Ismail Belmostefa

PowerShell Core ja sitä edeltävät komentorivi-pohjaiset hallintatyökalut

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

12.12.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Ismail Belmostefa PowerShell Core ja sitä edeltävät komentorivipohjaiset hallintatyökalut 38 sivua + 2 liitettä 12.12.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Ohjaaja Kari Sundberg Yliopettaja Markku Nuutinen

Insinöörityön aiheena oli PowerShell Core ja sitä edeltävät komentorivipohjaiset hallintatyökalut. Tavoite oli ymmärtää näiden asennusympäristö, alla käytetyt teknologiat ja niihin liittyvä terminologia. Tutkimustyö syntyi harjoittelutyön muistiinpanojen lopputuloksena sekä tutkimalla alan kirjallisuutta ja verkkomateriaalia.

Insinöörityössä selvitettiin PowerShell Coren ja sitä edeltävien komentorivipohjaisten hallintatyökalujen alla käytetyt teknologiat, terminologia, käyttökohteet ja motivaatio niiden syntyyn. Koska komentorivipohjaiset hallintatyökalut on rakennettu asennusympäristön tarjoamien palveluiden päälle, palveluiden toiminnan hahmottaminen edesauttaa komentorivin käyttöä ja soveltamista. Tutkimustyötä on hyödynnetty tietokoneen ylläpidossa, ohjelmoinnissa ja peruskäytössä. Tutkimustyö osoitti asennusympäristön kokonaiskuvan hallitsemisen tärkeyden komentorivipohjaisten työkalujen käytössä.

Avainsanat	DDE, OLE, COM, .NET, CMD.EXE, COMMAND.COM, WSH, MOM, PowerShell

Author Title Number of Pages Date	Ismail Belmostefa Microsoft Windows command line based management tools 38 pages + 2 appendices 12 December 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Kari Sundberg, Manager Markku Nuutinen, Principal Lecturer

This study was about PowerShell Core and the preceding command line interface management tools. The goal of this study was to understand the installation environment, the underlying technology, and the associated terminology. This study was accomplished with the help of notes taken during an internship, as well as by studying literature and internet sources on the subject.

In this study, PowerShell Core, the preceding command line interface management tools, the technologies, the terminology, their use cases, and the motivation behind their creation were analyzed. Because these command line interface based management tools are built on services provided by an installation environment, understanding the operation of these services helps to use and utilize these command lines. This study has been used in computer maintenance and programming. This study showed that a big part of using the command line based tools is understanding and managing how the installation environment works.

Keywords	DDE, OLE, COM, .NET, CMD.EXE, COMMAND.COM, WSH, MOM, PowerShell

Sisällys

Lyhenteet

BCL	*Base Class Library*. Perusluokkakirjasto, kaikille .NET-kielille yhteinen.
CLR	*Common Language Runtime*. .NET-ajoympäristö, joka hallitsee suoritettavia .NET-ohjelmia.
COM	*Component Object Model*. Binäärirajapintastandardi.
COR	*Common Object Runtime.* Yksi nimistä, jotka annettiin .NET:lle ennen sen lopullista nimeä.
DDE	*Dynamic Data Exchange*. IPC-metodi Microsoftin käyttöjärjestelmässä.
DSC	*Desired State Configuration*. Hallinta-alusta PowerShellissa.
ECMA	*European Computer Manufacturers Association.* Järjestö, joka helpottaa laaja-alaisesti tieto- ja viestintäteknologian sekä kulutuselektroniikan standardien luontia.
FCL	*Framework Class Library*. Kehystyökaluluokkakirjasto, joka sisältää kaikki muut luokat BCL:n lisäksi .NET Frameworkin luokkakirjastosta.
IPC	*Inter-process Communication*. Prosessien välinen kommunikaatiomekanismi käyttöjärjestelmässä.
ISE	*Integrated Scripting Environment*. Integroitu komentosarjaympäristö.
MFC	*Microsoft Foundation Class Library*. Microsoftin C++ -kirjasto, joka helpottaa COM-teknologiaan perustuvien ohjelmien kehitystä.
MOM	*Microsoft Operation Management*. Microsoftin hallintaratkaisu.
MS-DOS	*Microsoft Disc Operating System*. Tekstipohjainen komentorivi-käyttöjärjestelmä.

NGWS *Next Generation Windows Services.* Yksi nimistä, jotka annettiin .NET:lle ennen sen lopullista nimeä.

OLE *Object Linking and Embedding*. Microsoftin tekniikka, joka mahdollistaa olioiden sulauttamisen ja linkittämisen dokumenttiin tai toiseen olioon.

OMS *Operations Management Suite*. Microsoftin IT-hallintaratkaisu.

PCL *Portable Class Library*. Kannettava luokkakirjasto. Kokoonpano, joka toimii monella eri .NET Framework -alustalla.

RPC *Remote Procedure Call*. Etäproseduurikutsu.

UDT *Uniform Data Transfer*. Antaa COM:lle standarditavan siirtää dataa kahden ohjelman välillä.

VBA *Visual Basic for Applications*. Microsoftin kehittämä komentosarja-ohjelmointikieli.

VDM *Virtual DOS machine*. Virtuaalinen DOS-tietokone.

VTBL *Virtual Table*. Virtuaalitaulu, joka on taulu funktio-osoittimia virtuaalimetodeihin C++:ssa.

WMF *Windows Management Framework*. Microsoftin ajuri, joka sisältää päivityksiä ja ohjelmia Windowsille ja Windows Serverille hallintarajapintoihin.

WSF *Windows Scripting File*. Tiedosto, jossa voi käyttää useampaa eri kieltä samassa tiedostossa asentamalla eri skriptikoneita.

1 Johdanto

Päätavoite tässä insinöörityössä on saada ote hallintatyökalujen taustalla käytetyistä teknologioista ja niiden ominaisuuksista. Tämä tieto mahdollistaa asiaan sopivien teknologioiden ja työkalujen valinnan.

Selvitän insinöörityössä PowerShell Core -komentorivin ja sitä edeltävien komentorivipohjaisten hallintatyökalujen terminologiaa, ympäristöä ja taustalla käytettyä teknologiaa, motivaation niiden kehitykseen sekä niiden hyötykohteet. PowerShell Core -komentoriviä edeltävien komentorivipohjaisten hallintatyökalujen taustalla käytettyjä teknologioita on hyödynnetty tekstinkäsittelyohjelmista tietokantaohjelmiin niissä ympäristöissä, joihin PowerShell Corea edeltävät komentorivipohjaiset hallintatyökalut on asennettu. Tämän vuoksi tieto on hyödyllistä sekä ohjelmoijille että näiden ohjelmien käyttäjille.

Insinöörityön liitteenä ovat laatimani käsitekartat, jotka sisältävät insinöörityössä käsittelemäni asiat. Käsitekartta toimii runkona insinöörityötä lukiessa. Sen avulla hahmottaa myös eri osien väliset yhteydet.

2 Ei-.NET-pohjaiset komentorivit

2.1 COMMAND.COM ja CMD.EXE

Microsoft Disk Operating System (MS-DOS) on tekstipohjainen komentorivikäyttöjärjestelmä. Se on periytynyt IBM:n tietokoneisiin kehitetystä 86-DOS:sta. Microsoft esitteli MS-DOS:n vuonna 1981. Sen viimeinen päivitys oli vuonna 1994 MS-DOS -julkaisulla 6.22. MS-DOS antoi käyttäjän navigoida, avata ja muokata tiedostoja komentoriviltä graafisen käyttöliittymän sijaan. [1.]

MS-DOS command prompt on 16-bittinen DOS-ohjelma, joka sijaitsee COMMAND.COM:ssa suoritettavissa tiedostossa. Command tai COMMAND.COM on komentorivitulkki, jota tarvitaan Microsoftin käyttöjärjestelmän käynnistämiseen. Ajettaessa Windows NT:tä uudempaa Windows-sarjan käyttöjärjestelmää komentotulkista näkyy kaksi versiota: COMMAND.COM ja CMD.EXE. COMMAND.COM on kehitetty MS-DOS:n kanssa yhteensopivaksi. Windows NT ajaa COMMAND.COM:a Windows NT:n virtuaalisessa DOS-koneessa (VDM). Windows NT command shell on 32-bittinen Windows NT –konsoliohjelma, joka sijaitsee CMD.EXE-suoritettavassa tiedostossa. CMD.EXE tarjoaa enemmän ympäristömuuttujia kuin COMMAND.COM. CMD.EXE ja COMMAND.COM käyttävät samaa kuvaketta. [1; 2; 3; 4.]

COMMAND.COM ja CMD.EXE voidaan korvata kolmannen osapuolen työkaluilla. Norton Utility -työkalu sisältää versiossa 8.0 NDOS:n. Se on paranneltu COMMAND.COM, joka sisältää muun muassa .BAT-tiedoston sisääntulon ja vaihtaa levyn ja kansion yhdellä komennolla. [5.]

CMD.EXE:n voi korvata Take Commandilla. Se sisältää parannuksina interaktiivisen käyttöliittymän ja Windows-komentorivityökalun. Take Command tuo NDOS:n tavoin lisää komentoja ja ominaisuuksia. [6.]

2.2 Active Scripting ja sen isännät

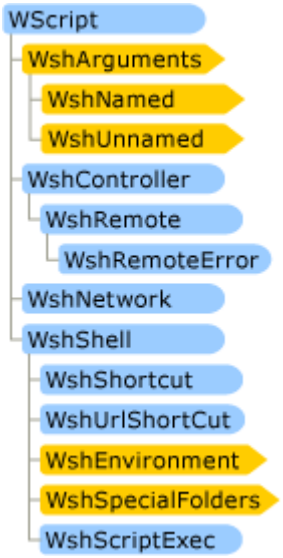
Microsoft esitteli 1990-luvulla Active Scripting -spesifikaation ja määritteli kokoelman COM-rajapintoja, jotka mahdollistivat näiden COM-rajapintojen implementoimien komentosarjakielien olevan isännöityjä missä tahansa hyväksytyssä isännässä. [7.]

