Juha Ahola-Olli

Microsoft .NET

Tietojärjestelmätieteen
Kandidaatintutkielma
27.5.2005

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Ahola-Olli, Juha Tuomas

Microsoft .NET / Juha Ahola-Olli

Jyväskylä: Jyväskylän yliopisto, 2005.

46 s.

Kandidaatin tutkielma

Tässä tutkimuksessa tutkitaan Microsoftin .NET-ohjelmistoarkkitehtuuria, sen rakennetta ja tärkeimpiä osia sekä sen hyötyjä ja haittoja perinteiseen Windows-ohjelmointiin verrattuna. Koska aiheesta ei ole tehty vielä kovin paljon arkkitehtuurin perusasioita käsitteleviä tutkimuksia, on tämän tutkimuksen tarkoitus paikata tätä aukkoa. Tutkimuksen tarkoitus on tuoda lisätietoa nimenomaan arkkitehtuurin alustasta (platform) ja kehyksestä (framework). Näiden lisäksi lopuksi käydään lyhyesti läpi arkkitehtuurin suurimpia etuja ja haittoja. Tutkimus on rajattu siten, että siinä keskitytään nimenomaan Microsoftin kaupalliseen .NET toteutukseen. Muita toteutuksia, kuten Rotor käydään lyhyesti läpi ja verrataan eroja Microsoftin toteutukseen.

Microsoft .NET-arkkitehtuuriin on perehdytty kirjallisuuden pohjalta ja tämä tutkielma perustuu eri lähteistä koottuun tietoon. Aihetta käsitellään myös käyttäen apuna lyhyitä ohjelmointiesimerkkejä.

AVAINSANAT: .NET, .NET Platform, .NET Framework, CLR, ASP.NET, Windows Forms.

SISÄLLYSLUETTELO

1 JOHDANTO

Ohjelmistotekniikka on muuttunut viime aikoina paljon. Hajautetut ohjelmistot ovat yhä tavallisempia ja myös internetistä on tullut yhä tärkeämpi. Ohjelmistot ovat aikaisempaa suurempia ja monimutkaisempia. Myös tietoturvaan on syytä kiinnittää yhä enemmän huomiota, sillä monimutkaiset ohjelmistot sisältävät usein tietoturva-aukkoja ja internet on pullollaan vihamielisiä matoja ja skriptejä.

Yhä useammat ihmiset joutuvat käyttämään työssään useampaa kuin yhtä tietokonetta tai muuta älylaitetta, kuten puhelinta. Heille olisi tärkeää, että heidän tietonsa, kuten kalenteri, kontaktit ja sähköpostit, olisivat saatavilla riippumatta paikasta ja ajasta. Olisi työn kannalta helppoa ja tehokasta jos esimerkiksi kalenterimerkinnät pääsisi tarkistamaan miltä tahansa koneelta ilman, että koneella tulee olla tiettyä ohjelmaa tai itse tiedostoja.

Näitä tarpeita vastaamaan ja ongelmia ratkaisemaan on kehitetty Microsoftin .NET-arkkitehtuuri, joka helpottaa ohjelmistokehittäjän kannalta nykyaikaisten sovellusten tekemistä. Käyttäjän kannalta etuja ovat tietoturvan paraneminen sekä ohjelmistojen ja internetin helpompi käyttö.

Useiden eri käyttöympäristöjen ongelma on myös pyritty ratkaisemaan .NETissä Web-palveluiden avulla. Web-palvelut ovat verkossa toimivia palveluita, joiden avulla käyttäjä pääsee kaikkiin tietoihinsa, kuten kalenteriin ja sähköposteihin verkon kautta riippumatta päätelaitteesta. Näin jokaisella koneella ei tarvitse olla käyttäjän tiedostoja, vaan ne löytyvät keskitetysti verkosta.

Tämän tutkimuksen tavoitteena on selvittää mikä Microsoft .NET-arkkitehtuuri on ja mitä hyötyä siitä on. Koska aihe on varsin tuore, eikä siitä ole vielä ehditty tehdä paljon tutkimuksia, aion esitellä arkkitehtuuria yleisesti sekä sen perusrakennetta. Rakenteen lisäksi aion tutkia mitä etuja .NETistä on

ohjelmistotekniikan ja ohjelmistojen kehittäjän kannalta perinteiseen Windows-ohjelmointiin. Pyrin käsittelemään tekniikkaa luonnollisesti myös kriittisestä näkökulmasta. Tutkimus on rajattu siten, että siinä keskitytään nimenomaan Microsoftin kaupalliseen .NET toteutukseen. Muita toteutuksia, kuten Rotor käydään lyhyesti läpi ja verrataan eroja Microsoftin toteutukseen. Nimenomaan Microsoftin toteutukseen syvennytään sen vuoksi, että se on toteutuksista selvästi laajin ja monipuolisin. Muut toteutukset ovat ainakin toistaiseksi enemmän kokeiluasteella.

Toisesta kappaleesta löytyy yleinen johdatus Microsoft .NET-ohjelmistoarkkitehtuuriin. Kappaleista 3 ja 4 löytyvät tutkimuksen tärkeimmät asiat eli .NET platform ja .NET framework. Varsinkin luku 3 on tärkeä osa tutkimusta, sillä framework sisältää kaikki arkkitehtuurin tärkeimmät komponentit, kuten ajoympäristön ja yhteiset luokkakirjastot. Molemmat komponentit ovat tärkeitä ja varsinkin CLR:n varaan on rakennettu suuri osa arkkitehtuurin toiminnallisuudesta ja parannuksista perinteiseen ohjelmointiin verrattuna. Arkkitehtuurikuvauksen jälkeen käydään läpi arkkitehtuurin huonoja ha hyviä puolia kappaleessa 5. Tutkimuksen päättää lyhyt yhteenvetokappale.

2 MICROSOFT .NET

Tässä kappaleessa esitellään yleisesti Microsoft .NET ja kerrotaan mikä se on sekä mikä sen tarkoitus on. Luvussa kerrotaan myös arkkitehtuurin historiasta ja sen nykytilasta.

.NET on Microsoftin keväällä 2002 julkaisema uusi ohjelmistoarkkitehtuuri. .NET kehys eli framework on monikielinen ympäristö, jossa ohjelmointikielen voi valita lähes mielensä mukaan tuetuista kielistä, joita löytyy tällä hetkellä lähes 50. Tuettuja kieliä ovat tutumpien C++, C# ja Pascalin lisäksi myös hieman harvinaisemmat kielet, kuten Smalltalk, Scala, Lua, Lisp sekä G# [21]. Kaikki .NETin tukemat kielet on listattu liitteessä 1.

.NETin pääasiallinen tarkoitus on tehdä ohjelmistojen kehitystyöstä helpompaa, nopeampaa ja tuottavampaa [9]. Tarkoitus on myös saada ohjelmistojen hajauttaminen toimimaan verkon yli hyödyntäen ns. tilattomia palveluita sekä mahdollistaa eri alustoilla toimivien sovellusten välisen kommunikoinnin. Microsoftin pääjohtaja Bill Gates onkin todennut, että .NETin kaltaisia suuren luokan uudistuksia tulee vain 5-6 vuoden välein. Edelliset suuret muutokset olivat siirtyminen DOSista Windowsiin 90-luvun alussa ja seuraavaksi siirryttiin 16 bitistä 32 bittisyyteen kun Windows 3.1 vaihtui Windows 95:een puolessa välissä 90-lukua [2]. Käytännössä Microsoft haluaa myös kiinnittää ohjelmistojenkehittäjien huomion yhä enemmän Windowsiin ja .NET kilpaileekin vahvasti mm. Javan ja Corban kanssa kehittäjien huomiosta [4].

.NET takaa myös paremman tietoturvan ja mahdollistaa uudenlaisten Web-palveluiden (Web Services) toteuttamisen. Paremman turvallisuuden takaa hallittu koodi. .NETistä löytyy myös mm. turvallisuusmalli, joka mahdollistaa koodin lataamisen internetistä ja käyttämisen ilman nykyisiä virus- ja mato-ongelmia. Useimmat .NETin parannuksista perinteiseen Windows

ohjelmointiin verrattuna perustuvat .NETin ajoympäristöön CLR:ään. CLR:stä ja muusta .NET kehyksen rakenteesta kerrotaan lisää luvussa 4.

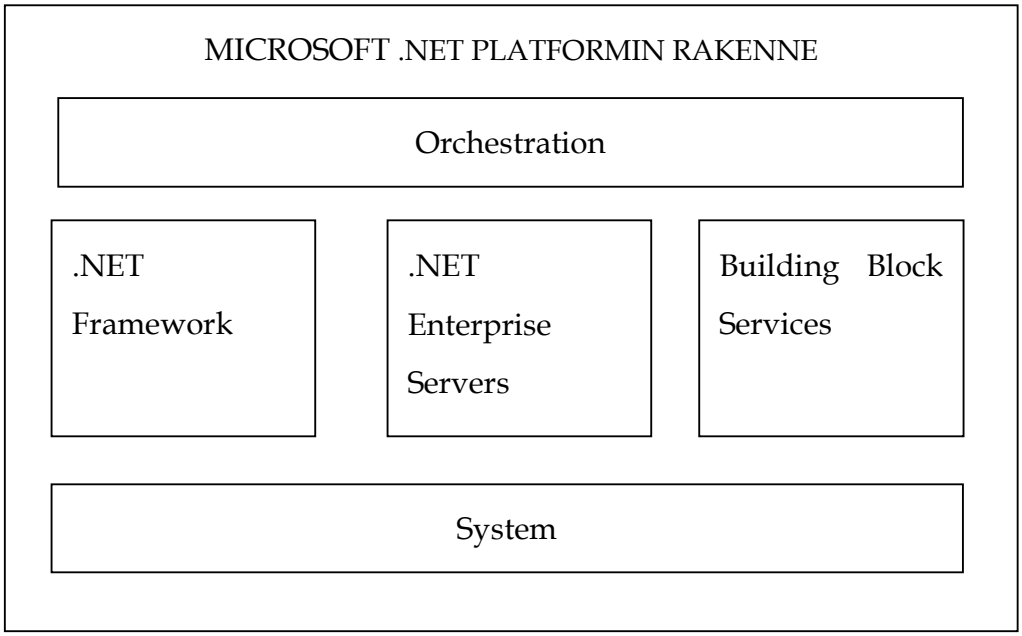
Vaikka tällä hetkellä .NET toimiikin parhaiten Windowsin alla, on arkkitehtuuri suunniteltu siten, että .NET sovellukset on mahdollista suorittaa myös muilla alustoilla, jos ajoympäristö on toteutettu kyseiselle alustalle. CLR ja C# ovat ECMA Internationalin (European Computer Manufacturing Association) standardoimia tekniikoita ja se mahdollistaa toteutusten tekemisen myös muille käyttöjärjestelmille, kuten Linuxille tai Mac OS:lle.

Viime aikoina tekniikka on ollut hieman pimennossa, mutta nyt se on saamassa jälleen uutta potkua menestykselleen. Alun perin keväällä 2002 julkaistuun Microsoft .NET framework 1.0:aan on toki jo tullut päivitys (versio 1.1), mutta se oli suureksi osaksi alkuperäisen version virheiden korjausta. Vuoden 2005 loppuun mennessä kuitenkin on odotettavissa uudistettu versio 2.0, joka tulee sisältämään paljon uusia ominaisuuksia [14]. Uuden kehyksen beta-versio 1 on ollut ladattavissa Microsoftin sivuilla kesästä 2004 lähtien ja huhtikuussa 2005 ilmestyi Microsoftin sivuille ladattavaksi beta-versio 2.

3 .NET PLATFORM

Tässä kappaleessa tarkastelemme Microsoftin .NET arkkitehtuurin alustaa (platform) ja sen tärkeimpiä osia. Alustan rakennetta käydään läpi aluksi yleisellä tasolla ja sen jälkeen keskitytään hieman tarkemmin alustan eri osa-alueisiin ja niiden tarkoitukseen.

.NET platform eli alusta on .NET arkkitehtuurin perusta, joka sisältää kaikki .NETin osa-alueet (ks. kuva 1). Alustaan kuuluu siis ympäristö sekä työkalut uuden sukupolven palveluiden toteuttamiseen ja hallintaan.



