Nico Virkki

Työkaluvaraston hallintaohjelmisto

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikka
Insinöörityö
3.5.2013

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Nico Virkki Työkaluvaraston hallintaohjelmisto 39 sivua 3.5.2013
Tutkinto	Insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja(t)	Sovelluskonsultti Eero Räisänen Yliopettaja Erja Nikunen

Norpe Oy on vuonna 1953 perustettu kylmäkalusteratkaisujen valmistaja. Norpe Oy toimii ympäri maailmaa ja on markkinajohtaja Pohjoismaissa ja Baltian alueella. Norpe Oy:ssä kalustekokoajat valmistavat kylmäkalusteet osista. Yrityksen liiketoiminnan kasvaessa henkilöstön sekä työkalujen lukumäärä ovat kasvaneet vuosien saatossa. Tällä hetkellä Norpe Oy:n työntekijöillä on käytössä satoja erilaisia työkaluja. Työkalut luovutetaan työntekijöille työkaluvarastosta ja työkaluvaraston vastuuhenkilö pitää niistä kirjanpitoa.

Suuren työkaluvaraston kirjanpitoon tarvitaan siihen soveltuvaa ohjelmistoa. Norpe Oy:n käytössä oli komentorivipohjainen FoxPro-tietokantaan pohjautuva sovellus. Tämä vanha sovellus ei enää täyttänyt nykyhetken tarpeita, joten uuden sovelluksen tuottaminen oli ajankohtaista.

Vanhassa FoxPro-sovelluksessa oli komentorivipohjainen käyttöliittymä. Sovellus käytti FoxPro-tietokantaa. Vanhan sovelluksen ongelmana oli se, että työkaluvaraston ylläpitäjän ollessa poissa, kukaan ei pystynyt käyttämään sovellusta, eikä kirjanpito pysynyt ajan tasalla. Uudelta CoolTool-sovellukselta toivottiin graafista käyttöliittymää ja, että sen käyttö olisi helpommin omaksuttavaa.

Tämän insinöörityön tavoitteena oli toteuttaa uusi työkaluvaraston hallintasovellus. Tässä työssä käsitellään uuden CoolTool-työkaluvaraston hallintasovelluksen arkkitehtuuria ja sen käyttämää Microsoft .Net-sovelluskehyksen teknologiaa sekä tutkitaan mahdollisia jatkokehitystarpeita. Tämän työn tekninen toteutus on tehty kesätyöharjoittelun ohessa vuonna 2011 1.6 – 10.8. CoolTool-sovellus on tehty Microsoftin Visual Studio 2010:llä .NET-ohjelmakomponenttikirjastoa hyödyntäen.

Avainsanat	C#, .NET, ASP.NET

Metropolia

Abstract

Author(s) Title Number of Pages Date	Nico Virkki Management software for tool storage 39 pages 3 May 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Eero Räisänen, Software Consultant Erja Nikunen, Principal Lecturer

Norpe Oy is refrigeration solutions provider founded in 1953 in Porvoo, Finland. Norpe Oy operates globally and it is the market leader in the Baltic Rim area and the Nordic countries. In Norpe Oy the refrigeration equipment collectors assemble the refrigeration solutions. Employees in Norpe Oy have hundreds of different tools. The tools are released to employees from a tool storage and the person in charge of the tool storage keeps a tally of them.

The storage with a great number of tools obviously need a piece of software and database to keep track of the tools. Norpe Oy had management software based on FoxPro database. This software no longer met the increased needs of maintaining the storage and its tools. So there was a demand for new management software.

The problem with the old management software was that when the administrator of the tool storage was away noone could use the management software because it took time to get familiar with it. Requirements for new CoolTool management software were that it should have graphical web-based user interface and the usage would be easy to adopt.

The aim of this study was to produce new tool storage management software. The thesis introduces the new CoolTool tool management software architecture and .NET Framework technology behind it. The technical implementation was done during summer internship in 2011. CoolTool software was made using Microsoft Visual Studio 2010 and .NET Framework components.

Keywords	C#, .NET, ASP.NET

Sisällys

Lyhenteet

POCO	Plain Old Clr Object, yksinkertainen .NET-luokka.
string	Sovelluskehityksessä käytetty termi merkkijonosta.
.NET	.NET Framework -sovelluskehys.
Triggeri	Tietokannan automaattinen proseduuri.
XML	Extensive Markup Language, merkintäkieli.
C#	Microsoftin kehittämä oliopohjainen ohjelmointikieli.
CIL	Common Intermediate Language, matalan tason ohjelmointikieli.
CLR	Common Language Runtime, .NET-ympäristön virtuaalikone.
Työkalu	Työkalu, jota käytetään kylmäkalusteiden valmistukseen.
Kulutustuote	Työkalu tai sen varaosa jota tarvitaan kylmäkalusteiden valmistuksessa.
Työkalupakki	Kuvaa työkaluvaraston hallintaohjelmistoon kirjatun henkilön työkaluja.
Varaosapakki	Kuvaa työkaluvaraston hallintaohjelmistoon kirjatun henkilön kulutustuotteita.
SVN	Subversion-versionhallinta.
IIS	Internet Information Services. Microsoftin Web-palvelinohjelmisto.

