
Dim NewsheetName  As String 'name of the new chart sheet
Dim Found As Boolean        'flag for finding settings
Dim FindRow As Integer      'starting row of the "find"
Dim DataName  As String     'datasheet name
Dim SourceData As String    'variable for setting the source data for graphs
Dim row As Integer          '
Dim ChartType As String     'variable for chart type
Dim Exists As Boolean       'testing for graph to allready exist
Dim Xname, Yname As String  'names of the axes
Dim GraphName As String     'name of the graph
```

```
Dim GraphType As String     'variable for graph type (xlLine)
Dim FId, NId As String      'variables for defining the drawing area for each graph
                            'FId=First Id to look for, NId = next Id to look for
Dim FRow, NRow As Integer   'row numbers related to FId and NId
Dim MeasurePoints As Integer 'number of measure points

On Error GoTo errorhandler

FileName = ActiveWorkbook.Name

'graafien lkm - getting the graph count
GraphCount = Workbooks(FileName).Sheets("Asetukset").Range("ASXML_GraafiLKM").Value
'asettaa piirtoalueen pituuden
'(nyt huomioitu myös viimeinen rivi; ei tarvitse enää, kun luetaan useita rivejä)
DataRows = Workbooks(FileName).Sheets("Asetukset").Range("ASXML_DatariviLKM").Value
'datasheet name - change if needed!!
DataName = "Data"

'if only one file of data has been read (there is nothing to draw)
If DataRows <= GraphCount Then
  Exit Sub
End If

'start setting search from row
FindRow = 5
Found = False
'start Id search from row 2
FRow = 2
FId = Workbooks(FileName).Sheets("Data").Range("G2").Value

Application.ScreenUpdating = False

'For each graphics
For i = 1 To GraphCount

  'haetaan kyseisen kohteen mittauspisteiden lukumäärä
  MeasurePoints = Workbooks(FileName).Sheets("Asetukset").Range("L" & i + 4).Value
```

```
  'gets the first and last occurrence of target Id in FId and saves the
  'row numbers for later
  NRow = FRow
  Do Until Found = True
    If Workbooks(FileName).Sheets("Data").Range("G" & NRow).Value = FId Then
      NRow = NRow + 1
    Else
      NId = Workbooks(FileName).Sheets("Data").Range("G" & NRow).Value
      FId = NId
      Found = True
    End If
  Loop

're-setting the flag for different use
Found = False
'setting the beginnin row for each loop round
FindRow = 5


  ' Huomautin jo edellisen liitteen esimerkissä, että
  ' ratkaisuissa on käytetty useasti muotoa, jossa
  ' Range-objekteihin viitataan käyttämällä nyt
  ' Workbooks(FileName).Sheets("Asetukset/Data") -objektiviittausta.
  ' (Edellisessä esimerkissä jopa
  ' xlapp.Workbooks(fname).Sheets(1) -objektiviittausta.)
  ' Kuten tutkielmani luvussa 5.5. olen maininnut, optimaalisempi
  ' tapa tehdä sama olisi sijoittaa usein käytettävä viittaus
  ' objektimuuttujaan esimerkiksi näin:
  ' Dim FrequentlyUsedSheet as Object
  ' FrequentlyUsedSheet = Workbooks(FileName).Sheets("Asetukset/Data")
  ' Näin seuraavat sijoituslauseet tulisivat muotoon:
  ' muuttuja = FrequentlyUsedSheet.Range("alue").Value
  ' ja vähentäisi siirtymistä objektihierarkiassa.

  'count of data series in graph (2-12)
  'SerCount = Workbooks(FileName).Sheets("Asetukset").Range("L" & i + 4).Value
  GraphType = Workbooks(FileName).Sheets("Asetukset").Range("M" & i + 4).Value
```

```
'Getting the right options for a graph from 'Asetukset'
Do Until Found
  If Workbooks(FileName).Sheets("Asetukset").Range("F" & FindRow).Value = GraphType Then
    ChartType = Workbooks(FileName).Sheets("Asetukset").Range("G" & FindRow).Value
    SourceData = Workbooks(FileName).Sheets("Asetukset").Range("G" & FindRow + 4).Value
    Xname = Workbooks(FileName).Sheets("Asetukset").Range("G" & FindRow + 5).Value
    Yname = Workbooks(FileName).Sheets("Asetukset").Range("G" & FindRow + 6).Value