Active Scripting -spesifikaatio määrittää kieltä prosessoivan koneen. Active Scripting -isäntä käyttää konetta komentosarjakielen tulkitsemiseen. Esimerkiksi vbscript.dll ja jscript.dll ovat Active Scripting -koneita, jotka tuodaan Windowsissa system32-kansioon. [7.]

Active scripting -isäntiä ovat muun muassa IIS-verkkopalvelin ja WSH, jotka tulevat Windowsin mukana. WSH käsittää kaksi isäntäohjelmaa: Wscriptin, joka on interaktiivinen tila, ja Cscriptin, joka on BAT-tiedostojen isäntä Active Scriptingille. Active Scripting -koneita saa myös kolmansilta osapuolilta esimerkiksi Perliin ja Pythoniin. [8; 9; 10.]

2.3 WSH-oliot

Kuten kuvassa 1 näkyy, WSH koostuu hierarkkiseen järjestykseen asetetuista 14 oliosta, jossa Wscript-olio on juurena. Olioiden COM-rajapinnat voidaan jakaa kahteen kategoriaan: komentosarjojen ajon- ja ongelmanhallintaan tarkoitettuihin olioihin sekä apufunktioina toimiviin olioihin. Komentosarjojen ajon- ja ongelmanhallintaan tarkoitetut oliot sisältävät kokoelman rajapintoja, jotka mahdollistavat WSH-muokkauksia, kuten COM-funktioiden, CreateObject- ja GetObject-funktioiden suorittamisen sekä viestien ulostulon näyttöön. Apufunktiot ovat ominaisuuksia ja metodeja, jotka suorittavat tapahtumia, kuten verkkolevykartoitus ja rekisteriavainten muokkaus. Apufunktioita voidaan käyttää yksinkertaisiin kirjautumiskomentosarjoihin. [11.]



Kuva 1. WSH-oliomallin hierarkia [11].

Microsoft Operation Management (MOM) tarjoaa tapahtumaohjatun operaation monitoroinnin, suorituksen seurannan sekä turvallisuus-, auditointi- ja komentosarjamahdollisuuden. MOM-komentosarjat voidaan kirjoittaa millä tahansa ActiveX-kielellä. [12.]

Myös MOM on Active Scripting -isäntä kuten WSH, mutta se määrittää WSH:sta poikkeavat ympäristömuuttujat. MOM sisältää Wscriptin sijaan ScriptContextin. Molemmat isännät voivat käyttää toistensa ympäristössä luotuja olioita. [13.]

2.4 WSF-tiedosto

Kuten esimerkkikoodissa 1 näkyy, käyttäjät voivat asentaa eri komentosarjakoneita ja käyttää niitä samassa tiedostossa, jota kutsutaan nimellä Windows Scripting File (WSF). WSF-tiedoston pääte on .wsf. WSF on xml-tiedosto, johon komentosarjoja kirjoitetaan tietyllä kielellä script-elementtien sisälle, jossa kieli määritetään language-attribuutin sisälle. WSF-tiedoston sisällä ulkoiseen komentosarjatiedostoon voi viitata src-attribuutilla. WSF on hyödyllinen esimerkiksi silloin, kun tarvitaan jokin ominaisuus, jota toinen ohjelmointikieli ei tue. [14.]

```
<script language="JScript" src="myNewUtility.JS"/>
<script language=”VBScript” src="myUtility.vbs">
<script language=”VBScript”>
  WScript.Echo "hei"
</script>
```

Esimerkkikoodi 1. Ote WSF-koodista.

WSH on sisäänrakennettu Microsoft Windows -käyttöjärjestelmiin Windows 98 -versiosta lähtien. WSH-version näkee kirjoittamalla Cscriptin komentoriville. [15.]

3 Compound Document -dokumenttityyppi ennen COM-teknologiaa

3.1 Windows-käyttöjärjestelmän prosessien välinen kommunikaatio

Windows-käyttöjärjestelmän mekanismit helpottavat ohjelmien välistä kommunikaatiota ja datan vaihtoa. Prosessien välinen kommunikaatio (Inter Process Communication, IPC) voi tapahtua ohjelmien välillä yhden koneen sisällä tai samassa verkossa olevien tietokoneiden ohjelmien välillä. IPC:ksi luokiteltavat ohjelmat voivat toimia asiakkaana, palvelimena tai kumpanakin. [16.]

Windows tarjoaa nykyisin seuraavia IPC-metodeja: DDE, COM, RPC, Windows Sockets, Clipboard, Data Copy, File Mapping, Mailslots ja Pipes. [16.]

Seuraavaksi tarkastellaan OLE-teknologiaa, joka hyödyntää pohjanaan DDE- ja COM-teknologiaa.

3.2 DDE-teknologia

Dynamic Data Exchange (DDE) suorittaa metodeja ja jakaa tietoa ohjelmien välillä. Kommunikaatio DDE:ssä tapahtuu kolmen argumentin mallilla: ohjelma, aihe ja tavaran nimi. DDE kirjaa ohjelmat niiden nimen perusteella. Jokainen ohjelma omaa aihepiirin ja siinä kappaleen. Jos esimerkiksi tarvittaisiin Excel-tiedoston tietyn alkion tieto, nähtäisiin DDE:ssä ohjelman nimi "Excel", aihepiirinä Excel-tiedosto "ExcelWorkbook.xls" ja kappaleena alkio "r1c2". [17.]

DDE-ohjelma tallentaa topic- ja item-merkkijonot Atom-tauluun jakaakseen ne muiden ohjelmien kanssa. DDE-datan tallennuksen rajat on näin määritelty Atom-taulun rajojen mukaan. DDEML on kirjasto, joka helpottaa DDE-ohjelmien merkkijonojen hallintaa ja jakamista. DDE:tä käytetään vielä nykyisin yksinkertaisissa prosessien välisissä kommunikaatioissa, kuten leikepöytä- sekä leikkaa ja liitä -ohjelmissa ja Access-tietokannassa. [18; 19; 20.]

3.3 OLE-kehystyökalu ja Compound Document -tiedostotyyppi

Object Linking and Embedding (OLE) on Microsoftin ensimmäinen oliopohjainen kehystyökalu. Microsoft esitteli OLE 1.0 -spesifikaation vuonna 1991. [21, s. 706.]

Compound document on dokumentti, joka sisältää tekstin lisäksi myös kuvia, taulukoita, videoita, ääniä tai muita multimediaominaisuuksia. Compound document -teknologioita ovat esimerkiksi XML, XSL, ActiveX documents ja OLE. Compound-dokumentteja luodaan Object linking and embedding (OLE) -teknologialla, jotka sisältävät tietoja monesta eri lähteestä. [21, s. 706.]

OLE:n avulla olioita voidaan sulauttaa compound-dokumenttiin, joka avautuu sille määritetyllä ohjelmalla. Esimerkiksi OLE:n mahdollistavassa tekstinkäsittelyohjelmassa dokumenttiin voidaan sulauttaa laskentataulukko-olio. Perinteisissä leikkaa ja liitä -metodeissa vastaanottava ohjelma muuttaa liitetyn datan tietoa. Sulautettu dokumentti sen sijaan säilyttää kaikki alkuperäiset ominaisuudet. Jos käyttäjä päättää muokata sulautettua dataa, Windows aktivoi alkuperäisen ohjelman ja lataa sulautetun dokumentin. Esimerkiksi Excel-laskentataulu-oliota klikattaessa Windows aktivoisi laskentataulukko-ohjelman. OLE voi pitää aktiivisen linkin kahden dokumentin välillä tai sulauttaa dokumentin toisen sisään, kun taas DDE voi vain siirtää dataa kahden ohjelman välillä. [21, s. 706; 22; 23; 24.]

Teen seuraavaksi yleiskatsauksen OLE-kirjastoihin ja kerron VTBL:n ja Windows-leikepöydän käytöstä OLE 1.0 -ohjelmassa.

Olecli.dll

Olecli.dll on Microsoft Windowsin OLE-asiakaskirjasto. Kaksisuuntainen kommunikaatio tapahtuu asiakasohjelmiston ja OLE-kirjaston välillä. Asiakas kutsuu asiakaskirjastosta funktioita suorittaakseen OLE-tehtäviä, kuten olion luonti, renderöinti, lataus ja tallennus. Asiakaskirjasto lähettää tilatietoja ohjelmistolle kutsumalla ohjelmiston määrittelemät takaisinkutsufunktiot. Asiakaskirjasto asettaa asiakkaan takaisinkutsufunktioon tilatiedon, joka sisältää huomautuksen siitä, että olio on muuttunut, uudelleennimetty tai tallennettu tai OLE-palvelinohjelma on sulkenut sen. [25.]

Olesvr.dll

OLESVR.DLL on OLE-palvelinohjelman palvelinkirjasto. Kaksisuuntainen kommunikaatio tapahtuu palvelinohjelmiston ja OLE-palvelinkirjaston välillä. Palvelinkirjasto vie tietoa palvelinohjelman 27 ohjelman määrittelemän takaisinkutsufunktion kautta, joilla se voi esimerkiksi rekisteröidä tai poistaa saatavuutensa tai ilmoittaa tallentaneensa tai uudelleennimenneensä tiedoston. [25.]

Object handler

Object handler -palvelinohjelma on tyypillisesti pieni ja latautuu tehokkaammin kuin täysin funktionaalinen palvelinohjelma. Object handler luodaan käyttämällä olesvr.dll-palvelinkirjastoa, samoin kuin täysin funktionaalinen palvelinohjelma. Object handler on dynaamisesti linkattu kirjasto, joka sisältää vain palvelinohjelman olioiden luokkien tukemiseen tarvittavat funktiot. Esimerkiksi kun asiakasohjelma kutsuu olion verbiä, kutsu voidaan prosessoida object handlerilla, joka latautuu muistiin, prosessoi kutsun ja poistuu ilman pääpalvelinohjelman apua. [25.]