Kuva 1. Microsoft .NET platform [11]

Platformin alin kerros on käyttöjärjestelmä eli tässä tapauksessa Windows. Tuorein Windows XP on .NET-pohjainen, kuten tulee myös olemaan myös seuraava Windows, joka kulkee nimellä Longhorn. Vanhemmista Windowseista .NET-sovellukset saa toimimaan uudemmilla kuin Windows 98 eli XP:n lisäksi ainakin Windows ME, NT 4.0 ja 2000–käyttöjärjestelmillä. Tuen

vanhemmille järjestelmille saa asentamalla Microsoftin sivuilta ladattavan Microsoft .NET Framework redistributable package –paketin.

Käyttöjärjestelmän päällä on arkkitehtuurin ydin, .NET kehys eli framework. Framework jaetaan yleensä kahteen pääosaan, jotka ovat ajoympäristö ja yhdistetyt luokkakirjastot. Näiden lisäksi kehyksestä löytyvät data ja XML-luokat sekä käyttöliittymäkomponentit. .NET kehyksen pohjalla toimii ajoympäristö Common Language Runtime (CLR), joka huolehtii mm. ohjelmien kääntämisestä sekä muistinhallinnasta.

Ajoympäristön päällä ovat yhdistetyt ohjelmointikirjastot (Framework base classes), jotka sisältävät kaikille ohjelmointikielille yhteiset luokkakirjastot. Luokkakirjastojen avulla onnistuu esimerkiksi tiedon syöttö/tulostus, merkkijonojen käsittely, säikeiden hallinta jne.

Luokkakirjastojen yläpuolella ovat vielä data ja XML-luokat, joiden avulla kehittäjät pystyvät käyttämään XML-muodossa olevaa dataa tietokannoista. XML-luokat mahdollistavat XML-muotoisen datan muokkaamisen ja etsimisen. Data-luokkien avulla taas hoidetaan tietokantoihin liittyviä tehtäviä, kuten yhteyden muodostus ja hallinta, sekä tietokanta haut ja muokkaukset.

Frameworkin päällimmäisenä osana ovat mm. käyttöliittymäkomponentit, Windows Forms ja ASP.NET, joiden avulla toteutetaan perinteinen Windows-käyttöliittymä tai selainpohjainen käyttöliittymä, jonka kautta käytetään Web-palvelua. [5].

.NET Enterprise Server –perhe sisältää palvelinohjelmistoja kehittäjien käyttöön ja ne on usein tarkoitettu suurten yritysten käyttöön. Palvelimet on suunniteltu Web-palveluiden tuottamiseen ja jakeluun. Useimmat palvelimet käyttävät perustoiminnoissaan XML-kieltä, mikä takaa sen, että rakenteisista tiedoista voidaan tehdä yhdenmukaiset, vaikka sovellusten toimittajia olisi useita [7]. Tietoa voidaan myös helposti liikutella verkossa, koska palvelut tukevat XML:n

lisäksi muitakin yleisiä internet standardeja, kuten HTTP (HyperText Transfer Protocol), HTML (HyperText Markup Language) ja FTP (File Transfer Protocol).

Enterprise Server – perhe koostuu mm. seuraavista palvelimista [2]:

- Application Server on .NET alustaan integroitu komponentti, joka sisältää palvelin ja asiakaspään työkaluja, joiden avulla Web-palveluiden hallinta onnistuu helposti.

- Commerce Server on käytännössä Microsoftin Internet Information Server eli IIS. Komponentit tarkoitus on tarjota yrityksille mahdollisuus nopean e-kauppapaikan perustamiseen verkkoon.

- SQL Server tarjoaa erityisesti Web-palveluille suunnatun tehokkaan ja skaalautuvan tietokantapalvelimen. Palvelin tukee myös hyvin XML:ää ja HTTP:tä.

- Mobile Information Server on suunnattu erityisesti mobiililaitteille. Palvelin toimii sovelluspalvelimen lailla, mutta sitä käytetään mobiililaitteilla. Palvelin mahdollistaa esimerkiksi sähköpostien ja kalenterin lukemisen matkapuhelimella.

Building Block Servicet ovat ”rakennuspalikoita”, joita voidaan käyttää apuna, kun luodaan Windows- ja web-sovelluksia. Palikoihin perustuu esimerkiksi Microsoftin kehittämä HailStorm-projekti, jonka ideana on se, että käyttäjä tallentaa tietonsa verkossa olevaan ”globaaliin passiin” ja pääsee niihin käsiksi myös muualta kuin omalta koneeltaan [24]. Palvelussa käyttäjän ei tarvitse myöskään tunnistautua kuin kerran istunnossa. Käytännössä tämä tarkoittaa sitä, että yhdellä kirjautumisella toimii myös kaikki muut www-sivut ja palvelut, jotka vain tukevat passia. Käytännön esimerkki HailStormista on Microsoftin .NET Passport, joka on käytössä esimerkiksi Microsoftin internet-

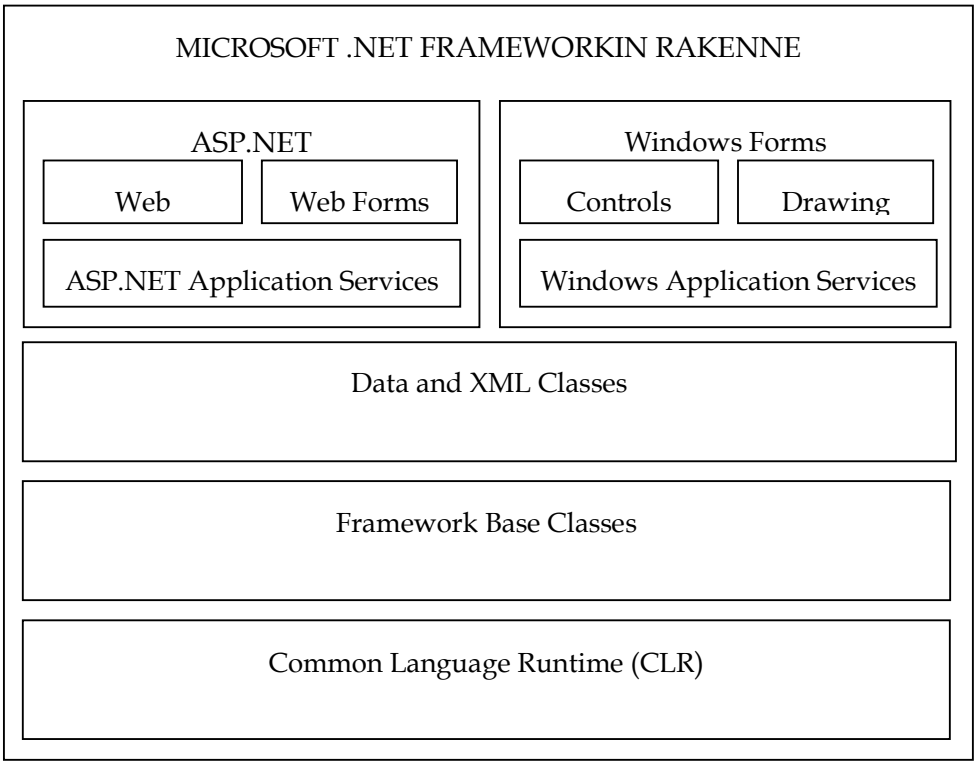
selain pohjaisessa ja ilmaisessa sähköpostipalvelussa Hotmailissa (www.hotmail.com).

Mainittujen osien lisäksi .NET platformiin kuuluu lisäksi ”ali-komponentteja”, kuten Visual Studio .NET [18]. Visual Studio .NET on .NET sovellusten toteuttamiseen tarkoitettu ohjelmointityökalu, jossa on mahdollista käyttää myös muiden kuin Microsoftin toteuttamia ohjelmointikieliä, jotka vain on integroitu .NETiin [2]. Visual Studio on myös vahvasti kiinni Microsoftin Internet Information Serverissä, jolloin Visual Studiolla on helppo toteuttaa Web-palveluita perinteisten Windows ohjelmien lisäksi.

4 .NET FRAMEWORK

Tässä luvussa käsitellään .NET arkkitehtuurin ydintä eli .NET frameworkia. Aluksi framework käydään yleisesti läpi ja sen jälkeen tarkastelemme sen eri osia tarkemmin omissa alaluvuissaan.

.NET kehys eli framework –ympäristö sisältää kaikki .NET-sovelluskehitykseen, julkaisemiseen ja ohjelman suorittamiseen tarvittavat osa-alueet [5]. Kuten edellisessä kappaleessa todettiin, voidaan framework jakaa kahteen pääosaan: ajoympäristöön ja yhdistettyihin luokkakirjastoihin. Näiden lisäksi framework jaetaan data ja XML-luokat sekä käyttöliittymäkomponentit sisältäviin kerrokseen (ks. kuva 2). Kaikissa lähteissä data ja XML-luokkia ei ole sisällytetty .NET kehykseen, vaan ne on sisällytetty samaan kerrokseen luokkakirjastojen kanssa. Noudatan tässä tutkielmassa kuitenkin Microsoftin alkuperäistä arkkitehtuurikuvausta, josta kyseinen kerros löytyy.



Kuva 2. Microsoftin .NET Frameworkin rakenne [15].

Framework – ympäristöön on koottu monia komponentteja, jotka aikaisemmin olivat ohjelmointikielikohtaisia. Tällä on pyritty mahdollistamaan monikielisyys, jolloin ohjelman funktioita ei kutsuta kielen omasta kirjastosta, vaan yhteisestä luokkakirjastosta. Tällöin ohjelmistokielten oma osuus rajoittuu lähinnä kunkin kielen syntaksiin, kuten esimerkiksi sijoituslauseen ja for-silmukkaan. Seuraavissa alakappaleissa keskitytään .NET kehyksen eri osien käsittelyyn.

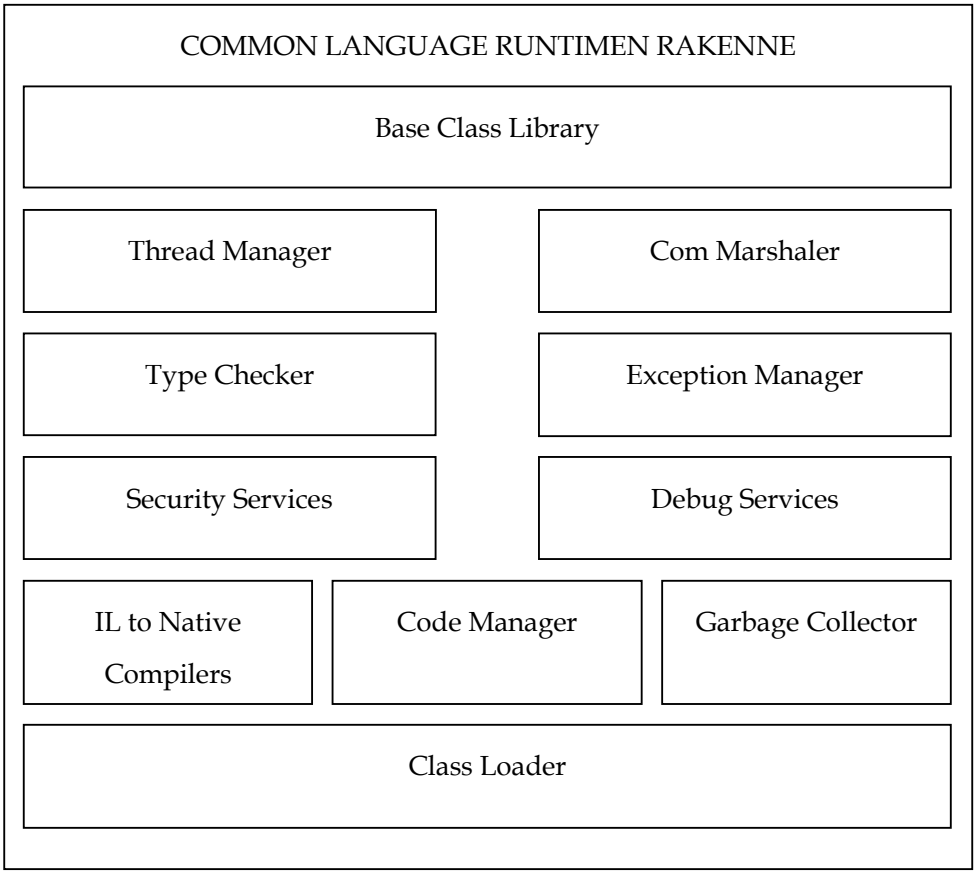
4.1 Common Language Runtime (CLR)

Common Language Runtime (CLR) on .NET kehyksen alin ja tärkein osa. CLR:n tarkempi rakenne on määritelty kuvassa 3. CLR on Microsoftin toteuttama virtuaalinen suoritusympäristö, jossa .NET-sovelluksia ajetaan. Ajoympäristö on toteutettu Ecma Internationalin standardoiman CLI:n mukaisesti [22]. Ecman standardi (Common Language Infrastructure) on saatavilla internetistä osoitteesta http://www.ecma-International.org/publications/standards/Ecma-335.htm.