    'name of the sheet and graph from the listing of the targets
    NewsheetName = "Position " + CStr(i)
    GraphName = Workbooks(FileName).Sheets("Asetukset").Range("J" & i + 4).Value
    Found = True
  End If
  'settings of each graph take 11 rows
  FindRow = FindRow + 11
Loop

'Adds graph if it does not allready exist
Exists = 0

For j = 1 To Sheets.Count
  If Sheets(j).Name = NewsheetName Then
     Exists = 1
  End If
Next j

'if graph does not exist, add graph, move and rename else select the sheet
If Exists = 0 Then
  Charts.Add
  ActiveChart.ChartType = xlLine
  ActiveChart.Location Where:=xlLocationAsNewSheet
  SheetName = ActiveChart.Name
  Workbooks(FileName).Activate
  Sheets(SheetName).Select
  Sheets(SheetName).Move After:=Sheets(Sheets.Count)
  Sheets(SheetName).Name = NewsheetName
Else
  Workbooks(FileName).Activate
```

```
  Sheets(NewsheetName).Select
End If

'setting source data according the setting and the count of measurepoints
  If SourceData = "All" Then
    If MeasurePoints = 6 Then
    ActiveChart.SetSourceData Source:=Workbooks(FileName).Sheets(DataName).Range( _
    "D" & FRow & ":D" & NRow - 1 & ",i" & FRow & ":j" & NRow - 1 & ",l" & FRow & _
    ":m" & NRow - 1 & ",o" & FRow & ":p" & NRow - 1 & ",r" & FRow & ":s" & _
    NRow - 1 & ",u" & FRow & ":v" & NRow - 1 & ",x" & FRow & ":y" & NRow - 1), _
    PlotBy:=xlColumns
    ElseIf MeasurePoints = 5 Then
    ActiveChart.SetSourceData Source:=Workbooks(FileName).Sheets(DataName).Range( _
    "D" & FRow & ":D" & NRow - 1 & ",i" & FRow & ":j" & NRow - 1 & ",l" & FRow & _
    ":m" & NRow - 1 & ",o" & FRow & ":p" & NRow - 1 & ",r" & FRow & ":s" & _
    NRow - 1 & ",u" & FRow & ":v" & NRow - 1), PlotBy:=xlColumns
    ElseIf MeasurePoints = 4 Then
    ActiveChart.SetSourceData Source:=Workbooks(FileName).Sheets(DataName).Range( _
    "D" & FRow & ":D" & NRow - 1 & ",i" & FRow & ":j" & NRow - 1 & ",l" & FRow & _
    ":m" & NRow - 1 & ",o" & FRow & ":p" & NRow - 1 & ",r" & FRow & ":s" & _
    NRow - 1), PlotBy:=xlColumns
    ElseIf MeasurePoints = 3 Then
    ActiveChart.SetSourceData Source:=Workbooks(FileName).Sheets(DataName).Range( _
    "D" & FRow & ":D" & NRow - 1 & ",i" & FRow & ":j" & NRow - 1 & ",l" & FRow & _
    ":m" & NRow - 1 & ",o" & FRow & ":p" & NRow - 1), PlotBy:=xlColumns
    ElseIf MeasurePoints = 2 Then
    ActiveChart.SetSourceData Source:=Workbooks(FileName).Sheets(DataName).Range( _
    "D" & FRow & ":D" & NRow - 1 & ",i" & FRow & ":j" & NRow - 1 & ",l" & FRow & _
    ":m" & NRow - 1), PlotBy:=xlColumns
    ElseIf MeasurePoints = 1 Then
    ActiveChart.SetSourceData Source:=Workbooks(FileName).Sheets(DataName).Range( _
    "D" & FRow & ":D" & NRow - 1 & ",i" & FRow & ":j" & NRow - 1), PlotBy:=xlColumns
    End If

    'naming and coloring the data series
    n = 1
    For k = 1 To MeasurePoints * 2
      ActiveChart.SeriesCollection(k).Name = "=""Oletusarvo MP " & n & """"
```