Virtual table

Ohjelma kutsuu alustuksen aikana asiakas- tai palvelinohjelman sopivat kirjastot pointterilla VTBL data -struktuuriin. Kirjasto käyttää VTBL:ssä osoitinta ohjelman takaisinkutsufunktioiden kutsumiseen. VTBL antaa OLE-kirjastojen kutsua ohjelman määrittelemät funktiot, jotka toteuttavat metodin riippumatta funktion nimestä. OLE-kirjasto määrittelee VTBL:n avulla, mitkä metodit asiakkaan tai palvelimen on toteutettava, määrittelemättä miten ne on toteutettava. [25.]

Kirjastojen välinen kommunikaatio

Microsoft Windows -ohjelman kehitystyökalun versiossa 3.1 OLE-kirjastojen alkuperäisessä implementaatiossa kirjastot kommunikoivat keskenään DDE-viestitysprotokollalla. Kirjastot ovat piilottaneet kirjastojen välillä käytetyn protokollan, eivätkä ne vaikuta OLE-asiakkaan tai palvelimen suunnitteluun. [25.]

Shell.dll

Shell.dll tarjoaa OLE-ohjelmille API-funktioita, joilla ohjelma voi lukea ja muuttaa Windows-rekisteritietokantaa. Rekisteritietokanta sisältää tiedot järjestelmään asennetuista OLE-palvelimista sekä niiden tukemista olioluokista ja verbeistä. Shell.dll tarjoaa myös raahaa ja pudota -toiminnon, jolla tiedosto raahataan ja pudotetaan tiedostoselaimella asiakasohjelman kohdedokumenttiin ja sulautetaan tiedoston dokumenttiin. [25.]

Olion asettaminen leikepöytään

Kun palvelinohjelma asettaa olion leikepöydälle, se täydentää sen yhdellä tai useammalla graafisella olioesityksellä. Nämä esitykset sisällytetään metatiedostoformaattiin (CF_METAFILEPICT), laiteriippuvaiseen bitmap-formaattiin (CF_BITMAP) tai laiteriippumattomaan bitmap-formaattiin (CF_DIB). Kun käyttäjä liittää olion asiakasohjelman säiliödokumenttiin, OLE-kirjastot käyttävät yhtä näistä esitysformaateista olion esittämiseen. Esitysformaatti esittää olion esiintymisen oliokohtaisesti. Olion esityksessä käytettävästä esitysformaatista riippuen sen ulkonäkö muuttuu, kun bitmapin tai metatiedoston kokoa muutetaan. [25.]

Sulautettu olio tallennetaan leikepöydälle OwnerLink-formaatilla, Native-formaatilla ja yhdellä kolmesta esitysformaatista (CF_METAFILEPICT, CF_BITMAP tai CF_DIB). Ohjelmien kehityspaketin mukana tulevassa Windows.h-otsikkotiedostossa jokainen esitys ja esimääritelty Windows-formaatti on määritelty nimettynä vakiona. Kuitenkaan kaikkia OLE-ohjelmien käytössä olevia OwnerLink- ja Native-formaatteja ei ole määritelty Windows.h-tiedostossa. Jokaisen OLE-ohjelman on kutsuttava RegisterClipboardFormat-funktio rekisteröidäkseen OwnerLink- ja Native-formaatit Windowsissa. [25.]

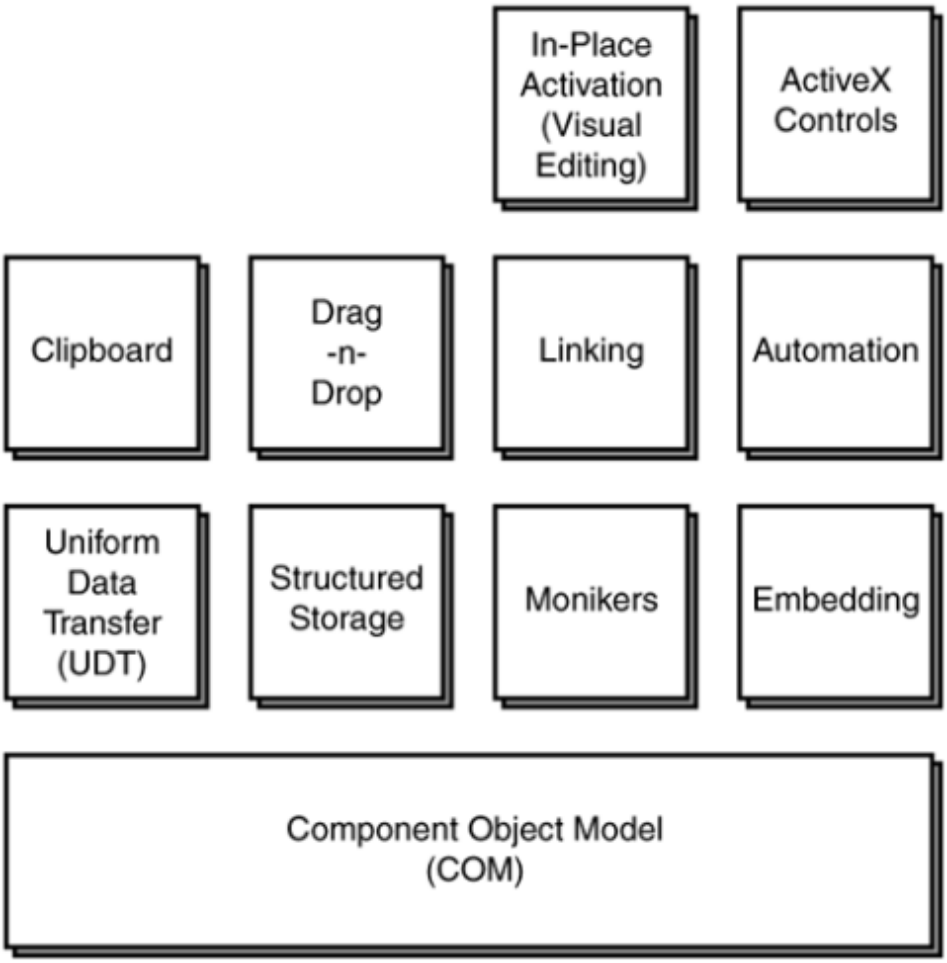
3.4 Ei-OLE-komponentit

Microsoft sisällytti ei-OLE-komponentteja varten käyttöjärjestelmiinsä Windows-käyttöjärjestelmän versiosta 3.1 alkaen XP-versioon asti The Object Packager-ohjelman, jolla voidaan pakata ja sisällyttää OLE-asiakkaaseen ei-OLE-komponentteja. [26.]

4 OLE 2.0- ja ActiveX-teknologiat

Kerron nyt OLE 2.0:sta, joka on OLE 1.0:n jälkeläinen, sekä ActiveX-teknologiasta, joka on nimi latausta helpottaville verkkotekniikoille. Tämän jälkeen syvennyn komponenttikeskeiseen teknologiaan. [21, s. 706-707.]

OLE 2.0 on rakennettu COM:n päälle. Kuten kuvassa 2 näkyy, OLE 2.0:n tarjoamia palveluita ovat OLE clipboard extensions, Drag and Drop, Linking, Monikers, In-Place Activation, Automation, ActiveX Controls, OLE and Active Documents, Embedding, Uniform Data Transfer (UDT) ja Structured Storage. [21, s. 706-708.]



Kuva 2. OLE-teknologia ja sen tarjoamat palvelut [21, s. 706-708].

4.1 OLE:n nimi

OLE (Object Linking and Embedding) ei ole enää linkkauksen ja sulauttamisen akronyymi. Microsoft kuvaa OLE-termillä nykyisin OLE 2.0 -teknologiaa. OLE on laajennettava arkkitehtuuri, jota voidaan parantaa ja laajentaa muuttamatta sen perusteita. OLE 2.0 -julkaisuun lisättiin OLE-kontrollit. OLE 1.0:lla luodut oliot toimivat edelleen ja ovat vuorovaikutuksessa OLE-ohjelmien kanssa. Niiden toiminnallisuudet on kuitenkin rajoitettu alkuperäiseen OLE 1.0 -spesifikaatioon, joten niissä ei ole versiointitarpeita. [21, s. 708.]

4.2 MFC-kirjasto ja komponenttiteknologia

Microsoft Foundation Class (MFC) on Microsoftin luoma C++-kirjasto, jolla on helpotettu COM-teknologiaan perustuvien ohjelmien luomista. MFC-ohjelmoinnissa hyödynnetään paljon Visual Studioon integroidun ohjelmointiympäristön ohjattuja toimintoja eli wizardeja. [21, s. 714.]

4.3 Käyttäjän vuorovaikutus COM-tekniikan kanssa

Ensimmäinen näkyvyys: Käyttäjän malli

Ensimmäinen OLE-dokumentin ominaisuus on yleinen käyttäjämalli, jossa OLE-dokumenttien käsittelyyn vaadittavat käyttöliittymäominaisuudet (User Interface) pysyvät samoina ohjelmistoista toiseen. [21, s. 710.]

Seuraava näkyvyys: Automation

Seuraavan tason näkyvyys on Automation, jolla käyttäjä voi muokata ohjelmia olioina niiden ominaisuuksien ja metodien kautta käyttämällä korkean tason ohjelmointikieltä, kuten VBA, VBScript tai Javascript. Näitä oliota voidaan muokata, ja ne voidaan asettaa toimimaan vuorovaikutuksessa keskenään toimenpiteiden suorittamiseksi. Käyttäjä voi esimerkiksi luoda Microsoft Word -tiedoston, joka päivittää Excel-laskentataulukkoa tai Access-tietokantaa. [21, s. 711.]

Kolmas näkyvyys: ActiveX-kontrollit

ActiveX-kontrollit ovat itsenäisiä, uudelleenkäytettäviä komponentteja, jotka voidaan sulauttaa ohjelmiin. ActiveX-kontrollit paljastavat muokattavat ominaisuudet ja metodit. Ominaisuus voi olla päivämäärä, arvo tai taustan väri. Metodi voi olla funktio, joka muuttaa päivämääräarvoa tai taustaväriä. Käyttäjä näkee vain kontrollin, joka ottaa sisääntulon ja vie sen sen sisältämälle ohjelmalle. ActiveX-kontrollin metodeilla voidaan suorittaa tietynlaisia operaatioita. ActiveX-kontrolli voi Automation-ominaisuuksien lisäksi alustaa tapahtumia, jotka on lähetetty kontrollerin säiliöön. [21, s. 712.]