Ajoympäristön tärkein tehtävä on tarjota tuki Intermediate Language – välikielen (IL) suorittamiseen ja mahdollistaa näin eri ohjelmointikielillä toteutettujen ohjelmistokomponenttien välinen yhteistyö ja .NET-sovellusten kääntäminen. IL-välikielestä voidaan käyttää myös termiä Common Intermediate Language (CIL) tai Microsoft Intermediate Language (MSIL). Käytän tässä tutkielmassa termiä CIL, koska se on määritelty ECMAn standardissa.

CLR kehitettiin myös monta muuta tavoitetta mielessä. Tarkoituksena oli esimerkiksi ohjelmoinnin yksinkertaistaminen, luoda tekniikka helpomman ja turvallisemman ohjelmistokehityksen aikaansaamiseksi sekä mahdollistaa Web-palveluiden toteuttamisen. CLR:stä haluttiin saada myös konvergenssipiste erilaisille tekniikoille ja mahdollistaa kielen integrointi tavalla huomattavasti aikaisempaa tehokkaammin [8].

CLR vastaa hieman Javan virtuaalikonetta, JVM:ää (Java Virtual Machine), sillä se toimii virtuaalikonetta vastaavalla tavalla, eli kääntää CIL-välikielisen koodin suoritettavaan muotoon. Erona Javan virtuaalikoneeseen on se, että JVM toimii tulkkina koodin ja prosessorin välillä, kun CLR kääntää välikielen aina natiivikoodiksi ennen suoritusta [5]. Ajoympäristön luvataan myös osaavan optimoida koodi eri prosessorityypeille. Koodin suorittamisen lisäksi CLR tarjoaa .NET-sovelluksille myös muita tärkeitä palveluita, joita ovat mm. roskienkeruu, debuggaus ja koodin turvallinen suoritus.



Kuva 3. Ajoympäristön rakenne [22].

Roskienkeruu on Javasta tuttu ominaisuus, joka tarkoittaa sitä, että muistista poistetaan automaattisesti oliot, joihin ei enää viitata. Roskienkeruu ja muistinhallinta vähentävät ohjelmoijan tekemiä virheitä muistin varaamisesta

ja vapauttamisessa ja tekevät ohjelmista näin hieman turvallisempia. CLR:stä löytyy myös ns. Application memory isolation-toiminto, joka estää sovellusta pääsemästä varatuille muistialueille ja aiheuttamaan näin toimintahäiriöitä [5].

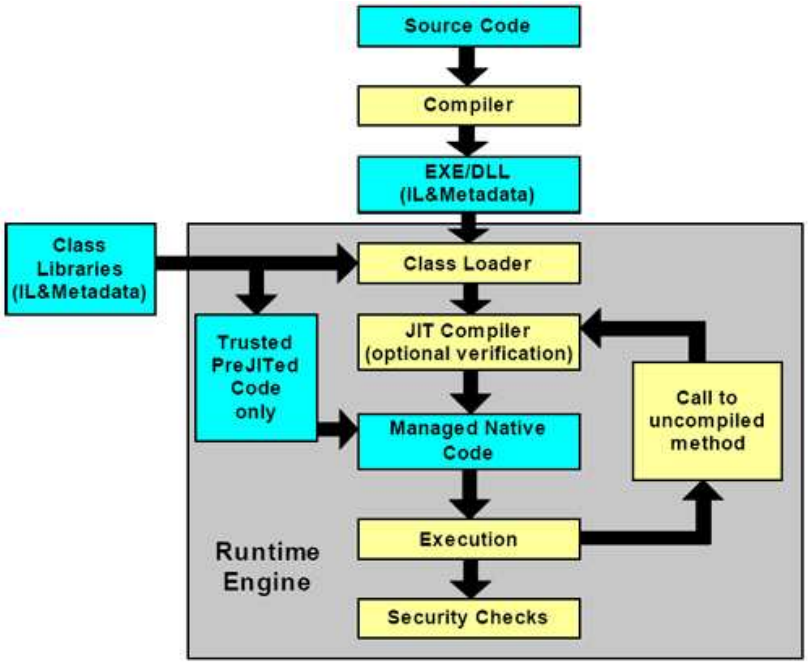
Ajoympäristön vastuulla ovat sovellusten suorittamisen lisäksi muistin varaus, prosessien ja säikeiden hallinta, turvallisuudesta ja resursseista huolehtiminen sekä komponenttien väliset riippuvuudet ohjelmaa ajattaessa. Lähes jokaisessa kielessä on toki CLR:ää vastaava ajoympäristö, mutta .NETissä poikkeuksellista on se, että siinä on yhdistetty ympäristö monille eri kielille. Jollain tuetulla kielellä kirjoitettu lähdekoodi käännetään ensin kielikohtaisella kääntäjällä CIL-välikielimuotoon ja vasta CLR ajoympäristössä suoritetaan CIL-kielelle esikäännetty koodi.

.NET-arkkitehtuurin ohjelmat jaetaan niin kutsutuissa *assemblyissa,* joka vastaa suurin piirtein Javan jar-pakettia [3]. Kukin assembly sisältää metadataa, kuvauksen assemblyyn sisällytetyistä tyypeistä sekä ajettavaa CIL-koodia standardin mukaisesti pakattuna. Ajoympäristö vastaa assemblyjen lataamisesta sekä näissä olevan koodin suorittamisesta hallitusti. .NET-arkkitehtuuriin tarkoitettua koodia kutsutaankin hallituksi koodiksi (managed code).

Hallitun koodin suorittaminen perustuu siihen, että CIL-välikielimuodossa oleva koodi ladataan aluksi ajoympäristön muistiin. Sitä mukaa kun metodeja kutsutaan, käännetään ne natiiviksi konekieliseksi binäärikoodiksi, joka sitten suoritetaan. Ajoympäristön Runtime engine tutkii CIL-välikieltä, jolloin sillä on paremmat virhe- ja tyyppitarkistusmahdollisuudet ja vasta JIT (Just-In-Time) – kääntäjä kääntää koodin juuri ennen suoritusta konekieliseksi (ks. kuva 4) [7].

Hallittua koodia saatetaan syyttää hitaammasta suorituksesta verrattuna tavalliseen binäärikoodin suorittamiseen, mutta todellisuudessa kerran käännetty koodi säilyy suorituksen ajan muistissa. Käytännössä siis JIT-käännös täytyy tehdä vain kerran kutakin metodia kohden. Varsinainen koodin suoritus

ei siis ole juurikaan hitaampaa kuin perinteisen sovelluksen suorittaminen. JIT-kääntäjän sijasta voidaan käyttää myös Native Code Generator-kääntäjää (NGEN), jossa .NET-assembly voidaan kääntää jo käännösaikana natiiviksi koodiksi, jolloin suoritustehokkuus paranee.



Kuva 4. Ajoympäristön Runtime Engine [7].

Koska .NET on avoin standardi, on arkkitehtuuri mahdollista toteuttaa myös muille alustoille kuin Windowsille. Seuraavaksi esitellään lyhyesti eri toteutukset ja niiden erot toisiinsa nähden.

4.1.1 Microsoftin kaupallinen toteutus

Microsoftin kaupallinen .NET-toteutus on tällä hetkellä laajin, eniten käytetty ja parhaiten tunnettu toteutus [14]. Toteutus toimii luonnollisesti vain Windowsin päällä ja vaatii toimiakseen Windows 98-käyttöjärjestelmän tai sitä uudemman Windowsin. Toteutuksen suurin etu verrattuna toisiin .NET-toteutuksiin on erittäin laaja luokkakirjasto. Toteutuksen luokkakirjasto sisältää esimerkiksi

Windowsin käyttöliittymäohjelmointia varten Windows.Forms -nimiavaruuden sekä ASP.NET -nimiavaruuden, jonka avulla on mahdollista toteuttaa Web-palveluita.

4.1.2 Rotor (SSCLI)

Rotor, viralliselta nimeltään Shared Source Common Language Infrastructure (SSCLI), on Microsoftin tutkimuskeskuksen (http://research.microsoft.com) kehittämä avoimen lähdekoodin ajoympäristötoteutus [14]. Tällä hetkellä Rotor toimii Windows XP:n lisäksi FreeBSD- ja Mac OS X -käyttöjärjestelmissä. Rotor on siis ainoa aidosti käyttöjärjestelmäriippumaton .NET-toteutus. Käyttöjärjestelmäriippumattomuus on saavutettu erityisellä PAL (Platform Abstraction Layer) – kerroksella.

Koska Rotorin lähdekoodi on vapaasti saatavilla, on sen avulla mahdollista tutustua .NET arkkitehtuurin syvemmin. Esimerkiksi roskienkeruun tai JIT-kääntäjän toteutuksia on mahdollista tutkia lähdekoodin avulla.

Varsinaiseen käyttöön Rotorista ei ainakaan toistaiseksi ole, sillä se ei tarjoa samanlaisia luokkakirjaston palveluja kuin Microsoftin kaupallinen toteutus. Esimerkiksi Windows Forms -käyttöliittymäkomponentit, ADO.NET -tietokantaliittymät sekä web-ohjelmointiin kehitetty ASP.NET puuttuvat Rotorista kokonaan.

4.1.3 Mono-project

Mono-projekti on avoimeen lähdekoodiin perustuva CLI-toteutus joka toimii useissa käyttöjärjestelmissä [17]. Tällä hetkellä tuetut käyttöjärjestelmät ovat Linux, Windows, Mac OS X, BSD sekä Netware. Viimeaikoina Monon viimeistelyä on vauhdittanut Microsoftin kilpakumppani Novell. Yrityksen tarkoitus on tarjota Linux-pohjaisia, mutta Windows-yhteensopivia tuotteita. Monon ASP.NET -tuki mahdollistaakin Web-palveluiden tuottamisen puoli-

ilmaisilla Linuxeilla. Microsoft suhtautuu Monoon kuitenkin kohtalaisen positiivisesti, sillä Mono on toistaiseksi Rotorin lisäksi ainoa konkreettinen todiste .NET-ytimen siirrettävyydestä.