```
      ActiveChart.SeriesCollection(k + 1).Name = "=""Tulos MP " & n & """"
      ActiveChart.SeriesCollection(k).Border.ColorIndex = 2 + n
      ActiveChart.SeriesCollection(k + 1).Border.ColorIndex = 2 + n
      k = k + 1
      n = n + 1
    Next

  'drawing only the data (option for 6 measurepoints only)
  ElseIf SourceData = "DataOnly" Then
    ActiveChart.SetSourceData Source:=Workbooks(FileName).Sheets(DataName).Range( _
    "D" & FRow & ":D" & NRow - 1 & ",J" & FRow & ":J" & NRow - 1 & ",M" & FRow & _
    ":M" & NRow - 1 & ",P" & FRow & ":P" & NRow - 1 & ",S" & FRow & ":S" & _
    NRow - 1 & ",V" & FRow & ":V" & NRow - 1 & ",Y" & FRow & ":Y" & NRow - 1), _
    PlotBy:=xlColumns

    'naming and coloring the series
    For k = 1 To MeasurePoints
      ActiveChart.SeriesCollection(k).Name = "=""Tulos MP " & k & """"
      ActiveChart.SeriesCollection(k).Border.ColorIndex = 2 + n
    Next

  End If

  'adjusting the graph
  With ActiveChart
     .HasTitle = True
     .ChartTitle.Characters.Text = GraphName
     .Axes(xlCategory, xlPrimary).HasTitle = True
     .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = Xname
     .Axes(xlValue, xlPrimary).HasTitle = True
     .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = Yname
 End With
 ActiveChart.ApplyDataLabels Type:=xlDataLabelsShowNone, LegendKey:=False
 ActiveChart.HasDataTable = False

'updating flag and setting the starting row for next graph drawing
FRow = NRow
Found = False
```

```
Next i

'save the workbook
Workbooks(FileName).Save

    'to avoid the handler
    Exit Sub

errorhandler:
    MsgBox ("Virhe: " & Err)
    Exit Sub

End Sub
Sub find_and_store_content(TagToFind As String, row As Integer, col As Integer)

'Macro written 22.8.2005 by Ville Koskiniemi
'Last modified 23.8.2005
'Modified for printing, added comments 23.11.2005, VK
'Finds the tag by name TagToFind, gets the string in it and stores it in Excel-sheet
'cell defined in parameters (row,col)

Dim Content As String   'content between tags

  'etsitään tagi
  Cells.find(What:=TagToFind, After:=ActiveCell, LookIn:=xlFormulas, _
        LookAt:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, _
        MatchCase:=False).Activate

  'puretaan tagit pois
  Content = FindContent(ActiveCell.Value, ">", "<")

  'siirretään sisältö soluun
  Workbooks(FileName).Sheets("Data").Cells(row, col).Value = Content

End Sub

Public Function FindContent(expString As String, stag As String, etag As String) As String
```

```
'Macro written 16.6.2005 by Ville Koskiniemi
'Modified 23.6.2005
'Last modified 23.8.2005
'Modified for printing, added comments 23.11.2005, VK
'Gets the content between tags.

   Dim starter As Integer 'starting mark of the content string which is returned
   Dim stopper As Integer 'stopping mark of the content string which is returned
   Dim expLength, i As Integer
   Dim retString, ch As String

   'finds starting mark
   expLength = Len(expString)
    retString = ""
    For i = 1 To expLength
     ch = Mid(expString, i, 1)
     If ch = stag Then
       starter = i
       i = expLength
     End If
    Next i

    'finds ending mark
    For i = starter To expLength
      If expString <> "" Then ch = Mid(expString, i, 1)
      If ch = etag Then stopper = i
    Next i

    If expString <> "" Then
      FindContent = Mid(expString, starter + 1, stopper - starter - 1)
    Else
      'set this as you want
      FindContent = "empty"
    End If