5 Component Object Model -järjestelmä

5.1 Tarkastelun kohteet

Tietokoneen historia vaikutti Component Object Model -järjestelmän (COM) syntyyn. Jotta COM-järjestelmän toimintaa olisi helpompi ymmärtää, käyn lyhyesti läpi hajautettujen tietokoneiden kehityksen alusta alkaen selvittääkseni COM:n motiivin. [27. luku 1.]

5.2 Tyhmät terminaalit

Yritykset vuokrasivat 1950- ja 1960-luvulla keskustietokoneita. Keskustietokoneita oli vähän, ja niitä käytti yksi henkilö kerrallaan. Käyttäjien määrän lisäämistä varten otettiin käyttöön niin sanotut tyhmät terminaalit. Ne toimivat suoran linkin kautta ikkunana keskustietokoneessa suoritettavaan prosessiin, eikä niillä voinut vaikuttaa prosessiin. PC:n tulon myötä tyhmät terminaalit emuloitiin PC:hen, jolloin keskustietokoneeseen voitiin kirjautua etäisesti ja tarkastella prosessia aivan kuten ei-emuloidulla tyhmällä terminaalilla. Emuloidun terminaalin kommunikaatioprotokollat olivat monimutkaisempia kuin ei-emuloidun tyhmän terminaalin, koska emuloidussa tyhmässä terminaalissa käyttäjällä oli oikeus valita haluamansa kommunikaatioprotokolla. [27. luku 1.]

5.3 Tyhmät terminaalit: Two-Tier tai asiakas-palvelin

PC:n kehittyessä siirryttiin asiakas-palvelinteknologiaan, jota kutsuttiin myös Two-Tier-ympäristöksi. Tier on suomeksi taso. Palvelimet hallitsivat pysyvää palvelinpuolen dataa, ja asiakkailla oli graafinen käyttöliittymä palvelimille eri kommunikaatioprotokollien kautta. [27. luku 1.]

Kommunikaatio monimutkaistui asiakaspuolen tietokoneiden älykkyyden kehityksen myötä. Kommunikaation protokollaspesifikaation ohjelmointi vei kehittäjiltä liikaa aikaa, minkä vuoksi asiakas-palvelinkommunikaatiotaso rakennettiin käyttämään vain yhtä tiettyä protokollaa. [27. luku 1.]

Asiakastietokoneet eivät olleet riittävän tehokkaita liiketoimintalogiikkaan ja monimutkaisiin laskuihin, ja niissä olleiden tietoturvapuutteiden vuoksi niihin jouduttiin rakentamaan kotitekoisia turvaohjelmia. Kommunikaation helpottamiseksi tarvittiin uusi taso tai marshalling-ohjelma, joka hallitsisi asiakas-palvelinvuorovaikutusta. Tällaista asiakas-palvelinvuorovaikutusta hoitavaa osaa kutsuttiin middlewareksi. [27. luku 1.]

5.4 Three-Tier-ympäristö

Three-Tier korjasi Two-Tierin puutteet. Three-Tieria kutsuttiin myös hajautetuksi järjestelmäksi, ja se koostui kolmesta tasosta. Tämä arkkitehtuuri mahdollisti sen, että jokainen osa suoritti sille kuuluvat tehtävät. Ensimmäinen taso oli Front-End, joka oli kuin Two-Tier Front-End, mutta siinä vähennettiin asiakkaan ”älykkyyttä”. Front-Endin, jota kutsuttiin myös thin-clientiksi, vastuualueena oli esittää, vastaanottaa ja validoida käyttäjän data. [27. luku 1.]

Front-Endin tehokkuusongelmat siirrettiin seuraavalle tasolle, jota kutsuttiin Middle-Tieriksi. Middle-Tier oli tehokas ja älykäs palvelin, joka tuki monimutkaista liiketoimintaprosessointia. [27. luku 1.]

Viimeinen taso Three-Tierissä oli kolmas taso, jossa oli Middle-Tierin tavoin tehokkaita tietokoneita. Niiden tarkoitus oli hallita pysyvää liiketoimintadataa. [27. luku 1.]

Vastuun jakaminen toteutettiin RPC-protokollalla, jota tasot käyttivät keskusteluun. Ensimmäinen taso keskusteli keskimmäisen tason kanssa ja keskimmäinen taso kolmannen tason kanssa RPC-protokollalla. [27. luku 1.]

RPC-protokolla oli funktion etäkutsustandardi, jonka avulla funktio voitiin kutsua sijainnista riippumatta. Funktion sijainti oli niin sanotusti läpinäkyvä; se oli paikallinen tai etäinen. RPC:n ansiosta ohjelmoijat pystyivät laajentamaan Three-Tierin Multi-Tieriksi. RPC:n muita hyötyjä olivat palvelimen sijainnin hallinnointi välittäjällä, nimen ja kuorman hallinta, tietoturva, monisäikeisyys sekä verkkoprotokollan riippumattomuus. [27. luku 1.]

5.5 Hajautetut oliot

Hajautetut oliot -teknologia (Distributed objects) tuli RPC:n jatkumona. Hajautetut oliot -teknologia sisälsi RPC:n ominaisuudet, ja se hyödynsi olio-ohjelmointia. Funktion kutsun sijaan se loi olion ilmentymän ja lähetti sille viestejä. Oliot voivat sijaita missä tahansa, kuten funktiot RPC:ssä. Middleware-väliohjelmisto sisälsi yhteisen kokoelman toiminnollisuuksia olioille. Oliot löysivät toisensa suorituksen aikana ja keskustelivat keskenään The Object Request Broker -kanavan kautta. [27. luku 1.]

5.6 Hajautetut komponentit

Keskustietokoneet suorittivat monoliittista ohjelmaa. Asiakas-palvelin- ja hajautetussa järjestelmässä työ voitiin jakaa. Komponenttiteknologian tavoite oli olla ohjelmointikieli, työkalu, käyttöjärjestelmä ja verkkoriippumaton. Component Object Model (COM) perustuu hajautettuihin komponentteihin. [27. luku 1.]

Komponenttiteknologiassa tehtävänjako oli jaettu ja määritelty komponenttikohtaisesti. Komponenttien ylläpito ja uudelleenkäyttö binääritasolla oli vaivatonta. Komponenttiteknologialla rakennettua komponenttikirjastoa voi verrata staattiseen kirjastoon. Komponenttikirjastossa ei tarvittu header-tiedostoja eikä siihen tarvinnut sisällyttää kirjastoa staattisen kirjaston tavoin. [27. luku 1.]

5.7 Component Object Model -standardi

COM-standardi ja Microsoftin COM-implementaatio

Component Object Model (COM) on binäärirajapintastandardi. Microsoftin COM toteuttaa tämän standardin ja tukee näin binääriyhteensopivuutta. [27. luku 3.]

COM-komponentti koostuu COM-luokista ja kirjoitetaan PE-tiedostoformaattiin .dll:ään tai .exe:een. Sen ajonaikailmentymää kutsutaan COM-olioksi tai yleisemmin hajautetuksi olioksi. COM-olio perustuu COM-standardissa määriteltyihin sääntöihin. COM hyödyntää olio-ohjelmointiominaisuuksia, kuten periytymistä, kapsulointia ja polymorfismia. [27. luku 3.]

COM-järjestelmä toteuttaa asiakas-palvelinmallin. COM-komponentit voivat olla laajasti hajautettuja ja toimia sekä asiakkaina että palvelimena. Tästä syystä asiakas-palvelin-notaatio heikkenee ja asiakas-palvelin jätetään yleensä pois, jolloin COM-olioita kutsutaan ohjelmistokomponenttien yhteydessä hajautetuiksi olioiksi. [27. luku 3.]

COM ja binäärirajapinta

Binääriyhteentoimivuus taataan rajapinnoilla. COM hyödyntää C++-kääntäjässä yleisesti käytettyä vptr/vtbl-tekniikkaa tarjotakseen tuen binääriyhteentoimivuuteen, dynaamiseen sitomiseen ja polymorfismiin. Mikä tahansa kieli tai työkalu, jolla on tuki rajapintojen viittauksiin, voi työskennellä ylimääräisellä epäsuoralla viittauksella COM-olioiden kanssa. [27. luku 2.]

COM-rajapinta tukee ominaisuuksia ja metodeja. COM-rajapinta koostuu ominaisuuksien sijaan suurimmaksi osaksi metodeista. Rajapintaominaisuudet ovat käsittelyä varten, eli ne ovat metodeja, jotka palauttavat tai asettavat arvoja. COM- standardi määrittää, että kaikilla COM:a tukevilla rajapinnoilla tulee olla perittynä IUnknown-rajapinta. IUnknow-rajapinta mahdollistaa referenssin laskun ja tyyppimuunnoksen. Referenssien laskun avulla hallitaan olion elämänkaarta. IUnknown sisältää kolme metodia, jotka ovat AddRef, QueryInterface ja Release. QueryInterface metodilla asiakas voi löytää dynaamisesti COM-olion tarjoamat rajapinnat. AddRef inkrementoi olion sisäistä laskuria. Release dekrementoi olion sisäistä laskuria. Olion sisäisen laskurin avulla nähdään, onko olio käytössä. Kun olion sisäinen laskuri on nolla, Release-metodi deallokoi olion. [27. luku 3.]

COM-komponenttien asennus ja kutsu

COM-komponentin sijainti merkitään rekisteriin, jotta COM Service Control Manager (SCM) -palvelunhallitsija löytää sen. Rekisteri toimii paikallisesti asennetun COM-komponentin säilytyspaikkana. [27. luku 2.]