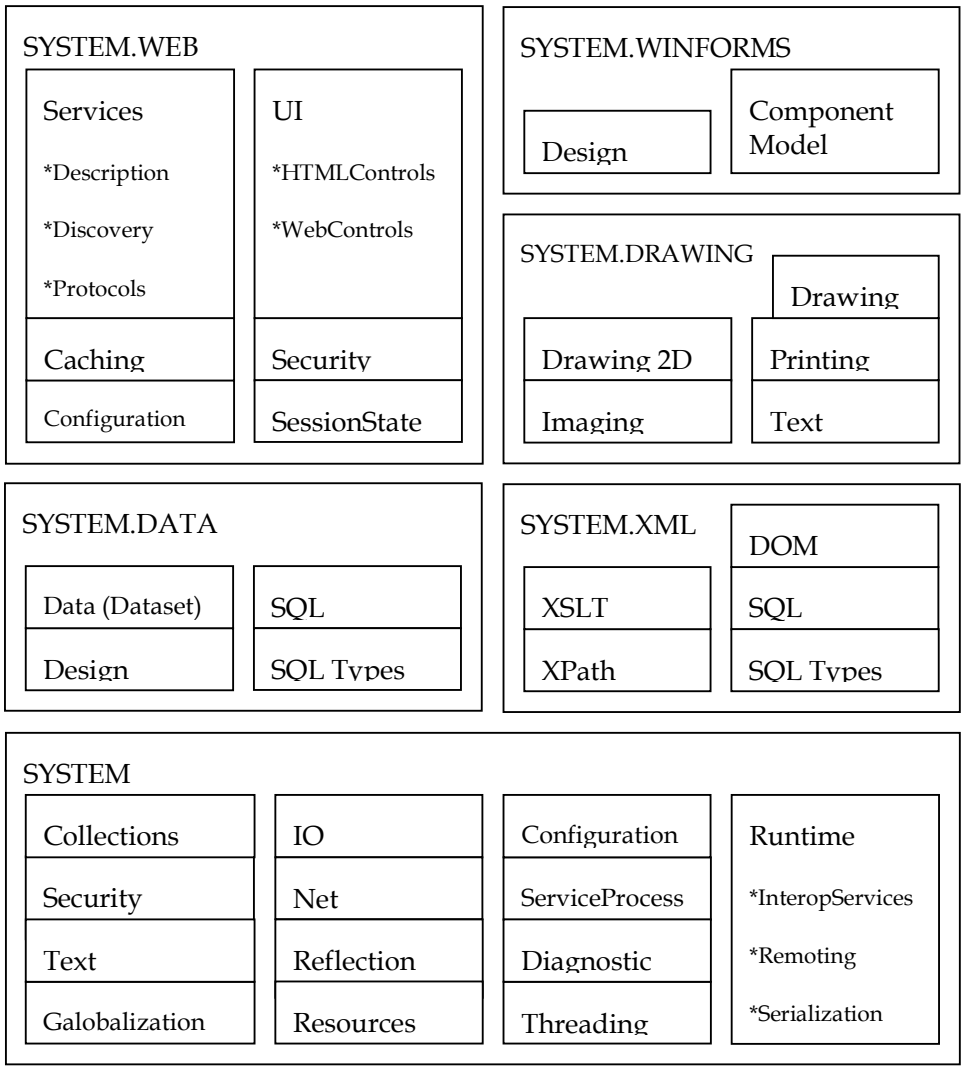
4.2 Yleiset luokkakirjastot

.NETin yleiset luokkakirjastot, Framework base classes, sisältävät valtavan määrän yleisiä ja kaikkien kielten käytettävissä olevia luokkia ja metodeja. Suuri osa aikaisemmin käytetyistä ohjelmointikielten ominaisuuksista löytyy nyt yleisistä luokkakirjastoista [2]. Luokkakirjastot sisältävät sekä abstrakteja perittäviä luokkia sekä luokkia, joista voi luoda ilmentymän eli olion.

Yleisten luokkakirjastojen yksi suurimmista eduista on se, että sovellusten lähdekoodit ovat huomattavasti aikaisempaa lyhyempiä. Tutkimuksen mukaan saman esimerkkiohjelman tekeminen J2EE:llä vaati 14 004 riviä koodia, kun .NET-versioon tarvittiin vain 2096 riviä koodia [16]. Tämä 85 % pienempi koodin määrä tarkoittaa luonnollisesti sitä, että sovellus on valmis paljon J2EE-sovellusta aikaisemmin. Myös itse tuote on turvallisempi ja vakaampi, koska virheiden määrä kasvaa rivimäärän kasvaessa.

.NET kehyksen luokat on jaettu nimiavaruuksiin (namespaces), koska olemassa on useita satoja luokkia. Nimiavaruus tarkoittaa samaan nimettyyn ryhmään kuuluvia luokkia. Nimiavaruuden juuriluokka on System –nimiavaruus, ja kaikki luokat periytyvät Object -pääluokasta.

Standardiluokkakirjastot (alin luokka kuvassa 5) tarjoavat perustoimintoja, kuten esimerkiksi I/O-toiminnallisuus, säikeiden hallinta, tekstin hallinta jne. Luokat vastaavat sellaisia vanhempia luokkia, kuten esimerkiksi STL, MFC, ATL ja Javan perusluokat.



Kuva 5. .NETin nimiavaruus [22].

Perusluokkien päällä ovat data ja XML-luokat, jotka perusluokkien avulla huolehtivat tiedon hallitsemisesta ja XML-muokkauksesta. Ylimpänä .NET frameworkissa on kuvattu varsinaisten sovellusten tekemiseen tarvittavia luokkia [7].

Fyysisesti nimiluokat sijaitsevat useassa erillisessä DLL-tiedostossa. Vaikka funktio tai objekti on osa System – nimiavaruutta, ei se kuitenkaan vielä takaa, että kääntäjä automaattisesti löytää sen [19]. Kehitystyökalussa pitääkin erikseen kertoa, mitkä nimiavaruudet käännökseen otetaan mukaan

riippumatta käytetystä kielestä. Nimiavaruuksiin voidaan myös viitata, jolloin nimiavaruudesta voidaan käyttää aliasta eikä aina tarvitse kirjoittaa koko nimiavaruuden osoitetta. Esimerkiksi tyypillinen Visual Basic .NET lomake saattaa sisältää seuraavat rivit [2]:

```
Imports System.WinForms
Imports Debugging = System.Diagnostics.Debug
```

Ensimmäinen rivi määrittelee yksinkertaisesti vain WinForm-nimiavaruuden metodit ja ominaisuudet käytettäväksi koodissa. Toisella rivillä luodaan alias, jonka ansiosta koodissa ei tarvitse viitata System.Diagnostic.Debug olioon, vaan voidaan käyttää aliasta ja viitata pelkästään Debugging -olioon.

Seuraavassa käydään läpi joitain tärkeimpiä .NET-arkkitehtuurin luokkakirjastoja ja niiden tarjoamia palveluita [2]. Kattavin luokkakirjasto löytyy tällä hetkellä kaupallisesta Microsoftin .NET toteutuksesta.

- *System*-nimiavaruus on nimiavaruuden juuriluokka ja se tarjoaa yleisimmin käytetyt tyypit kuten *Object* (kaikkien luokkien kantaluokka), *Exception* (kaikkien poikkeusten kantaluokka) ja *EventData* (Tapahtumien argumenttien kantaluokka)

- *System.Collections*-nimiavaruus tarjoaa luokkia ja rajapintoja joiden avulla toteutetaan yleisimmät tietorakenteet ja kokoelmat kuten taulukot, jonot ja listat.

- *System.Data*-nimiavaruus sisältää luokkia, joiden avulla on mahdollista käsitellä tietokannoissa olevaa tietoa. Nimiavaruus sisältää luonnollisesti luokat myös tietokantayhteyden muodostusta ja hallintaa varten.

- *System.Diagnostics*-nimiavaruus tarjoaa luokat koodin seuraamiseen sekä debuggaukseen.

- *System.IO*-nimiavaruus tarjoaa tyyppejä ja luokkia joiden avulla tietoa voidaan kirjoittaa ja lukea virroista ja tiedostoista.
- *System.Math*-nimiavaruus tarjoaa matemaattisia laskutoimituksia, kuten neliöjuurta ja logaritmi-funktioita.
- *System.Reflection*-nimiavaruus sisältää luokat, joiden avulla on mahdollista tutkia metadataa.
- *System.Security*-nimiavaruus tarjoaa sovelluskehittäjille pääsyn .NET-arkkitehtuurin turvallisuusominaisuuksiin. Luokan avulla on mahdollista toteuttaa mm. salausta ja pääsylupia.
- *System.Threading*-nimiavaruus tarjoaa luokat ja rajapinnat, jotka mahdollistavat säikeiden käytön ohjelmissa.
- *System.Web*-nimiavaruus mahdollistaa selaimen ja palvelimen välisen kommunikoinnin.
- *System.Web.Services*-nimiavaruus tarjoaa luokat joiden avulla voidaan toteuttaa Web-palveluita.
- *System.Web.UI*-nimiavaruus mahdollistaa Windowsin käyttöliittymäkonseptia muistuttavan tekniikan www-pohjaisten sovellusten käyttöliittymien kehittämiseen.
- *System.Windows.Forms*-nimiavaruus tarjoaa Windows API-tyylisen rajapinnan käyttöliittymien kehittämiseen Windows-käyttöjärjestelmissä.
- *System.XML*-nimiavaruus tarjoaa standardinmukaisen tavan käsitellä XML-muotoista dataa.

4.3 Data- ja XML-luokat

Data ja XML-luokat sisältävät nimensä mukaisesti datan ja XML:n käsittelyyn tarvittavia palveluita. Datalla tarkoitetaan tässä tapauksessa tietokantoja ja kerros sisältääkin monipuoliset palvelut tietokantojen käyttöön ja muokkaamiseen. Tietokannat ja XML ovat myös tiukasti kiinni toisissaan .NET-arkkitehtuurissa, mistä kertoo esimerkiksi se, että .NETissä tietokannat voidaan esittää XML-muodossa tai toisinpäin. Myös normaalin SQL-haun tulokset voidaan esittää XML dokumenttina seuraavalla tavalla [2]:

SELECT etunimi, sukunimi FROM tyontekija FOR XML AUTO

Yllä oleva SQL-lause palauttaa tulokset XML-muodossa seuraavasti:

```
<?xml version="1.0"?>
  <root>
    <row etunimi=”Matti” sukunimi=”Meikäläinen” />
    <row etunimi=”Maija” sukunimi=”Meikäläinen” />
    <row etunimi=”Pekka” sukunimi=”Meikäläinen” />
  </root>
```

Seuraavissa alakappaleissa on kerrottu tarkemmin .NET-kehyksen sisältämistä Data- ja XML-luokista.

4.3.1 Data-luokat

.NET-arkkitehtuurin data-luokat sisältävät siis komponentteja, joiden avulla sovelluksen ja tietolähteen välinen kommunikointi voidaan hoitaa helposti ja tehokkaasti. Perinteinen Microsoftin tietokantakomponentti ADO (ActiveX Data Objects) on mukana, jotta .NET olisi yhteensopiva vanhempien tietokantasovellusten kanssa. Mukana on otettu myös ADOn uusin versio eli ADO.NET. ADO.NET on .NET arkkitehtuuria varten kehitetty ohjelmointirajapinta tietokannan käyttämistä varten. Se on toteutettu Web-palveluita silmälläpitäen ja ADO.NETin suunnittelussa onkin kiinnitetty erityisesti huomioita mm. skaalautuvuuteen ja XML:ään [13]. ADO.NETin

sanotaan tekevän vähintäänkin kaikki samat asiat kuin perinteinenkin ADO, mutta paremmin [23][2].

ADO.NET on itse asiassa joukko kirjastoja, joiden avulla .NET-sovellukset voivat keskustella tietokantojen kanssa. Kirjastot sisältävät luokkia, joiden avulla luodaan tietokantayhteyksiä, suoritetaan SQL-lauseita ja prosessoidaan hakutuloksia. ADO.NETiä voidaan käyttää myös yhteydettömänä datavarastona, jolloin dataa voidaan käsitellä, vaikka yhteyttä varsinaiseen tietokantaan ei olisikaan käytössä sillä hetkellä. Vaikka ADO.NET käyttää toiminnassaan paljon vanhan ADOn komponentteja, löytyy siitä myös joitain uusia, kuten DataAdapter ja DataSet.