End Function

Sub get_settings()
```

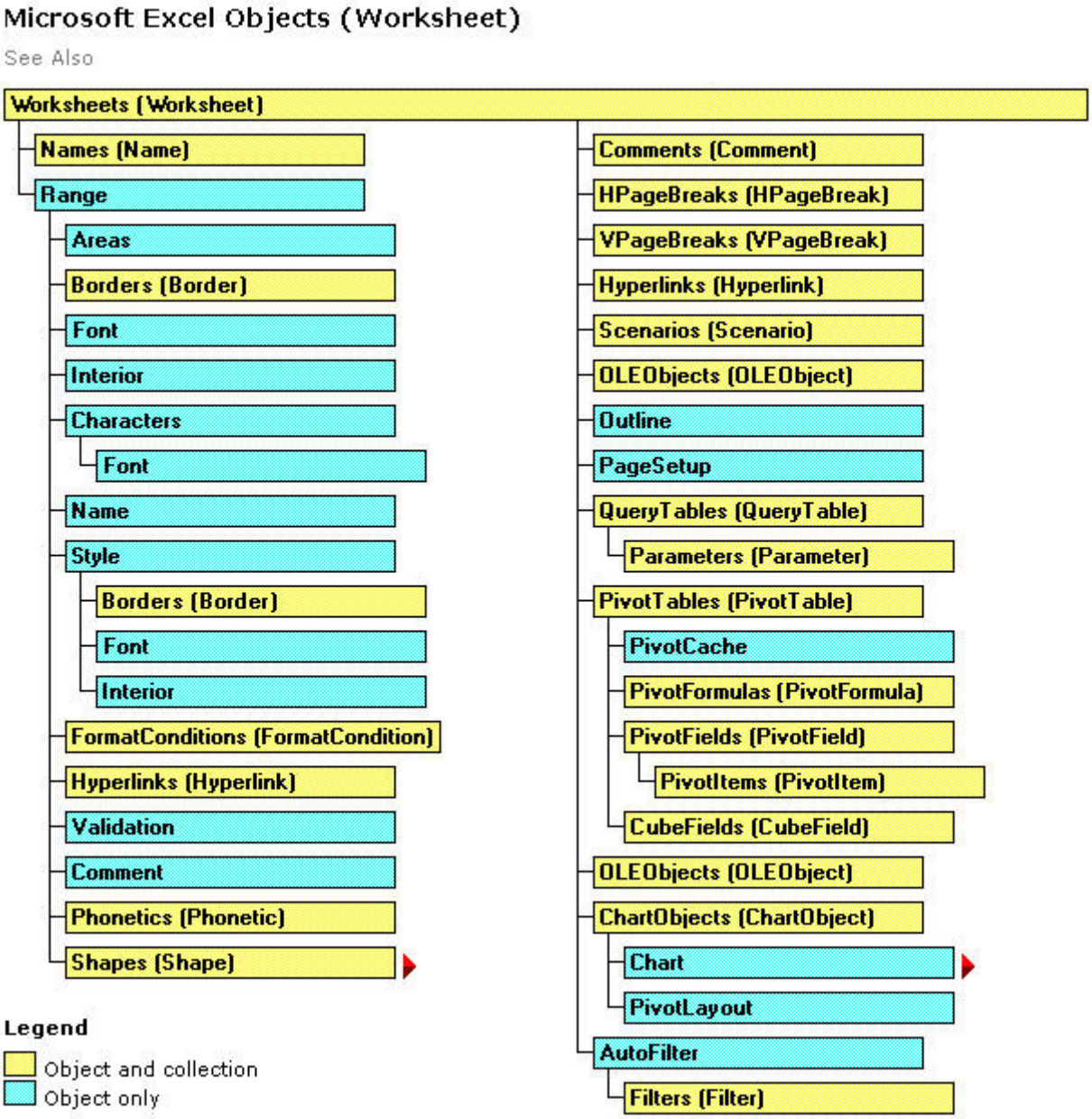
```
'Macro written 23.8.2005 by Ville Koskiniemi
'Modified 23.8.2005
'Last modified 23.8.2005
'Modified for printing, added comments 23.11.2005, VK
'Gets the count of targets and measurepoints from the XML-file, sets
'them automatically to "Asetukset"

Dim TagToFind As String
Dim Content As String   'content between tags
Dim BeginningRow, FindRow, MemRow As Integer
Dim counter As Integer
Dim row, col, nextRow As Integer
Dim flag, endLoop As Boolean
Dim i As Integer