COM-komponentit paljastavat toimintansa yhden tai useamman rajapinnan kautta. COM-luokkien metodeihin pääsee käsiksi vain rajapinnan kautta. Asiakas käyttää COM-olioita sopimuksen mukaisilla julkaistuilla rajapinnoilla. COM-komponenteilla on metatiedostot, joissa COM-tyyppejä kuvataan kieliriippumattomalla tavalla. COM-komponenttien metatiedostot kirjoitetaan Interface Definition Language -kielellä (IDL). Microsoft käyttää tähän tarkoitukseen MIDL:ää. MIDL on perustettu DCE IDL:n päälle. MIDL määrittelee luokat, kirjastot, rajapinnat ja metodit, mikä käännetään MIDL-kääntäjällä rajapintasäilytyspaikkaan. COM:ssa rajapintasäilytyspaikkaa kutsutaan tyyppikirjastoiksi. Sen tiedoston pääte on yleensä .tbl (type libraries). COM-olio paljastaa tyyppitiedot tyyppikirjaston kautta, näin ohjelma löytää dynaamisesti palvelut mistä tahansa COM-oliosta. [27. luku 3.]

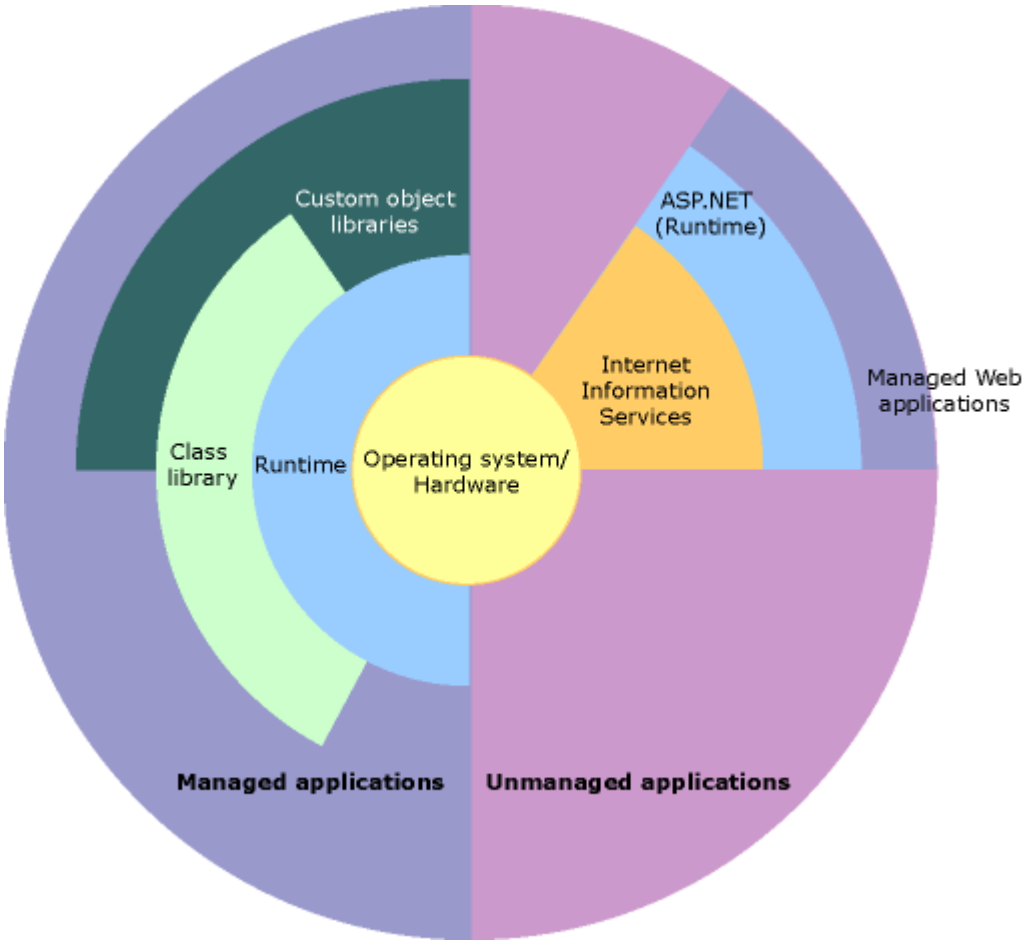
Tyyppikirjastossa käytetään library-avainsanaa kerrottaessa MIDL-kääntäjälle, että tyyppikirjasto luodaan tietylle komponentille. coclass-avainsanan sisällä määriteltyjen rajapintojen sanotaan olevan COM-olion implementoimia. Sekä library- että coclass-avainsanoille määritetään UUID, joka on 128-bit globally unique identifier (GUID). Koska GUID on ajasta ja paikasta riippumaton tunnus, saadaan tällä automaattinen versionhallinta. Coclassin UUID:ta kutsutaan myös CLSID:ksi. CLSID:n sijaan voidaan asettaa progid-merkkijononimi, joka on luettavampi ja helpompi muistaa, mutta tällöin poistuu CLSID:n ominaisuudet versionhallinta ja yksilöllisyys. COM-rajapintojen metodeihin voidaan luoda useita eri implementaatioita ja kutsua niitä omilla IID:eillä, joka on lyhenne Interface ID:stä, jotka toimivat myös GUID:na. [27. luku 3.]

COM-luokat voidaan luoda joko niiden omilla luokka-ID:eillä (CLSID) tai niiden progid:llä (programmatic identifier string). Asiakasohjelmointikielissä COM-ohjelmointia on helpotettu. Addref ja Release kutsutaan automaattisesti VB:stä. VBA:ssa COM-olio luodaan progid:llä. [27. luku 3.]

6 .NET

6.1 .NET:n arkkitehtuuri ja sisältö

.NET koostuu ajoympäristöstä (Common Language Runtime) ja luokkakirjastosta. Common Language Runtime on ajoympäristö, joka hallitsee suoritettavia ohjelmia. Ajoympäristö voi toimia joko erikseen itsenäisenä tai isännöitynä ei-hallitussa ohjelmassa, kuten Internet Information Services -palvelimen (IIS) yllä. Luokkakirjasto tarjoaa testattuja, uudelleenkäytettäviä koodeja sisältävät kirjastot, joita ohjelmoija voi kutsua ohjelmistaan. Kuvassa 3 näkyvät järjestelmän, CLR:n ja luokkakirjastojen suhteet ohjelmiin sekä se, miten hallittu koodi operoi suuressa arkkitehtuurissa. [28.]



Kuva 3. Järjestelmän, CLR:n ja luokkakirjaston suhteet ohjelmiin ja hallitun koodin toiminta suuressa arkkitehtuurissa [28].

Luokkakirjastot on jaettu kahteen osaan: FCL ja BCL. Base Class Library koostuu tyypeistä, jotka esittävät sisäänrakennetut yhteisen kielen rajapinnan datatyypit, yksinkertaisen tiedostoon pääsyn, muokatut ominaisuudet, turvallisuusominaisuudet, merkkijonon muokkauksen, formatoinnin, virrat ja kokoelmat. Framework Class Libraryyn kuuluvat BCL ja sen ulkopuoliset luokat, jotka sisältävät muun muassa luokat työpöytäohjelmien ja verkkopalveluiden kehitykseen. [29.]

Ohjelmoijat voivat luoda .NET Frameworkin Portable Class Library -kokoonpanoja (PCL), jotka toimivat monella .NET Framework -alustalla. Tällaisia ovat muun muassa Windows-käyttöjärjestelmät, Xbox 360 ja Windows Phone. [30.]

Esiasennettu .NET-versio riippuu Microsoft Windowsin versiosta. Asennukseen tarvitaan pääkäyttäjän lupa.

6.2 .NET:n nimi

.NET:llä oli monta eri nimeä ennen sen lopullista nimeä. .NET-kirjasto sisältää edelleen jäänteitä vanhemmista nimistä eri tiedostoissa ja dokumenteissa, kuten MSCorLib.dll, joka on pääkokoonpano FCL:ssä. Ennen kuin .NET nimi oli valittu, uuden alustan jatkaja oli COM, joten se koodinimettiin COM 3.0:ksi. Sittemmin nimeksi valittiin Common Object Runtime, josta tuli nimi mscorlib, joka on lyhenne Microsoft Common Object Runtime Librarystä. Myöhemmin nimenä oli COM+ 2.0, NGWS ja lopulta .NET. [31; 32; 33; 34; 35.]

6.3 .NET:n ja COM:n erot

.NET tarjoaa aiempaa enemmän valmiita ratkaisuja ja poistaa entisen Windows-ympäristön puutteita. [36, s. xxii-xxv.]

Ennen .NET:a Windows-ohjelmoinnissa funktiot raportoivat eri tavoin virhetilanteista: funktiot palauttivat Win32-tilakoodeja, HRESULT-arvoja tai loivat poikkeustapahtumia. .NET yksinkertaistaa virhetilanteiden käsittelyn käyttämällä CLR-poikkeusolioita, jotka ovat käytettävissä moduuli- ja ohjelmointikieliriippumattomasti. CLR tekee virheiden paikallistamisen helpommaksi kutsupinon selailulla. [36, s. xxii-xxv.]

CLR yksinkertaisti ohjelmointimallin. Se ei sisällä ennen .NET:ä olleita käsitteitä kuten rekisteri, GUID, IUnknown, AddRef, Release ja HRESULT. Nämä on kuitenkin hallittava, jos on laadittava .NET-sovellus, joka toimii yhteen olemassa olevan ja vanhalla toteutetun koodin kanssa. [36, s. xxii-xxv.]

.NET-sovellus toimii kaikissa koneissa, joissa on ECMA-yhteensopiva versio CLR- ja FCL-ympäristöistä, ja se on täten alustariippumaton. [36, s. xxii-xxv.]

.NET Framework antaa mahdollisuuden käyttää olemassa olevia COM-komponentteja. CLR-yhteensopivien kielten on käytettävä CTS-tekniikkaa (Common Type System), mikä parantaa ohjelmointikielten liittämistä. Tätä ennen COM mahdollisti eri ohjelmointikielten yhteistoiminnan, mutta ei toisen kielen tyyppejä tai DLL-tiedostojen Win32-funktioita. [36, s. xxii-xxv.]