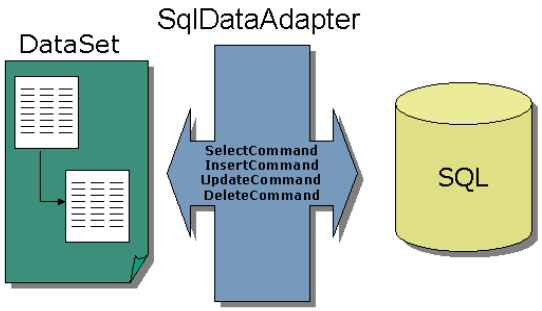
DataSet on ADO.NETin keskeinen komponentti, joka mahdollistaa mm. tietokannasta hakemisen, järjestämisen, suodatuksen ja siellä navigoinnin. DataSet on ikään kuin yhteydetön tietokanta tai tietovarasto, jonka tietokantamainen rakenne koostuu esimerkiksi tauluista, riveistä, sarakkeista ja suhteista. Se parantaa sovellusten tehokkuutta, sillä enää ei yksitellen haeta tietokantapalvelimelta dataa taulu kerrallaan, vaan kaikki data talletetaan DataSet komponentille ja tieto palautetaan yhdellä pyynnöllä palvelimelta. DataSet toimii oikeaa tietokantaa vastaavalla tavalla, jolloin sen avulla voidaan hakemisen lisäksi myös muokata tietoa. Kaikki muokkaukset talletetaan DataSet -komponentille ja myöhemmin muutokset päivitetään oikeaan tietolähteeseen. Käytännössä päivitys toimii siten, että DataSetin GetChanges-metodi luo uuden DataSet komponentin, joka sisältää vain päivitetyn datan. DataAdapter komponentti päivittää sitten uudesta DataSetistä tuoreet tiedot varsinaiseen tiedonlähteeseen. Muutokset voidaan luonnollisesti päivittää vasta kun käyttäjä itse niin haluaa [13] [6].

DataSet osaa myös lukea ja kirjoittaa XML-muotoista dataa. Komponentti käyttää apunaan XmlReader ja XmlWriter –luokkia, joiden avulla se pystyy mm. muuttamaan tietokannan rakenteen XML skeemaksi [15]. Tietokannan

rakenteen lisäksi komponentti osaa kirjoittaa myös vaikka koko tietokannan sisällön XML-muotoon ja se voidaan tallentaa tietokoneen muistin lisäksi talteen tiedostoon. Vastaavasti uusia tietokantoja voidaan luoda DataSet komponenttiin lukemalla XML-muotoinen tietokannan rakenne ja sisältö tiedostosta. DataSet ei koskaan kommunikoi suoraan tietokannan tai muun tietolähteen kanssa, eikä se siten ole riippuvainen tiedon lähteestä. Tämän ansiosta kehittäjä voi luoda mallin, jossa tietokannan sisältö tulee DataSet komponentille suoraan tietokannasta, XML-tiedostosta tai vaikka käyttäjän syötteestä [6].

Käytännössä relaatiodata ja XML ovat muutenkin kokeneet .NET kehyksessä vahvan integraation, sillä ADO.NET ja XML-luokat on yhdistetty samaan kerrokseen. Uudistusten ansiosta ADO.NET –luokka tarjoaa kehittäjille mahdollisuuden kommunikoida tietokannan kanssa XML-muodossa tietokantarajapintojen kautta, joita ovat esimerkiksi OLE DB, ODBC, Oracle ja SQL Server –rajapinnat [15].

ADO.NETin toinen uusi ydinkomponentti on DataAdapter. Sen tarkoitus on toimia siltana tietokannan ja yhteydettömän ADO.NET-komponentin välissä ja hoitaa näiden välinen kommunikointi (ks. kuva 6). DataAdapter tarjoaa tehokkaan tavan hakea tietoa tietokannoista esimerkiksi DataSet tai DataTable –komponenteille sekä päivittää muokattu tieto takaisin tietokantaan.



Kuva 6. DataAdapter toimii siltana DataSetin ja tietokannan välissä [13].

DataAdapterin ansiosta tietokantojen käytön tehokkuus on kasvanut erityisesti kun tietokanta pyörii Microsoft SQL-serverin päällä, jolloin voidaan käyttää eri menetelmiä kuin muiden valmistajien kantojen kanssa. Kuitenkin yhteistyön muidenkin valmistajien tietokantapalvelimien kanssa luvataan olevan entistä tehokkaampaa ja nopeampaa. DataAdapter sisältää joukon metodeita, joiden avulla tietoa haetaan, tuhotaan ja päivitetään, mutta niihin ei tässä tutkimuksessa syvennytä sen tarkemmin [15] [6].

4.3.2 XML-luokat

XML eli eXtensible Markup Language on SGML:stä (Standard Generalized Markup Language) kehitetty yksinkertaisempi versio. XML on tekstipohjainen metakieli, jonka avulla kuvataan tiedon rakennetta [1]. .NET-arkkitehtuurissa XML:n avulla voidaan kuvata tietokantojen rakenne sekä tietokannan sisältö. Tietokannan tiedot voidaan lukea suoraan tietokannasta tai vaikka tiedostosta..

.NET-arkkitehtuurin XML-luokat tarjoavat nimensä mukaisesti XML-muodossa olevan tiedon käsittelemiseen käytettäviä palveluita, kuten tiedon muokkaamista ja hakemista. Kuten edellisessä luvussa todettiin, tarjoaa XML-luokat kehittäjille mahdollisuuden käyttää XML-muodossa olevaa tietokantadataa jonkin tietokantakäyttöliittymän kautta. XML on myös yksi .NETin ydintekniikoista, sillä kaikki .NET kehyksen osat (Web-palvelut, ASP.NET jne.) käyttävät datan esitysmuotonaan juuri XML:ää. XML muotoisen tiedon käsittelemistä varten kerrokselta löytyy mm. XML-parseri ja XSL (eXtensible Stylesheet Language) transformeri [2].

.NETin XML-luokat sisältävät XML:n uuden version eli MSXML 3.0:n, joka on räätälöity erityisesti .NET-arkkitehtuuria varten. Uusi versio sisältää uusia ominaisuuksia, kuten laajemmat API-rajapinnat, helpomman ohjelmointimallin ja se noudattelee myös standardia paremmin. Myös tehokkuuden luvataan olevan vähintäänkin yhtä hyvä kuin aikaisemmin [15].

.NET arkkitehtuurille tyypillisesti XML-luokat on jaettu useaan nimiavaruuteen. Ydinluokat löytyvät System.Xml –nimiavaruudesta sekä System.Xml.Serialization, System.Xml.Xpath ja System.Xml.Xsl -nimiavaruuksista. Keskeisimmät nimiavaruudet sijaitsevat fyysisesti System.Xml.dll –tiedostossa, joka ohjelmoijan tulee ottaa kehitysympäristössä mukaan, jos haluaa käyttää sovelluksissaan näitä luokkia.

XML-luokkien ytimestä löytyy kaksi abstraktia luokkaa: XmlReader ja XmlWriter. XmlReader on luokka, joka tarjoaa nopean ja tehokkaan tavan lukea XML-muotoista dataa. XmlWriter taas on luokka, joka tarjoaa rajapinnan standardin mukaisen XML datan tuottamiseen. Sovellukset, jotka haluavat lukea tai kirjoittaa XML-dokumentteja käyttävät XmlReader ja XmlWriter –luokkia. Molemmista luokista löytyy perittyjä luokkia, jotka ovat erikoistuneet hieman eri asioihin, mutta toteuttavat kuitenkin juuriluokan vaatiman toiminnallisuudet.

Käytännössä XML-luokkia voidaan käyttää normaalin luokan tapaan. Alla olevassa koodissa esitetään kuinka XML-dokumentti generoidaan käyttämällä apuna XmlWriter – luokkaa [15].

```
public void WriteDocument(XmlWriter writer) {
  writer.WriteStartDocument();
  writer.WriteComment("generated by SampleWriter");
  writer.WriteProcessingInstruction("hack", "on person");
  writer.WriteStartElement("p", "person", "urn:person");
  writer.WriteStartElement("name", "");
  writer.WriteString("joebob");
  writer.WriteEndElement();
  writer.WriteElementInt16("age", "", 28);
  writer.WriteEndElement();
  writer.WriteEndDocument();
}
```

Yllämainitun koodin avulla saadaan seuraavanlainen XML-muotoinen dokumentti [15]:

```
<?xml version="1.0"?>
<!--sample person document-->
<?hack on person?>
<p:person xmlns:p="urn:person">
  <name>joebob</name>
  <age unit="year">28</age>
```

```
</p:person>
```

4.4 Käyttöliittymäkomponentit

.NET Frameworkin ylin kerros sisältää käyttöliittymien rakentamiseen vaadittavat käyttöliittymäkomponentit. .NETissä on kaksi tapaa rakentaa käyttöliittymä sovelluksille. Vanhaan tapaan on mahdollista rakentaa perinteinen Windows-ohjelma Windows Formsien avulla. Tämän lisäksi on mahdollista tehdä selainpohjainen käyttöliittymä, jonka kautta voidaan käyttää Web-palveluita. Käyttöliittymäkomponentit ovat yhteisten luokkakirjastojen tapaan kaikkien .NET-ohjelmointikielten käytettävissä. Kahden päätavan lisäksi on vielä mahdollisuus rakentaa täysin tekstipohjainen käyttöliittymä, mutta tässä tutkielmassa keskitytään lähinnä kahteen ensin mainittuun tapaan.

Seuraavissa kappaleissa keskitytään käyttöliittymä-kerroksen kahteen pääosaan, jotka ovat ASP.NET ja Windows Forms.

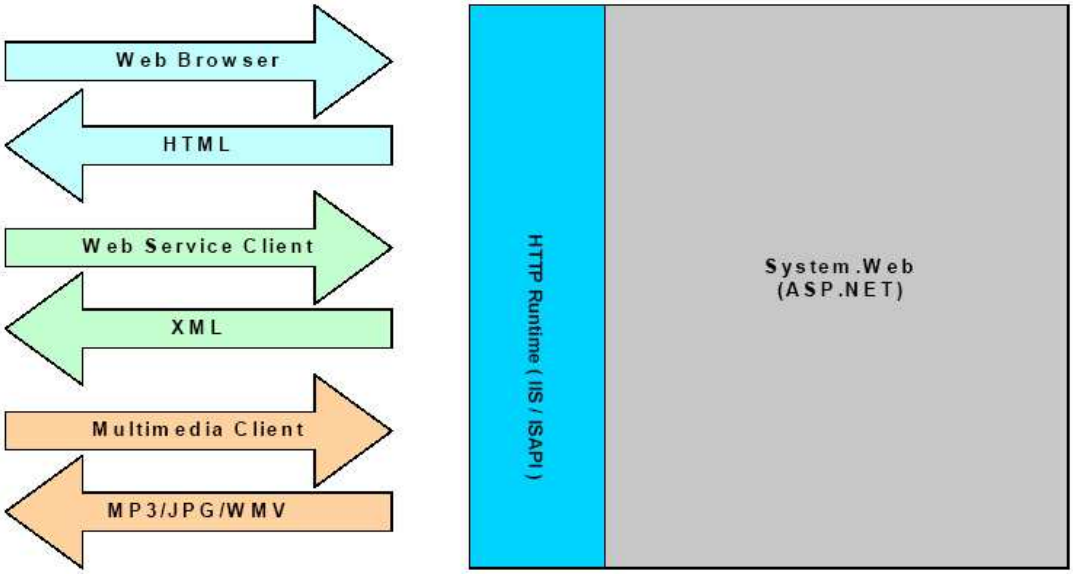
4.4.1 ASP.NET

ASP.NET on .NET arkkitehtuuria varten kehitetty ASP:hen (Active Server Pages) pohjautuva tuore ohjelma-alusta. Alkuperäinen ASP on Microsoftin dynaamisten www-sivujen tuottamiseen kehittämä teknologia ja ASP.NET on tarkoitettu enemmänkin Web-palveluiden tekemiseen. Se on lisäksi edeltäjäänsä helpompi ohjelmoida ja suorituskykykin on parantunut. ASP.NETiä kehitettiin aikoinaan nimellä ASP+ (Active Server Pages Plus), mutta myöhemmin nimi muutettiin sopivammaksi Microsoftin .NET strategialle [2].

ASP.NETin avulla voidaan luoda palvelinpäässä ajettavia sovelluksia, joista asiakas eli yleensä internet-selain näkee ainoastaan käyttöliittymän ja itse ohjelman suoritus tapahtuu piilossa. Sovellukset perustuvat standardoituihin

tekniikoihin, kuten HTTP- ja SOAP –protokollaan (Simple Object Access Protocol), sekä HTML- ja XML-kieliin. ASP.NETissä pyyntöjen käsittely ja muokkaaminen on siirretty CLR:n tehtäväksi. Tämä tarkoittaa sitä, että kun vanha ASP käytti yhtä pyyntö/vastaus-käsittelijää, niin ASP.NETissä -pyynnöt ohjataan erillisille käsittelijäobjekteille (ks. kuva 7) [7].