  'setting the variables
  flag = False
  endLoop = False
  counter = 0
  BeginningRow = 5
  FindRow = 6

break:


  Do Until endLoop = True
    'counter for positions
    counter = counter + 1

    'etsitään tagi
      Cells.find(What:="kohdenimi", After:=ActiveCell, LookIn:=xlFormulas, _
        LookAt:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, _
        MatchCase:=False).Activate

      'ensimmäisellä kierroksella false, sen jälkeen true
      If flag = True Then
        FindRow = ActiveCell.row
```

```
End If

'ensimmäisellä kierroksella false, sen jälkeen true
If flag = False Then
  BeginningRow = ActiveCell.row
  flag = True
End If

'jos ollaan hypätty takaisin tiedoston alkuun, niin endloop on tosi
'(näin käy kun find ei enää löydä uutta etsittävää tiedoston lopusta
' - ainut tilanne, missä findrow pienenee on tämä)
If BeginningRow = FindRow Then
  endLoop = True
  GoTo break
End If

'puretaan tagit pois
Content = FindContent(ActiveCell.Value, ">", "<")

'juokseva numerointi
Workbooks(FileName).Sheets("Asetukset").Cells(counter + 4, 9).Value = counter
'siirretään sisältö soluun
Workbooks(FileName).Sheets("Asetukset").Cells(counter + 4, 10).Value = Content
'laitetaan muistiin rivinumero
Workbooks(FileName).Sheets("Asetukset").Cells(counter + 4, 12).Value = ActiveCell.row
'graafityyppi - oletuksena 1
Workbooks(FileName).Sheets("Asetukset").Cells(counter + 4, 13).Value = 1

'etsitään tagi
Cells.find(What:="kohdeId", After:=ActiveCell, LookIn:=xlFormulas, _
  LookAt:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, _
  MatchCase:=False).Activate

'puretaan tagit pois
Content = FindContent(ActiveCell.Value, ">", "<")

'siirretään sisältö soluun
Workbooks(FileName).Sheets("Asetukset").Cells(counter + 4, 11).Value = Content
```

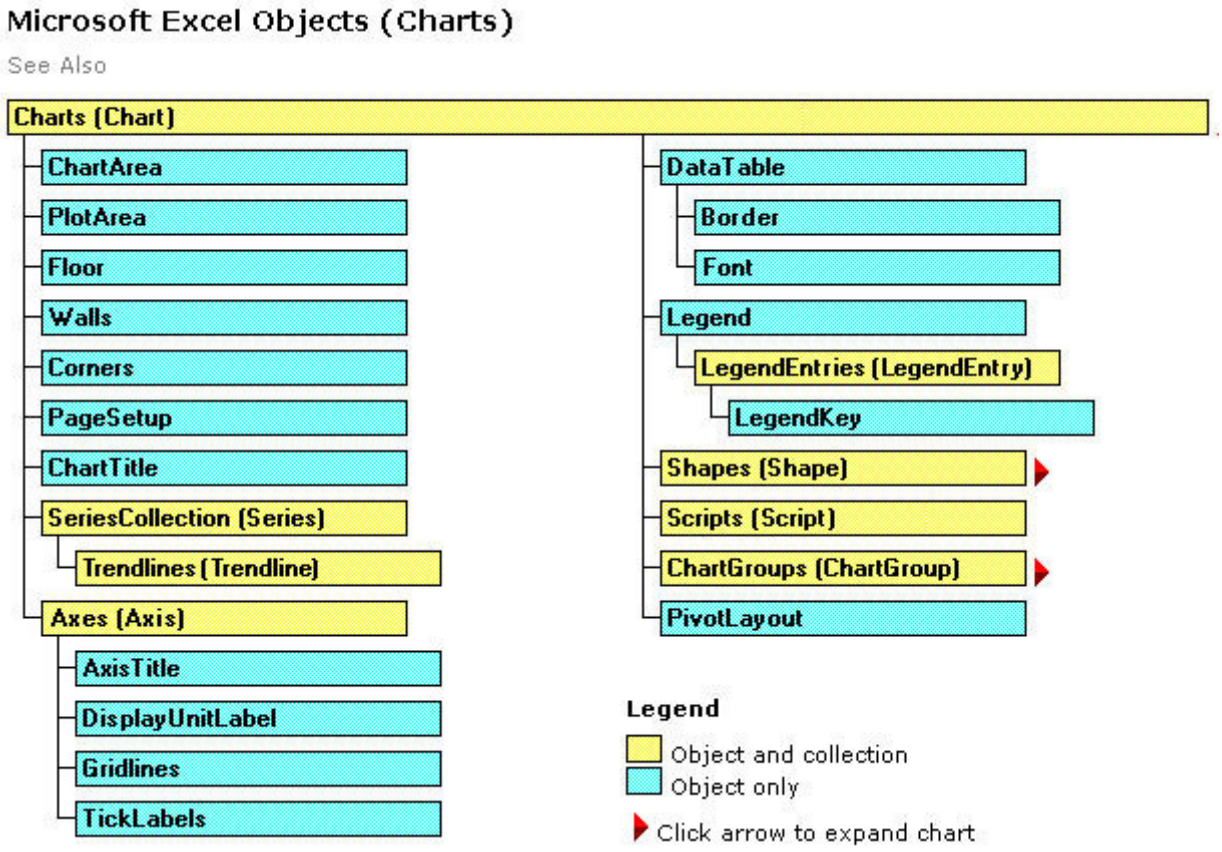
```
Loop