Ohjelmointi oliopohjaisen ohjelmointimallin avulla onnistuu tarjoamalla yhteinen ohjelmointimalli. Näin kaikki voi tapahtua .NET Frameworkin olioiden kautta, toisin kuin tätä edeltävässä ratkaisussa, jossa käyttöjärjestelmän toiminnot piti hakea erikseen DLL-kirjastojen funktioiden ja COM-olioiden kautta. [36, s. xxii-xxv.]

.NET Framework -sovellus asennetaan kopioimalla tiedostot hakemistoon ja lisäämällä pikakuvake käynnistysvalikkoon, työpöydälle tai pikakäynnistyspalkkiin (Quick Launch bar). Sovellus poistetaan tuhoamalla sen tiedostot. Aiemmin asentamisen yhteydessä jouduttiin asentamaan useita tiedostoja, tekemään muutoksia rekisteriin ja luomaan pikakuvakkeita ja ohjelmiston poistaminen jälkiä jättämättä oli lähes mahdotonta. Koska yksinkertaisiksi tyypeiksi kutsuttaviin .NET Framework - komponentteihin ei viitata rekisterin kautta, ei tässä olla rekisterin kanssa tekemisissä kuten COM-komponenttien kanssa. [36, s. xxii-xxv.]

.NET Framework pitää sovelluksen komponentit erillään toisista sovelluksista. Sovellus lataa käynnistyessään juuri ne komponentit, joista se on alun perin rakennettu ja jotka on testattu. Näin ollen sovelluksen pitäisi toimia jatkossakin, jos se toimii asennuksen jälkeen. Tätä ennen Windows-ohjelmoijat kohtasivat ”DLL-helvetin”, jossa uuden sovelluksen komponentit korvasivat käytössä olevan sovelluksen komponentteja, jonka seurauksena vanha sovellus alkoi käyttäytyä kummallisesti tai lakkasi kokonaan toimimasta. [36, s. xxii-xxv.]

CLR osaa varmistaa, että koodi on käyttöön oikeantyyppinen, jolloin olioita kutsutaan aina oikealla tavalla. Tällaiset tarkistukset estävät monia yleisiä ohjelmointivirheitä ja klassisia tietoturvahyökkäyksiä. CLR antaa myös hyvät keinot virheiden löytämiseen mahdollistamalla ohjelmien virheenjäljityksen ohjelmointikielestä riippumattomasti. [36, s. xxii-xxv.]

Ennen .NET:a ohjelmien turvallisuus hallinnoitiin käyttäjätunnuksille annetuilla oikeuksilla. .NET asettaa käyttöoikeudet koodeihin. [36, s. xxii-xxv.]

Ohjelmointikielten liittäminen tyypityksellä helpottaa palveluluokkien koodin kierrätystä muualla tapahtuvassa sovelluskehityksessä. [36, s. xxii-xxv.]

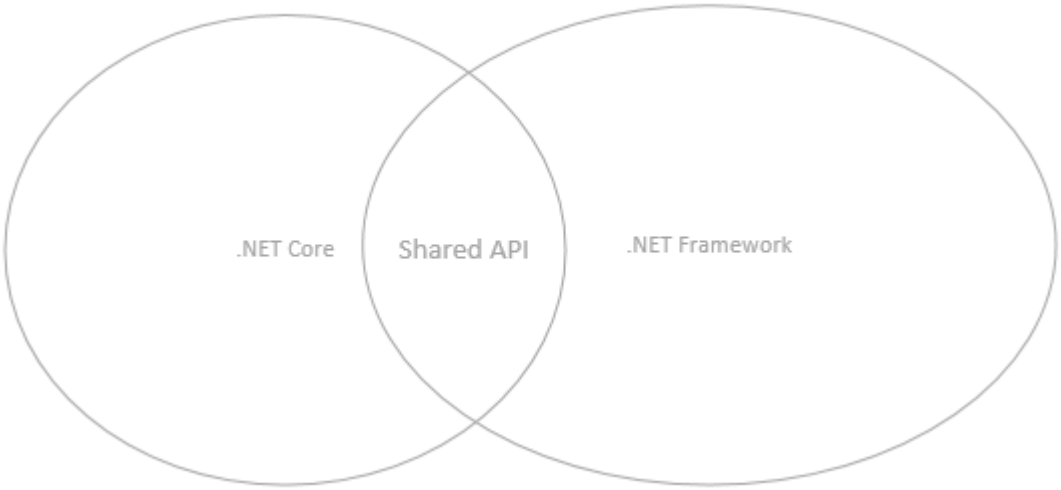
CLR suorittaa automaattisen muistinhallinnan (roskienkeruu) kirjaamalla resurssin käytön ja huolehtimalla, ettei sovellus kaadu resurssien ylittämiseen. Tätä ennen ohjelmointi vaati resurssien, kuten tiedostojen, muistin, näytön tilan, verkkoyhteyksien ja tietokannan hallintaa. Yleisin virhe oli resurssin vapauttamisen unohtaminen, joka johti lopulta sovelluksen toiminnan häiriintymiseen. [36, s. xxii-xxv.]

6.4 .NET Core

.NET Core on .NET Foundationin projekti. .NET Foundation on itsenäinen organisaatio, joka edistää avointa kehitystä ja yhteistyötä .NET-ekosysteemin ympärillä. Se toimii foorumina yhteisölle ja kaupallisille ohjelmoijille sekä laajentaa ja vahvistaa .NET- ekosysteemin tulevaisuutta kannattamalla avoimuutta ja yhteisön osallistumista innovaation kehittämiseen. [37.]

ASP.NET Core 1.0 on .NET Core päälle rakennettu ohjelmamalli, jolla kehitetään verkkopalveluja ja verkkosivuja. .NET Core tunnettiin koodinimellä ”Project K”. .NET Core ja ASP.NET Core syntyivät Microsoftin saaman asiakaspalautteen, joka oli saada ”.NET Linuxin käyttöjärjestelmän päällä”, sekä Microsoft Server Teamin pienen palvelintuotteen Windows Nanon pohjalta. [38.]

.NET Core on nimetty ”Coreksi”, koska se sisältää .NET Frameworkin ydinominaisuuksia mukaan lukien ajoajan ja kehystyökalukirjaston. Kuvassa 4 näkyy tämä suhde. [39.]



Kuva 4. NET Core- ja .NET Framework -alijoukko-ylijoukkosuhde.

ASP.NET MVC oli avointa lähdekoodia, mutta sen pohjana ollut alusta .NET Framework ei ollut. ASP.NET Core on nyt kokonaisuudessaan avoin lähdekoodiverkkoalusta, dokumentaatio mukaan lukien. [38.]

6.5 .NET:n ja .NET Coren erot

.NET Core ei tue kaikkia .NET Framework -ohjelmamalleja, koska moni niistä on rakennettu Windows-teknologioiden päälle, kuten WPF (.NET:n työpöytäohjelmamalli), joka on rakennettu DirectX:n päälle (rakennettu COM:n päälle). Konsoli ja ASP.NET Core -ohjelmamallit on rakennettu .NET Coren ja .NET Frameworkin päälle. [38.]

.NET Core sisältää monia samoja API:ja, mutta vähemmän kuin .NET Framework. .NET Core implementoi .NET Standard Library API:a, joka tulee edelleen kasvamaan ja sisältämään entistä enemmän .NET Framework BCL API:ja. Kerron Standard Librarystä luvussa 6.6. [38.]

.NET Core on avoin lähdekoodi, kun taas .NET Frameworkissa avointa lähdekoodia on vain lukuluvalla alaosa. .NET Coren ajoympäristö, kirjastot, kääntäjä, kielet ja työkalut ovat kaikki avoimena lähdekoodina GitHubissa, jossa osallistuneiden koodit on hyväksytty, testattu ja täysin tuettu. .NET Corea voi käyttää ilman rajoitteita, millä tahansa käyttöjärjestelmällä, työkalulla ja sovelluksella. .NET Framework tukee Windowsia ja Windows Serveriä, kun taas .NET Core tukee näiden lisäksi myös macOSia ja Linuxia. .NET Core -alusta käyttää MIT- ja Apache 2 -lisenssiä. .NET Coren dokumentaatio on lisensoitu CC-BY:n alaisena. .NET Core on kehitetty yhteisön kanssa ja Microsoftin tuella. [38.]

6.6 .NET Standard Library

.NET -ajoympäristön käyttäytyminen on standardisoitu ECMA 335 -spesifikaatioksi. .NET Standard Library -spesifikaatio standardisoi .NET Base Class Libraryn. Näin ohjelmoijat voivat tuottaa kannettavia kirjastoja, jotka toimivat kaikissa .NET ajoajoissa, mikä yhdenmukaistaa .NET ekosysteemiä. [40.]

.NET Core 1.0 implementoi standardikirjaston, kuten myös .NET Framework ja Xamarin. .NET Standard ratkaisee .NET-ohjelmoijien ongelman koodin jakamisesta alustojen läpi. .NET Standard on kokoelma ohjelmointirajapintoja, jonka kaikki .NET-alustat täytyy implementoida. Tämä yhtenäistää .NET-alustat ja estää niiden pirstoutumista tulevaisuudessa. .NET Standard 2.0 tulee olemaan implementointu .NET Frameworkilla, .NET Corella ja Xamarinilla. .NET Corelle tämä lisää monia muita pyydettyjä API:ja. [41.]

.NET Standard 2.0 sisältää yhteensopivuusvälilevyn .NET Framework -binääreille. Se suurentaa kirjastokokoelmaa, jota voi viitata .NET Standard -kirjastoista. .NET Standard API -määritelmä on nähtävissä dotnet/standard-säilytyspaikassa GitHubissa. [41.]

7 PowerShell-komentorivi

PowerShell-komentorivi on rakennettu .NET Framework CLR:n ja .NET Frameworkin päälle. Se hyväksyy ja palauttaa .NET Frameworkin olioita. PowerShell on alustariippumaton GitHubista saatavilla oleva avoimen lähdekoodin projekti. Se on järjestelmänvalvojalle ja hallinnon automaatioon suunniteltu automaatioalusta ja komentosarjakieli. [42; 43.]