Yksi ASP.NETin hyödyllisimmistä uudistuksista ASP:hen verrattuna on se, että ASP.NETissä koodi ja ulkoasu on eroteltu täysin toisistaan. Perinteisessä ASP:ssä koodi ja sivun muotoilu eli HTML-tagit olivat täysin sekaisin samassa tiedostossa, mutta ASP.NETissä käytetään taustakoodi (code-behind) nimistä tekniikkaa [19]. Käytännössä tämä tarkoittaa sitä, että koodi ja HTML-muotoilu sijaitsevat eri tiedostoissa, jolloin esimerkiksi ulkoasun muuttaminen on huomattavasti aikaisempaa helpompaa. Kun ohjelmoidaan esimerkiksi Visual Studio .NETillä ASP.NET sovellusta, tulee varsinainen ohjelmakoodi oletuksena erilliseen tiedostoon ja aspx-tiedostoon tulee sivun muotoilun ja HTML-kontrollien lisäksi vain viittaus lähdekoodin sisältävään tiedostoon.



Kuva 7. ASP.NETissä on erilliset käsittelijäobjektit, jotka käyttävät IHttpHandler-rajapintaa ja näin voidaan palvella erilaisia asiakkaita [7].

ASP.NET sovelluksia kirjoitettaessa ei ohjelmoijan myöskään tarvitse osata HTML-kieltä vaan ohjelmointityökalussa lisätään lomakkeelle vain HTML-kontrollit ja HTML-koodi täydennetään automaattisesti. Esimerkkinä napin lisäämiseen tarvittava koodi aspx-sivulla:

```
<asp:Button id=Button1 runat=”server” Width=”100” Height=”24” Text=”Lähetä tiedot”></asp:Button>
```

Napin takana olevan toiminnallisuuden voi .NETille tyypillisesti koodata millä tahansa .NET-ohjelmointikielellä. Sivulla olevaa skriptiä ei myöskään tulkata ajonaikana kuten ASP:ssä, vaan ASP.NET-sivu käännetään automaattisesti .NET-luokaksi silloin kun sivut asennetaan. Käännetty luokka talletetaan cacheen, josta sitä käytetään aina kun sivua kutsutaan. Tämä luonnollisesti nostaa suorituskykyä paljon, kun sivua ei tarvitse tulkata aina kun sitä pyydetään.

Toinen ASP.NETin suuri parannus on se, että ohjelmointi on helpompaa kuin aikaisemmin. Esimerkiksi kaikki kontrollit, kuten napit, tekstikentät ja tiputusvalikot, ovat web-palvelimen hallinnassa, joten koodaajan ei tarvitse enää huolehtia niiden tilasta. Käytännössä tämä tarkoittaa sitä, että ASP:ssä tuli koodata skripti, joka huolehti kontrollin tilasta, kun käyttäjä päivitti sivua. ASP.NETissä taas palvelin huolehtii tiloista automaattisesti, jolloin koodaaja voi keskittyä itse logiikan ohjelmointiin.

Nämä ja useat muut uudistukset tekevät ASP.NETistä paremman ja helpommin ohjelmoitavan tekniikan varsinkin Web-palveluiden kehittämistä ajatellen. Yleisesti ajatellaankin, että on melkein parempi pitää ASP:tä ja ASP.NETiä kokonaan erillisinä tekniikoina, sillä ne eroavat niin paljon toisistaan.

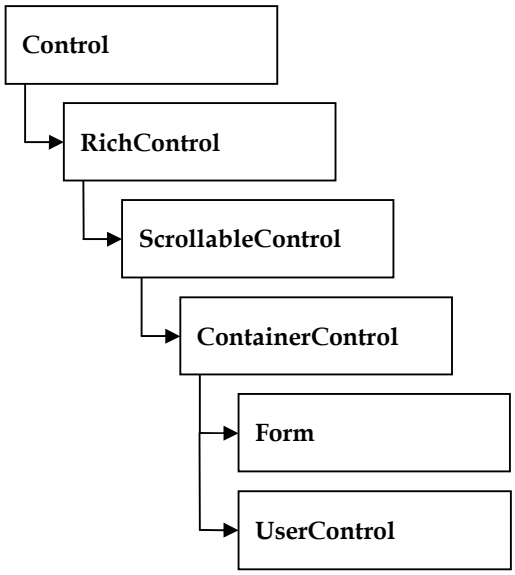
4.4.2 Windows Forms

Windows Forms mahdollistaa nimensä mukaisesti perinteisten Windows käyttöliittymien ohjelmoinnin. Käytännössä Windows Forms on .NET nimiavaruus, joka sisältää Windows käyttöliittymän ohjelmointiin tarvittavia GUI -komponentteja. Windows Forms tukee myös Web-palveluja siten, että yhteyden ottaminen palveluun on helppoa ja nopeaa [11].

.NETille tyypillisesti myös Windows Formseja voi käyttää samalla rajanpinnalla mistä tahansa .NET alustaa tukevasta kielestä. Windows Formsilla ja ASP.NETin Web Formsilla on samanlaiset suunnitteluperiaatteet, mutta luokat ja niiden toteutukset eroavat toisistaan. Vastaavilla luokilla ja toiminnoilla on kuitenkin samat nimet molemmissa nimiavaruuksissa, esimerkiksi molemmilla on luokka Button, jossa on tekstiä, onClick -tapahtuma sekä Font- ja ForeColor-ominaisuudet.

Windows Forms perustuu Windows Foundation Classes –luokkiin (WFC), jolloin esimerkiksi Visual Basicilla ohjelmoineille käyttöliittymän tekeminen on helppoa ja ennestään tuttua. Käyttöliittymän teko on muutenkin samanlaista kuin muissa ohjelmointiympäristöissä, sillä tyhjälle lomakkeella raahataan kontrolleja. Niille voidaan myös ohjelmoida tapahtumia ja metodeja, sekä muuttaa niiden ominaisuuksia.

Kaikkien Windows Forms – kontrollien juuriluokkana on System.WinForms.Control –luokka. Jokainen luokka periytyy luokan yläpuolella olevasta luokasta ja jokainen luokka lisää hieman toiminnallisuutta, kunnes saadaan toimiva lomake-luokka. Kuvassa 8 on kuvattu Windows Formsien luokkahierarkia.



Kuva 8. Wndows Forms – kontrollien periytyminen [11].

Control-luokka määrittelee perusluokan kontrolleille eli käyttöliittymän näkyville komponenteille. Control-luokka hallitsee mm. käyttäjän syöttöjä sekä ikkunan kokoa ja sijaintia. RichControl-luokka hallitsee lomakkeen näkyviä ominaisuuksia, kuten lomakkeelle piirtoa, fonttia ja taustan väriä. ScrollableControl mahdollistaa lomakkeen vierittämisen pysty- ja vaakasuunnassa. ContainerControl mahdollistaa sen, että kontrolli sisältää toisia kontrolleja sekä kontrollin kohdistamisen. Form-luokka on perus lomake-luokka, jonka avulla luodaan sovelluksissa lomakkeita. UserControl-luokan avulla luodaan kontrolleja.

Yksi parhaista Windows Formsin ominaisuuksista onkin perinnän mahdollisuus. Koska Windows Forms on vain luokka, voidaan siitä periä toisia luokkia, joista luodaan lomakkeita. Uusi lomake sisältää kaikki juuriluokan ominaisuudet ja kontrollit sekä mahdollisen koodin, joka on juuriluokasta peritty. Jos juuriluokkaan lisätään kontrolli tai sen ominaisuuksia muutetaan, muutokset näkyvät myös kaikissa perityissä lomakkeissa. Perinnän suurin etu on se, että suurimman osan työstä tarvitsee tehdä vain kerran. Kaikki

sovelluksen lomakkeet saavat myös automaattisesti esimerkiksi samanlaisen ulkoasun ja samanlaisen valikko-rakenteen [2].

Windows Forms on loppujen lopuksi melko samanlainen tapa tehdä käyttöliittymiä, kuin aikaisemmatkin menetelmät, jossa kontrolleja raahataan lomakkeella. Toki Windows Formsissa on jotain eroa aikaisempiin menetelmiin, mutta esimerkiksi ASP:n ja ASP.NETin erot ovat paljon suuremmat. Osaltaan tämän selittää se, että .NETissä on keskitytty enemmän Web-palveluiden kuin perinteisten Windows sovellusten tekemiseen.

5 .NETIN HYÖDYT JA HAITAT

Kappaleessa tutkitaan muutamia .NET-arkkitehtuurin parhaimpia ominaisuuksia. Näiden lisäksi esitellään .NET-arkkitehtuurista esitettyä kritiikkiä. Odotetusti uudet ja paljon mainostetut ominaisuudet, kuten kieliriippumattomuus saavat kehumisen lisäksi osakseen myös kritiikkiä varsinkin Microsoftin tiukan kilpailijan Sun Microsystemsin suunnalta. Osa ihmisistä on sitä mieltä, että Microsoft toi .NETin markkinoille vain vähentääkseen ohjelmistokehittäjien mielenkiintoa J2EE:tä kohtaan ja lisätäkseen sitä Windowsia kohtaan. Varmasti mielipide ei olekaan kovin kaukaa haettu, sillä J2EE ja .NET omaavat lähellä toisiaan olevat ominaisuudet ja toiminnallisuuden, molemmat teknologiat käyvät esimerkiksi tiukkaa kilpailua Web-palveluidensa paremmuudesta.

5.1 Hyödyt

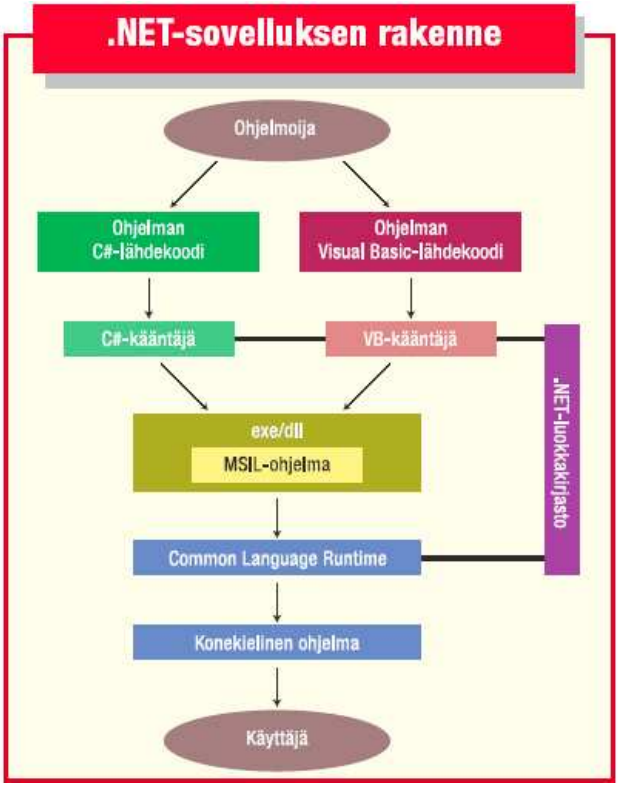
Seuraavissa alakappaleissa tutkitaan muutamaa tärkeintä ja mielenkiintoisinta .NET-arkkitehtuurin ominaisuutta. Esimerkiksi kieliriippumattomuus on kokonaan uusi ominaisuus verrattuna perinteiseen Windows-ohjelmointiin, kun taas Web-palveluita on voinut tehdä aikaisemminkin, mutta uuden ASP.NETin avulla kehitys on aikaisempaa helpompaa ja nopeampaa. Kappaleessa läpikäytäviä ominaisuuksia pidetään yleisesti tärkeimpinä asioina, joiden avulla .NET-arkkitehtuuri saattaa tulevaisuudessa lyödä itsensä läpi. Näitä ominaisuuksia ovat kieliriippumattomuus, parantunut tietoturva sekä Web-palvelut.