'etsitään viimeinen tagi (jos menee vikaan, niin tässä on pieni vaara!)
Cells.find(What:="tyomaarain", After:=ActiveCell, LookIn:=xlFormulas, _
  LookAt:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, _
  MatchCase:=False).Activate

'siirretään rivinumero soluun muistiin
Workbooks(FileName).Sheets("Asetukset").Cells(counter + 4, 12).Value = ActiveCell.row

'nollataan muuttujat seuraavaa silmukkaa varten
flag = False
endLoop = False
FindRow = 0

'for every position
For i = 1 To Workbooks(FileName).Sheets("Asetukset").Range("ASXML_GraafiLKM").Value

  'nollataan laskuri jokaiselle positiolle - lasketaan siis mittapisteitä!
  counter = 0
  'haetaan muistista ensimmäisen ja sitä seuraavan position rivinumerot
  row = Workbooks(FileName).Sheets("Asetukset").Range("L" & i + 4).Value
  nextRow = Workbooks(FileName).Sheets("Asetukset").Range("L" & i + 5).Value

  'jos ollaan viimeisellä kierroksella, poistetaan tiedoston viimeisen rivin
  'muistavan solun sisältö
  If i = Workbooks(FileName).Sheets("Asetukset").Range("ASXML_GraafiLKM").Value Then
    Workbooks(FileName).Sheets("Asetukset").Cells(i + 5, 12).Value = ""
  End If

  'select row of the current position
  Cells(row, 1).Activate

  'while between positions
  Do While row < nextRow

    'ei suoriteta ensimmäisen position kohdalla - muuten
    'suoritus pysähtyy heti ensimmäiseen mittapisteeseen
```

```
        If flag = True And i > 1 Then
          FindRow = ActiveCell.row
        End If

        'suoritetaan ensimmäisellä silmukkakierroksella
        If flag = False Then
          BeginningRow = Workbooks(FileName).Sheets("Asetukset").Range("L5").Value + 3
          flag = True
        End If

        'kun ollaan hypätty takaisin tiedoston alkuun -> breakout
        If BeginningRow = FindRow Then
          endLoop = True
          GoTo breakout
        End If


        'etsitään tagi (jos menee vikaan, niin tässä on pieni vaara!)
        Cells.find(What:="mittapiste", After:=ActiveCell, LookIn:=xlFormulas, _
          LookAt:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, _
          MatchCase:=False).Activate
        'countig measurepoints
        counter = counter + 1
        row = ActiveCell.row
      Loop

breakout:

      'siirretään määrä soluun
      Workbooks(FileName).Sheets("Asetukset").Range("L" & i + 4).Value = counter - 1

    Next
End Sub
```

Liite 3. Microsoft Excelin Worksheets- ja Charts-objektiluokkien rakenne



Kuva 1. Worksheet-objektin rakenne.



Kuva 2. Charts-objektin rakenne.