PowerShell on suunniteltu parantamaan komentoriviä ja komentosarjaympäristöä poistamalla pitkäaikaiset ongelmat ja lisäämällä ominaisuuksia. PowerShell yhtenäistää komentorivityökalut ja skriptauksen poistamalla valinnat, jotka on tyypillisesti jätetty ohjelmoijille. [44.]

PowerShell helpottaa sen ominaisuuksien löytämistä omilla cmdleteillä, kuten “Get-Command”-cmdlet, jolla etsitään komento ja “Get-Help”-cmdlet, jolla etsitään cmdletin dokumentaatio. PowerShell tekee näin helpoksi siirtyä komentojen kirjoittamisesta komentosarjojen luomiseen ja ajamiseen vuorovaikutteisesti. [44.]

PowerShell päästää käsiksi komentorivityökaluihin, COM-olioihin ja mahdollistaa .NET Framework Class Libraryn (FCL) käyttämisen. Tämä on parannus CMD.EXE:een, joka tarjoaa vuorovaikutteisen ympäristön monella komentorivityökalulla ja WSH-komentosarjaohjelmointiin, joka tarjoaa monta eri komentorivityökalua sekä COM-automaatiooliot, mutta ei interaktiivista ympäristöä. [44.]

PowerShell sisältää interaktiivisen syötteen ja komentosarjaympäristön, jota voi käyttää yksittäin tai yhdessä. PowerShellin avulla järjestelmänvalvoja voi automatisoida järjestelmähallitsijan resursseja ajamalla komentoja joko suoraan tai komentosarjojen kautta. PowerShell antaa tiedostojärjestelmäpääsyn lisäksi pääsyn myös rekisteriin ja digitaalisen allekirjoituksen sertifikaatiotallennukseen. Sillä voi hyödyntää olemassa olevia työkaluja ulkoisena komentona. Se on optimoitu toimimaan strukturoidun datan kanssa kuten XML, JSON sekä REST API:n ja oliomallien kanssa. Windows PowerShell sisältää cmdlet-konseptin, joka on yhden funktion komentorivityökalu. Windows PowerShell sisältää yli sata perusydin-cmdletia, jotka käyttäjä voi itse kirjoittaa ja jakaa. [42.]

PowerShell Gallery on PowerShellin sisällön keskussäilytyspaikka. PowerShell Gallerystä löytää uusia PowerShell-komentoja ja Desired State Configurationin (DSC), josta kerron DSC-kappaleessa. PowerShellGet-moduuli sisältää cmdlettejä, joilla voi etsiä, asentaa, päivittää ja julkaista PowerShell-tuotteita, kuten moduuleita, DSC-lähteitä ja komentosarjoja https://www.PowerShellGallery.com-säilytyspaikasta ja muista yksityisistä säilytyspaikoista. [45; 46.]

PowerShell Core

Versiosta 5.1 lähtien PowerShellin saa kahtena eri versiona: työpöytäversiona, joka sisältää .NET Frameworkin päälle rakennetun PowerShellin, ja Core-versiona, joka sisältää .NET Coren päälle rakennetun PowerShellin. [47.]

PowerShellin työpöytäversio sisältää kaikki toiminnallisuudet, kun taas vähemmän toiminnallisuuksia sisältävä PowerShellin Core-versio on tarkoitettu tiukkaresurssisiin ympäristöihin, kuten Nano Server ja Windows IoT. [47.]

DSC

Desired State Configuration (DSC) on hallinta-alusta Windows PowerShellissa. Se tarjoaa kokoelman kielilaajennuksia, kuten uusia cmdlettejä sekä resursseja, joilla voidaan määritellä deklaratiivisesti, miten ohjelmaympäristö asetetaan. [48.]

DSC tarjoaa työkalut testaukseen, olemassa olevien ohjelmapalveluiden ylläpitoon ja muokkaukseen sekä ohjelmapalveluympäristöön. [49.]

PowerShell ISE

PowerShell Integrated Scripting Environment (ISE) on Windows-ohjelma, joka sisältää sisäänrakennetun editorin komentosarjojen kirjoittamiseen, testaukseen ja virhejäljitykseen. ISE-ominaisuuksiin kuuluu muun muassa IntelliSense-sarkainmerkkitäydennys, syntaksin korostus ja kontekstiriippuvainen apu sekä lisäosilla lisättävät toiminnallisuudet. [48.]

Windows Management Framework

Windows PowerShell on osa Windows Management Frameworkia (WMF). WMF on ajuri, joka sisältää päivityksiä ja ohjelmia Windowsille ja Windows Serverille hallintarajapintoihin. Viimeisin WMF-ajuriversio 5.0 tuo OneGetin, joka on PowerShellin moduulinhallintajärjestelmä ja vastaa Ubuntun ”apt-get”-paketinhallintaa. [48.]

Microsoft Operation Management Suite

Operations Management Suite (OMS) -automaatiotyökalusarja mahdollistaa työskentelyn Azuren tai muiden pilvien kautta PowerShellilla ja DSC:llä Linuxissa tai Windows Serverissä käyttäen graafista rajapintaa. [48.]

8 Yhteenveto

Insinöörityössä selvitettiin PowerShell Core ja sitä edeltävät komentorivipohjaiset hallintatyökalut ja niiden taustalla olevat teknologiat. Insinöörityössä selvitettiin myös motivaatio eri teknologioihin, komentorivipohjaisten hallintatyökalujen kehitys ja niiden käyttökohteet.

Microsoftin lähestymistapa poikkeaa muista. Se käyttää jopa uusimmassa Microsoft Windows 10 -käyttöjärjestelmässä vuosien takaista teknologiaa. Työssäni selvitettyä tietoa tarvitaan etenkin työskenneltäessä Microsoftin ohjelmien parissa. Hyödynsin tietoja työskennellessäni insinööriharjoittelijana ABB:lla. Tietotaitoa voin soveltaa ohjelmoinnista MS Office -ohjelmiin asti.

Tutkimusta voisi jatkaa syventymällä vielä tarkemmin kunkin käytössä olevan teknologian toteuttamiin tekniikkoihin, jotta uusien projektien alkaessa niiden valitseminen ja käyttäminen olisi vaivatonta. Esimerkkinä .NET-kirjaston hyödyntäminen PowerShellilla ja raportointi graafisella käyttöliittymällä Excel COM -komponentin avulla.

Tässä insinöörityössä hankalinta oli terminologia. Microsoft käyttää termejä, joita ei ole vielä suomennettu. Eri osa-alueet vaativat uuden terminologian hallinnan, jotta asiaan voi syventyä. Eri osa-alueille siirryttäessä asiat selitetään eri termein, kuten esimerkiksi COM-komponentti ja .NET-tyyppi. Jokaiseen aihealueeseen tuli perehtyä erikseen, jotta pystyi luomaan paremman kokonaiskuvan.

Liitteenä 1 olevaa käsitekarttaa voisi vielä laajentaa sisällyttäen siihen tiedon siitä, mitkä osat ovat poistumassa, mitkä osat korvataan millä ja mitkä teknologiat ovat pysyvimpiä. COM-tekniikkaa ei enää käytetä ajureiden valmistuksessa, ja Windows 10 -päivityksessä poistettiin ActiveX uudesta Edge-selaimesta kokonaan. MS Office käyttää kuitenkin edelleen COM:a. Käsitekarttaan voisi lisätä myös, ketkä Microsoftin lisäksi hyödyntävät sen teknologioita. Esimerkiksi Applen käyttöjärjestelmässä Automation-automaatiotyökaluohjelma automatisoi tehtävät käyttöliittymän kautta hyödyntämällä Applen COM-standardin implementaatiota.

Lähteet

1 MS-DOS. 2016. Verkkodokumentti. Computer Hope. <http://www.computer-hope.com/jargon/m/msdos.htm>. Luettu 1.11.2016.

2 The Windows NT Command Shell. 1998. Verkkodokumentti. Microsoft. <https://technet.microsoft.com/library/cc750982.aspx>. Luettu 1.11.2016.

3 Microsoft DOS cmd and command. 2016. Verkkodokumentti. Computer Hope. <http://www.computerhope.com/cmd.htm>. Luettu 1.11.2016.

4 8. Applications. 2016. Verkkodokumentti. Rhys Haden. <http://www.rhy-shaden.com/nt_app.htm>. Luettu 3.11.2016.

5 Campbell, Tom. 1994. The Norton Utilities 8.0. Verkkodokumentti. Classic Computer Magazine Archive. <http://www.atarimagazines.com/compute/issue167/78_The_Norton_Utilities.php>. Luettu 1.11.2016.

6 Upgrade Your Windows Command Prompt / Windows CMD Command Line. 2016. Verkkodokumentti. JP Software. <https://jpsoft.com/take-command-windows-scripting.html>. Luettu 1.11.2016.

7 Soulami, Tarik. 2012. Inside Windows Debugging. Verkkodokumentti. Google Play. <https://books.google.fi/books?id=GbNCAwAAQBAJ&pg=PT170&dq=active+scripting&hl=en &sa=X&ved=0ahUKEwj41rrQ_4fQAhXya5oKHfaw-BQUQ6AEIJDAC#v=onepage&q=active% 20scripting&f=false>. Luettu 1.11.2016.

8 Using the command-based script host (CScript.exe). 2016. Verkkodokumentti. TechNet Microsoft. <https://technet.microsoft.com/enus/library/bb490887.aspx>. Luettu 1.11.2016.

9 To run scripts using the command-line-based script host (Cscript.exe). 2016. Verkkodokumentti. TechNet Microsoft. <https://technet.microsoft.com/enus/library/bb490816.aspx>. Luettu 1.11.2016.

10 CScript.exe / wscript.exe. 2016. Verkkodokumentti. SS64.com <http://ss64.com/vb/cscript.html>. Luettu 1.11.2016.

11 Windows Script Host Object Model. 2009. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/a74hyyw0(v=vs.84).aspx>. Luettu 1.11.2016.