5.1.1 Kieliriippumattomuus

Ohjelmoijan kannalta yksi .NETin suurimmista uudistuksista on ehdottomasti kieliriippumattomuus. Käytännössä tämä tarkoittaa sitä, että koodia voidaan kirjoittaa millä tahansa .NETin tukemalla ohjelmointikielellä. Microsoftin .NET

SDK:n (Software Development Kit) mukana tulevat kääntäjät Visual Basicin .NET-versiolle, uudelle C#-kielelle sekä Microsoftin Javaa muistuttavalle J#-kielelle. Kuten aikaisemmin jo todettiin, löytyy näiden lisäksi kolmansienosapuolien toteuttamia kääntäjiä kymmenille muillekin kielille ja kokoajan tulee uusia kääntäjiä. Osa näiden kielten kääntäjistä on maksullisia, mutta osan saa myös ilmaiseksi.

Kieliriippumattomuus perustuu .NETin ajonaikaiseen ympäristöön CLR:ään. Eri ohjelmointikielillä kirjoitetut ohjelmat käännetään .NET -ympäristössä aluksi CIL –välikielimuotoon (ks. kuva 9). CIL -muodossa olevat ohjelmat näyttävät suunnilleen samalta riippumatta siitä millä kielelle ne on kirjoitettu (vertaa kuvat 10 ja 11). CLR suorittaa sitten tätä CIL-koodia ja ajaa sovelluksen pyytämät toiminnot. CIL-ohjelmat näkyvät käyttäjälle tavallisina exe-tiedostoina, joten CLR:n tekemä suoritus on käyttäjän kannalta lähes näkymätöntä, lukuun ottamatta hivenen normaalia hitaampaa suoritusta. Nykykoneiden tehoilla viive on kuitenkin käytännössä näkymätön käyttäjälle.



Kuva 9. .NETissä eri kielillä kirjoitetut lähdekoodit käännetään samaan CIL-välikielimuotoon, josta CLR kääntää ne binäärikoodiksi [9].

```
static void Main() {
  int tunti = DateTime.Now.Hour
  if (tunti > 8)
    Console.WriteLine(”Hyvää päivää”);
  else
    Console.WriteLine(”Hyvää huomenta”);
}
```

Kuva 10. C# lähdekoodi [9]

```
Sub Main()
  Dim tunti As Integer
  tunti = DateTime.Now.Hour
  If (tunti > 8)
    Console.WriteLine(”Hyvää päivää”)
  Else
    Console.WriteLine(”Hyvää huomenta”)
  End If
End Sub
```

Kuva 11. Visual Basic .NET lähdekoodi [9]

Koska eri kielillä kirjoitetut koodit kääntyvät samaan CIL-muotoon, voidaan eri kielillä kirjoitettuja ohjelmia yhdistellä helposti. Isossa ohjelmistoprojektissa osa ohjelmasta voidaankin ohjelmoida esimerkiksi C++:lla ja toinen osa Javalla, jonka jälkeen molemmat lähdekoodit käännetään yhteiseen CIL-muotoon. Monikielinen ympäristö mahdollistaa myös kielten välisen perinnän, jolloin luokka voidaan kirjoittaa C++:lla ja periä C#:ssa [11]. Kieliriippumattomuus mahdollistaa käytännössä sen, että ohjelmistojen kehittäjät saavat vapaasti valita käyttämänsä kielen .NETin tukemista kielistä. Ohjelmoinnista tulee myös entistä tehokkaampaa kun ohjelmoijan ei tarvitse enää vaihtaa kielestä toiseen, jolloin uusien luokkakirjastojen opettelua ei tarvita.

5.1.2 Tietoturva

Toinen .NETin tuoma suuri etu on parannettu tietoturva. .NET ei sinänsä poista tietoturva-aukkoja, mutta siinä on paljon riskejä vähentäviä ominaisuuksia. Tärkein näistä on se, että .NET-koodin suoritus on valvottua. Käytännössä tämä tarkoittaa sitä, että koodia ei perinteiseen Windows tapaan anneta suoraan prosessorille vaan ajosta huolehtii CLR. CLR suorittaa CIL muodossa olevaa koodia ja suorittaa ohjelman haluamat pyynnöt. CLR kuitenkin huolehtii siitä, että ohjelmasta suoritetaan vain turvalliset pyynnöt. CLR estää tehokkaasti mm. ohjelmaa kirjoittamasta toisen ohjelman muistialueelle tai sotkemasta omaansa Application memory isolation –toimintonsa avulla. Yleinen tietoturva-aukko, puskuriylivuoto on myös estetty, sillä CLR pitää huolen, että 10 merkille varattuun muistipaikkaan ei kirjoiteta 20 merkkiä. .NETissä on myös muuttujien vahva tyypitys, eikä tämän vuoksi esimerkiksi kokonaislukua voi vahingossa käsitellä merkkijonona tai toisinpäin.

Muistinhallinnan lisäksi CLR valvoo myös ohjelman käyttämiä resursseja, joita ovat esimerkiksi tiedostot, ikkunat, leikepöytä ja verkkokanavat. Käytännössä CLR voi estää yksittäistä ohjelmaa pääsemästä internetiin tai se voi estää ohjelmaa kirjoittamasta kovalevylle. CLR:n ansiosta esimerkiksi nettisivuilta tulevat haitalliset koodinpätkät voidaan ajaa ns. hiekkalaatikossa eli turvallisessa ympäristössä, eivätkä ne silloin pääse muuttelemaan koneen asetuksia tai suorittamaan omaa koodiaan käyttäjän tietämättä. .NETiin sisältyy myös salausmenetelmiä, kuten SSL ja IPSec.

Hallitun koodin suorittaminen aiheuttaa luonnollisesti hieman viivettä ohjelman käyttöön, mutta nykykoneiden tehoilla viive on käytännössä merkityksetön.

5.1.3 Web-palvelut

.NET-arkkitehtuurissa on mahdollisuus kehittää Web-palveluita, jotka ovat internetissä toimivia palveluita, joita mikä tahansa ohjelma voi käyttää joko paikallisesti tai internetin kautta. Web-palvelut mahdollistavat sen, että käyttäjällä ei tarvitse enää olla erillistä ympäristöä jokaisella käyttämällään koneella, vaan tiedostot ja muu tieto ovat keskitetysti verkossa käyttäjän saatavilla. Käyttäjän käyttämiä palveluita voivat olla esimerkiksi kalenterimerkinnät tai tietokannat. Web-palvelun ja .NET-ohjelman välinen keskustelu tapahtuu SOAP-protokollan avulla. SOAP taas perustuu XML:ään. Web-palvelut käyttävät pelkästään avoimia internet protokollia, joten ne eivät ole alustariippuvaisia. Keskeisiä standardeja ovat SOAPin lisäksi UDDI (Universal Description, Discovery and Integration) ja WSDL (Web Service Description Language). .NETissä on panostettu myös avoimeen tiedonvälitykseen mikä näkyy käytännössä avointen tiedonsiirtomuotojen, kuten XML:n tukemisena.

Edellä mainitut standardit ovat siis keskeisimpiä. SOAPia käytetään kommunikoinnissa, ja viestirakenteet ovat sillä määritellyt. Se määrittää Web-palveluiden toimintamallin [10]. WSDL taas on tapa, jolla Web-palvelu – rajapinta määritellään. Se sisältää sekä kuvauksen Web-palvelu – ratkaisusta ja tiedon siitä, mitä parametreja kutsussa välitetään. UDDI mahdollistaa Web-palvelu – ratkaisun käyttämisen, sillä sen avulla tiedetään sen olemassaolosta.

5.2 Heikkoudet

Vaikka Microsoftin .NET-ohjelmistoarkkitehtuuria on kehuttu paljon, löytyy arkkitehtuurista myös parannettavaa ja heikkouksia. Microsoft haluaa haastaa .NET-tekniikallaan Sunin J2EE-teknologian ja nimenomaan J2EE:n kannattajilta löytyy paljon moitittavaa .NETistä. Vaikka parantunut tietoturva ja kieliriippumattomuus olivat .NETin parhaita puolia, on niitä myös kritisoitu vahvasti.

Pelkästään sitä, että .NET on Microsoftin lanseeraama ja kehittämä tekniikka pidetään huonona asiana. Sunin web-palveluasiantuntija Mark Hapner pitää yleensäkin .NET-pohjaista ohjelmistojenkehitystä huonona ajatuksena, koska Microsoft voi koska tahansa muuttaa .NETin rakennetta. Hapner on myös sitä mieltä, että vaikka .NET on tiukka kilpailija J2EE:lle, on se liikaa Microsoftin kontrollin alla, toisinkuin J2EE, joka tukee kehittäjien ja toimittajien yhteistyötä [12]. J2EE:n tukijoina ovatkin yrityksen, kuten SAP, Oracle, IBM, HP ja BEA, kun .NET on vain Microsoftin tukema teknologia [25].

5.2.1 Windows-keskeisyys

.NET-arkkitehtuurin yksi huono puoli on kaikista puheista huolimatta se, että arkkitehtuuri on Microsoftille tyypillisesti varsin Windows-keskeinen. Vaikka standardoitu arkkitehtuuri mahdollistaa tekniikan siirtämisen muille alustoille, ei ainakaan toistaiseksi kunnollisia ja tarpeeksi laajoja toteutuksia ole olemassa muille kuin Windowsille. Tekniikan tuoreus voi toki olla vielä syy tähän, ja aika näyttääkin kuinka asian kanssa tulee käymään.

Vaikka assemblyt, eli .NET-arkkitehtuurin ohjelmistokomponentit ovat standardin mukaisia, eivät ne silti ole sellaisenaan siirrettävissä ajoympäristöstä toiseen. Syy tähän löytyy ajoympäristöriippuvaisesta ”assembly loaderista”. Käytännössä tämä näkyy siinä, että Microsoftin kaupallisen toteutuksen helloworld.exe osaa itse käynnistää .NET-ajoympäristön, mutta Rotor-ajoympäristössä tulee kirjoittaa ”clix helloworld.exe”. Kääntämätön ohjelmakoodi on luonnollisesti siirrettävissä ajoympäristöstä toiseen, jos vain kääntäjä löytyy eivätkä luokkakirjastot aiheuta rajoitteita. Rajoitteet luokkakirjastojen kohdalla johtuvat siitä, että muut kuin Microsoftin .NET toteutus sisältävät huomattavasti pienemmät luokkakirjastot.

Kunnollisen ajoympäristön puuttuminen muilta kuin Windows-alustoilta rajoittaa myös Web-palveluille luvattua alustariippumattomuutta. Vaikka

palvelut keskustelevat yleisten standardien välityksellä, eivät ne silti lähde edes käyntiin ilman CLR:ää. Jos kunnollinen CLR olisi toteutettu esimerkiksi Linuxille, voisi Web-palvelun pystyttää ilmaiselle Linux-palvelimelle. Toki rajoitetun Web-palvelun voi toteuttaa Mono-projektin CLR:n päälle, mutta jos palveluun haluaa vähänkin enemmän toiminnallisuutta, joutuu turvautumaan Microsoftin CLR:ään [24].

5.2.2 Tietoturva

Ohjelmoijan on hyvä tietää, että Microsoftin .NET-toteutuksen käyttöliittymäkirjasto on osittain hallitsematonta koodia. System.Windows.Forms-nimiavaruudessa sijaitsevat käyttöliittymäluokat sisältävät suhteellisen paljon käyttöjärjestelmäriippuvaisia natiiveja funktiokutsuja Windows API:lle. Tällöin saattaa olla mahdollista suorittaa normaalisti CLR:n hylkäämiä toimintoja.

Myös HailStorm-projektia on moitittu mahdollisista tietoturvariskeistä. Kuten aikaisemmin mainittiin, tekniikan ideana on tarjota ”globaalia passia”, jonka avulla käyttäjä tunnistetaan eri sivustoilla ja palveluissa. Tekniikan riskinä saattaa olla kuitenkin se, että kaikkien käyttäjien tiedot sijaitsevat keskitetysti yhdessä paikassa, mistä saattaa syntyä vakavia tietoturvaongelmia [24].

5.2.3 Kieliriippumattomuus

Vaikka .NETin kieliriippumattomuus on yksi sen parhaista puolista, pidetään sitä myös huonona puolena. Varsinkin mahdollisuus ohjelmoida komponentti useilla eri kielillä herättää epäilyksiä kehittäjissä. Erityisesti koodin ylläpidettävyyden odotetaan kärsivän, jos koodista tulee sekavaa spagettia. Ylläpidosta tulee myös kallista, kun ylläpitäjäksi tarvitaan asiantuntija, joka tuntee kaikki kielet, joilla komponentti on ohjelmoitu. Tällaisen henkilön koulutus vie aikaa ja rahaa, mikä nostaa sovellusten kokonaiskustannuksia.

Projektissa tulee myös suuria ongelmia, jos sovellusta ohjelmoidaan kahdella eri kielellä ja toisen kielen taitaja vaihtaa työpaikkaa. Tällöin vaihtoehtoina on kouluttaa kielen taitajia tai palkata lisää ihmisiä yritykseen [20].

5.3 Yhteenveto

.NET-arkkitehtuuri on vielä melko tuore tekniikka, joten sitä kohtaan kohdistuu kehujen lisäksi myös kritiikkiä. Tutkimuksen kirjoittamisen aikana oli ongelmana löytää puolueetonta materiaalia arkkitehtuurin ominaisuuksista. Yleensä tekniikka kehuvia lähteitä tutkimalla juuret osoittivat Microsoftiin. Tällöin artikkelin luonne oli luonnollisesti tekniikkaa ja uusia mullistavia ominaisuuksia kehuva. Vastaavasti taas tekniikkaan kohdistettua kritiikkiä löytyi Sunin kirjoittamista artikkeleista. Tällaisissa tapauksissa oli yleistä, että .NETiä verrattiin J2EE:hen ja .NETin uusia ominaisuuksia, kuten kieliriippumattomuutta kritisoitiin ankarasti. Usein kirjoittajana oli myös jokin sitoutumaton organisaatio, mutta tutkimus oli tehty Microsoftille tai Sunille, mikä usein heijastui tuloksissa. Käytännössä tämä näkyi esimerkiksi tutkimuksessa, jossa ohjelmoitiin testisovellus J2EE:llä ja .NETillä ja vertailtiin sitä kuinka monta koodiriviä tarvittiin sovelluksen ohjelmoimiseksi molemmilla tekniikoilla. Testin tuloksia esiteltiin kahdessa erillisessä artikkelissa, joista toinen oli tehty Microsoftille ja toinen Sunille. Molemmissa artikkeleissa J2EE:n rivimäärä oli suurin piirtein sama, mutta .NETin rivimäärä oli toisessa artikkelissa noin 2000 ja toisessa noin 4000. Koodimäärä oli siis kaksinkertaistunut riippuen lähdekoodien tulkinnasta ja siitä kumman yrityksen tilauksesta artikkeli oli toteutettu.

6 YHTEENVETO

Tässä tutkimuksessa esiteltiin Microsoftin .NET –ohjelmistoarkkitehtuuria ohjelmistonkehittäjän näkökulmasta. Lisäksi esiteltiin arkkitehtuurin hyviä ja huonoja puolia kannalta. Tutkimuksen tarkoituksena oli selvittää mikä Microsoft .NET-arkkitehtuuri on ja millaista hyötyä siitä on lähinnä ohjelmistojenkehittäjän kannalta. Koska aihe on varsin tuore, oli tutkimuksen tarkoituksena esitellä aluksi arkkitehtuurin rakennetta ja keskittyä sen jälkeen haittoihin ja hyötyihin. Tutkimusta on rajattu siten, että siinä keskitytään nimenomaan Microsoftin kaupalliseen .NET toteutukseen. Muut toteutukset on rajattu pois lähinnä niiden keskeneräisyyden vuoksi.

Arkkitehtuurin rakennetta tutkittiin .NET alustan (platform) ja kehyksen (framework) osalta. Tutkimuksessa keskityttiin kehykseen, koska se on arkkitehtuurin ydin. Ajoympäristö CLR on kehyksen tärkein yksittäinen osa ja hoitaa mm. ohjelmien suorittamisen sekä mahdollistaa esimerkiksi paremman tietoturvan. Yleiset luokkakirjastot mahdollistavat sen, että jokaisella ohjelmointikielellä on yhteiset luokkakirjastot. Tällöin ei tarvitse opetella uusia kirjastoja jos ohjelmoija joutuu vaihtamaan kielestä toiseen. Ajoympäristön ja luokkakirjastojen lisäksi .NET kehyksestä löytyy Data- ja XML-luokat sekä käyttöliittymäkomponentit. Osa näistä komponenteista perustuvat vanhempiin tekniikoihin, mutta jokaisessa komponentissa on myös jotain uutta.

Koska .NET-arkkitehtuuri on varsin tuore tekniikka, on myös sen hyviä ja huonoja puolia tutkittu. Myös tätä asiaa pyrittiin tutkimaan ohjelmistokehittäjän kannalta. Parhaita puolia kehittäjien kannalta onkin ehdottomasti .NETin kieliriippumattomuus ja parantunut tietoturva, jolloin ohjelmoija ei voi esimerkiksi vahingossa sotkea muistialueita. Myös muuttujien vahva tyypitys auttaa ohjelmoijaa, kun kokonaislukua ei voi vahingossa käyttää merkkijonona tai toisinpäin. Molempia ominaisuuksia pidettiin myös kehittäjän kannalta huonoina puolina. Perusteluina oli esimerkiksi se, että jos sovellus

ohjelmoidaan usealla eri kielellä, saattaa lähdekoodista tulla vaikeasti hallittavaa ja päivitettävää.

.NET-arkkitehtuurin rakenteen esittelyä pitäisin varsin luotettavana, mutta tekniikan hyötyjen ja haittojen tutkimisessa oli enemmän haastetta. Valitettavasti lähes aina erilaisten .NETin ominaisuuksia arvioivien tutkimusten taustalta löytyi Microsoft tai Sun, jolloin täysin luotettavaa tutkimusta oli vaikea löytää. Microsoft luonnollisesti kehui uutta tekniikkaansa, kun taas Sun yritti hieman laskea .NET-innostusta, koska tekniikka kilpailee J2EE:n kanssa kehittäjien mielenkiinnosta.

Jatkotutkimuksen aihe voisikin olla tarkempi vertailu näiden kahden tekniikan välillä. Koska Microsof .NET-arkkitehtuurin rakenne on nyt tutkittu, voitaisiin jatkotutkimuksessa keskittyä puhtaasti arkkitehtuurin ominaisuuksiin ja niiden toimivuuteen käytännössä. Haastetta tällaiselle tutkimuksella antaa se, että .NET-pohjaisia sovelluksia ei kuitenkaan vielä ole kovin paljoa käytössä, vaikkakin määrä kasvaa kokoajan.

7 LIITTEET

7.1 .NETin tukemat kielet

.NETin tukemat kielet 22.5.2005

Ada
APL
AsmL
Basic
BETA
BF
C
C#
C++
Caml
Chrome
Cobol
Delphi
Eiffel
Forth
Fortran
G#
Haskell
IL/MSIL (Intermediate Language)
Java (J#, IKVM.NET – Java VM for .NET)
JavaScript
Lexico
LISP
LOGO
Lua
Mercury
Mixal Assembly Language
Mondrian
Oberon
Nemerle
Pan
Perl
Pascal
PHP
Prolog
Python
Python Variants
Ruby
RPG
Scala
Scheme
Small Talk
SML (Standard Meta Language)
Spry
Tcl/Tk

LÄHDELUETTELO

[1] Cheng J., Xu J. 2000. XML and DB2. Proceeding of the 16th International Conference on Data Engineering. February 29-March 3. 569-573.

[2] Conard J., Conrad J., Francis B., Glynn J., Harvey B., Hollis B., Ramachandran R., Schenken J., Short S., Ullman C., Dengler P. 2000. Introducing .NET. Lontoo: Wrox Press Inc.

[3] ECMA International, CLI 2002. Standard ECMA-335 [online]. ECMA International [viitattu 11.4.2005]. Saatavilla www-osoitteesta <http://www.ecma-International.org/publications/standards/Ecma-335.htm>

[4] Farley J. 2000. Microsoft .NET vs. J2EE: How Do They Stack Up? [online]. O'Reilly Media [viitattu 20.4.2005]. Saatavilla www-osoitteesta <http://java.oreilly.com/news/farley_0800.html>

[5] Flink T. 2001. Siirtyminen monitasoarkkitehtuuriin: Microsoft .NET:in tarjoamat mahdollisuudet. Jyväskylän yliopisto, tietotekniikan laitos.

[6] Gero C. 2003. Introduction to ADO.NET [online] Prenia Software & Consulting Services [viitattu 21.4.2005]. Saatavilla www-osoitteesta <http://www.prenia.com/Downloads/ADO.NET.doc>

[7] Haavisto H. 2001. .NET-viestinvälitystekniikat kunnonvalvontajärjestelmän toteutuksessa. Tampere university of technology, automation degree program.

[8] Hamilton J. 2003. Language integration in the common language runtime. ACM Sigplan Notices 38(2), 19-28.

[9] Heikniemi J. 2004. Microsoft .NET yhdistää. Mikrobitti 11/2004, 78-80.

[10] Kalliola J. 2004. XML-tuki ohjelmointikielissä & Web-palvelut [online]. TKK Viestintätekniikan laboratorio [viitattu 11.4.2005]. Saatavilla www-osoitteesta <http://www.media.hut.fi/as0110/kalvot/8-soap.pdf>

[11] Kero P. 2000. Microsoft .NET [online]. Helsingin yliopisto [viitattu 14.4.2005] Saatavilla www-osoitteesta <http://www.cs.helsinki.fi/u/laine/otv/NET.html>

[12] Krill P. 2005. Java vs. Microsoft .Net debate rages [online]. InfoWorld [viitattu 20.5.2005]. Saatavilla www-osoitteesta <http://www.infoworld.com/article/05/03/05/HNpaneljava_1.html>

[13] Microsoft 2002. ADO.NET Overwiew [online]. Microsoft Corporation [viitattu 21.4.2005]. Saatavilla www-osoitteesta <http://samples.gotdotnet.com/QuickStart/howto/default.aspx?url=/quickstart/howto/doc/adoplus/ADOPlusOverview.aspx>

[14] Microsoft 2005. Microsoft .NET Homepage [online]. Microsoft Corporation [viitattu 22.3.2005]. Saatavilla www-osoitteessa <http://www.microsoft.com/NET/>.

[15] Microsoft 2005. .NET Framework Developer Center [online]. Microsoft Corporation [viitattu 14.4.2005]. Saatavilla www-osoitteessa <http://msdn.microsoft.com/netframework/>.

[16] Miller G. 2003. The Web services debate: .NET vs. J2EE. Communications of the ACM 46(6), 64-67

[17] Mono-Project 2005. Main Page [online]. Mono-Project [viitattu 11.4.2005]. Saatavilla www-osoitteesta <http://www.mono-project.com/Main_Page>

[18] Munro J. 2001. The Microsoft .NET Development Environment [online]. ExtremeTech [viitattu 16.4.2005]. Saatavilla www-osoitteesta <http://meta.csd.uwo.ca/~wade/Research/PLG/Papers/netdoc.pdf>

[19] Platt D. S. 2001. Microsoft .NET: uudet ominaisuudet. Helsinki: Edita.

[20] Roman E., Vawter C. 2001. J2EE vs. Microsoft.NET: A comparison of building XML-based web services [online]. The Middleware Company [viitattu 25.4.2005]. Saatavilla www-muodossa <http://www.theserverside.com/articles/pdf/J2EE-vs-DotNET.pdf>

[21] Ritchie B. 2005. .NET Languages [online]. Brian Ritchie's .NET Development Site [viitattu 10.4.2005]. Saatavilla www-osoitteesta <http://www.dotnetpowered.com/languages.aspx>

[22] Schmitt D. 1999. Microsoft .NET Overview [online]. Microsoft Corporation [viitattu 16.4.2005]. Saatavilla www-osoitteesta <http://www.umsl.edu/studentlife/misclub/Framework-HP.ppt>

[23] W3Schools. Introduction to ADO [online]. Refsnes Data [viitattu 21.4.2004]. Saatavilla www-osoitteesta <http://www.w3schools.com/ado/ado_intro.asp>

[24] Weiss A. 2001. Microsoft's .NET: platform in the clouds. NetWorker 5(4), 26-31.

[25] Williams J. 2003. The Web services debate: J2EE vs. .NET. Communications of the ACM 46(6), 58-63.