12 Microsoft Operations Manager. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/aa505337.aspx>. Luettu 1.11.2016.

13 Part 1 - The Basics. 2016. Verkkodokumentti. TechNet Microsoft. <https://technet.microsoft.com/en-us/library/ee176904.aspx>. Luettu 1.11.2016.

14 Using Windows Script Files (.wsf). 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/library/15x4407c(v=vs.84).aspx>. Luettu 3.11.2016.

15 What Is WSH. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/shzd7dy4(VS.85).aspx>. Luettu 3.11.2016.

16 Interprocess Communications. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/windows/desktop/aa365574(v=vs.85).aspx>. Luettu 3.11.2016.

17 About Dynamic Data Exchange. Verkkodokumentti. 2016. Microsoft. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms648774(v=vs.85).aspx>. Luettu 1.11.2016.

18 About the DDEML. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms648713(v=vs.85).aspx>. Luettu 1.11.2016.

19 About Atom Tables. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms649053(v=vs.85).aspx>. Luettu 1.11.2016.

20 DDE Function. 2016. Verkkodokumentti. Microsoft. <https://support.office.com/enus/article/DDE-Function-79e8b21c-2054-4b48-9ceb-d2cf38dc17f9>. Luettu 1.11.2016.

21 Williams, Mickey & Bennett, David. 2000. Visual C++6 Unleashed. SAMS.

22 Linked Objects. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/enus/library/windows/desktop/ms678811(v=vs.85).aspx>. Luettu 1.11.2016.

23 Embedded Objects. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms679749(v=vs.85).aspx>. Luettu 1.11.2016.

24 About Dynamic Data Exchange. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms648774(v=vs.85).aspx>. Luettu 1.11.2016.

25 OLE Concepts and Requirements Overview. 1999. Verkkodokumentti. Microsoft. <https://support.microsoft.com/en-us/kb/86008>. Luettu 1.11.2016.

26 Creating a package. 2016. Verkkodokumentti. Microsoft. <https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/enus/packager_rules.mspx?mfr=true>. Luettu 1.11.2016.

27 Thai, Luan L. 2011. Learning DCOM. O'Reilly Media; 1 edition.

28 Overview of the .NET Framework. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx>. Luettu 3.11.2016.

29 BCL (Base Class Library) vs FCL (Framework Class Library). 2016. Verkkodokumentti. Stackoverflow. <http://stackoverflow.com/questions/807880/bcl-base-class-library-vs-fcl-framework-class-library>. Luettu 3.11.2016.

30 Portable Class Libraries. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/gg597391(v=vs.100).aspx>. Luettu 3.11.2016.

31 Cidade, Mark. 2013. Future of the Component Object Model. Verkkodokumentti. Stack Overflow. <http://stackoverflow.com/questions/133632/future-of-the-componentobject-model>. Luettu 1.11.2016.

32 Burton, Kevin. 2002. Net Common Language Runtime Unleashed. Verkkodokumentti. Google Play. <https://books.google.fi/books?id=3059QRxPNQcC&pg=PT38&lpg=PT38&dq=COM3+.NET +framework+name&redir_esc=y#v=onepage&q=COM3%20.NET%20framework%20name&f =false>. Luettu 1.11.2016.

33 Cidade, Mark. 2008. What does mscorlib stand for?. Verkkodokumentti. Stack Overflow. <http://stackoverflow.com/questions/15061977/what-does-mscorlib-stand-for>. Luettu 1.11.2016.

34 Moth, Daniel. 2016. mscorlib.dll. Verkkodokumentti. The Moth. <http://www.danielmoth.com/Blog/mscorlibdll.aspx>. Luettu 3.11.2016.

35 Grygus, Andrew. 2016. Microsoft's NGWS => Microsoft.NET. Verkkodokumentti. Aax. <http://aaxnet.com/editor/edit007.html>. Luettu 3.11.2016.

36 Richter, Jeffrey. 2003. Inside Microsoft .NET Ohjelmointi. Edita Prima Oy.

37 About the .NET Foundation. 2016. Verkkodokumentti. .NET Foundation. <https://www.dotnetfoundation.org/about>. Luettu 1.11.2016.

38 Announcing .NET Core 1.0. 2016. Verkkodokumentti. Microsoft. <https://blogs.msdn.microsoft.com/dotnet/2016/06/27/announcing-net-core-1-0/>. Luettu 1.11.2016.

39 dotnet/core. 2016. Verkkodokumentti. GitHub. <https://github.com/dotnet/core>. Luettu 1.11.2016.

40 .NET Standard Library. 2016. Verkkodokumentti. Microsoft. <https://docs.microsoft.com/en-us/dotnet/articles/standard/library>. Luettu 3.11.2016.

41 Landwerth, Immo. 2016. Introducing .NET Standard. Verkkodokumentti. .NET Blog. <https://blogs.msdn.microsoft.com/dotnet/2016/09/26/introducing-net-standard/>. Luettu 1.11.2016.

42 PowerShell/PowerShell. 2016. Verkkodokumentti. GitHub. <https://github.com/PowerShell/PowerShell>. Luettu 2.11.2016.

43 Getting Started with Windows PowerShell. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/powershell/scripting/getting-started/getting-started-with-windows-powershell>. Luettu 3.11.2016.

44 About Windows PowerShell. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/powershell/scripting/getting-started/fundamental/about-windows-powershell>. Luettu 3.11.2016.

45 PowerShell Gallery. 2016. Verkkodokumentti. Microsoft. <https://www.powershellgallery.com/>. Luettu 5.11.2016.

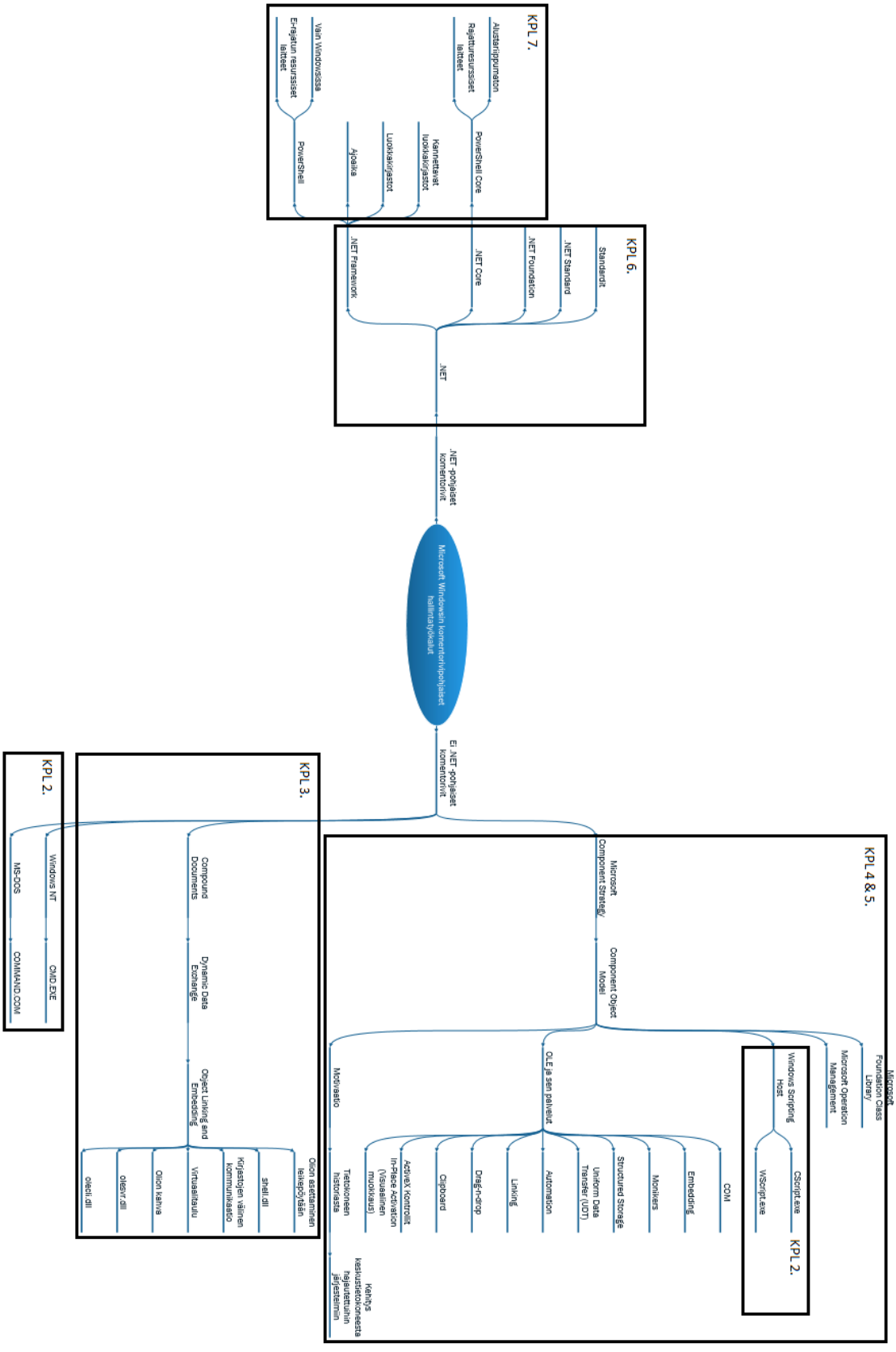
46 The PowerShell Gallery. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/powershell/gallery/readme>. Luettu 5.11.2016.

47 Developing PowerShell Cmdlets for Nano Server using the PowerShell Core SDK. 2016. PowerShell Team. Verkkodokumentti. Microsoft. <https://blogs.msdn.microsoft.com/powershell/2016/05/04/developing-powershell-cmdletsfor-nano-server-using-the-powershell-core-sdk/>. Luettu 1.11.2016.

48 PowerShell. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/enus/powershell/mt173057.aspx>. Luettu 5.11.2016.

49 Windows PowerShell Desired State Configuration Overview. 2016. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/powershell/dsc/overview>. Luettu 5.11.2016.

Microsoft Windowsin komentorivipohjaiset hallintatyökalut



Käsiksipääsytavat Microsoft Windowsin eri osiin