1 Johdanto

Insinöörityön tarkoituksena on korvata Norpe Oy:n työkaluvaraston vanha FoxProlla toteutettu työkaluvarastonhallintasovellus uudella CoolTool-sovelluksella, .NET-sovelluskehystä hyödyntäen.

Norpe Oy on Porvoossa toimiva kylmälaitteiden valmistukseen erikoistunut yritys, joka työllistää noin 400 työntekijää. Työkaluja tämänkokoisessa teollisuusyrityksessä on mittava määrä, joten niiden kirjanpito on syytä hoitaa tietokantapohjaisesti. Työkalujen kirjanpitoa ylläpitää työkaluvaraston työntekijä, jonka kautta jokainen työkalu kulkee. Työkaluvarastossa on varsinaisten työkalujen lisäksi, joita tarvitaan kylmälaitteiden valmistuksessa, työkalujen varaosia, hanskoja, kenkiä jne.

Työkalujen kirjanpitoon työkaluvarasto on käyttänyt aikaisemmin käyttöliittymänään tekstipohjaista ratkaisua käyttävää FoxProlla toteutettua työkaluvaraston hallintasovellusta. Sitä ajetaan MS-DOS -ympäristössä. Selvää on, että nykyajan mittapuulla sovellus on jo parhaat päivänsä nähnyt, kun teknologia on kehittynyt eteenpäin. Sovelluksen uusimisen tarpeellisuuteen vaikutti myös se, että vanha sovellus tuli epäluotettavaksi, sillä datan määrä oli kasvanut 20 vuodessa sovelluksen äärirajoille.

Tarkoituksena oli siis korvata tämä vanha sovellus uudella sovelluksella. Vaatimuksena uudelle sovellukselle oli, että sen tulisi toimia selaimessa ja tulisi hyödyntää Microsoft .NET Frameworkia ja C#-ohjelmointikieltä. Vaatimusten puitteissa uuden sovelluksen arkkitehtuuriksi muodostui .NET-sovelluskehyksen ASP.NET -kirjastoa hyödyntäen toteutettu web-käyttöliittymä ja tietokantaoperaatioita varten .NET-sovelluskehyksen ADO.NET -kirjaston komponentit. Sovelluksen tietokantana toimii Microsoft SQL Server 2005. Tässä työssä käydään läpi edellä mainittuja ohjelmakomponenttikirjastoja sekä uutta CoolTool-työkaluvarastonhallintasovellusta, joka korvasi vanhan sovelluksen.

2 Norpe

Norpe Oy on suomalainen, Porvoossa vuonna 1953 perustettu, kylmäkalusteratkaisujen valmistaja, jonka toiminta ulottuu Eurooppaan, Lähi-itään sekä Australiaan. Norpe on markkinajohtaja Pohjoismaissa ja Baltian alueella sekä kasvaa voimakkaasti Keski-Euroopassa ja Venäjällä. Norpen pääkonttori sijaitsee Porvoossa, ja se työllistää yli 400 kylmäalan ammattilaista. [Norpe, Norpe tänään, 2013.]

2.1 Norpen historia

Norpen perusti Reijo Mäntyoja vuonna 1953. Nimellä Norpe toiminta alkoi vuonna 1954 Porvoossa. Yhtiön ensimmäiset tuotteet olivat 500-1200 litraiset jäähdytyskaapit, maidon jäähdyttimet sekä Suomen ensimmäiset avonaiset kylmälasikot. Tämän jälkeen vuosina 1958 - 1973 jälkeen toiminta laajeni kylmä- ja pakastekuljetusautojen valmistuksesta supermarket-kalusteiden ja kotikäyttöisten pakastekaappien valmistukseen. 80-luvun puolessa välissä Norpe fuusioitui Hollming Oy:n kanssa, ja yhtiö sai uudeksi nimekseen Hollming Oy Norpe, josta palattiin takaisin vanhaan Norpe-nimeen jo vuonna 1990. Vuonna 2005 MB-rahastot ostivat Norpen. [Norpe, Laatua jo lähes 60 vuotta, 2013.]

Insinöörityön sovellusta tein Norpe Oy:ssä IT-osastolla sovellussuunnittelijaharjoittelijana sovellusasiantuntijatiimissä, jonka tehtävänä on kehittää ja ylläpitää yrityksessä käytettäviä tietojärjestelmiä. IT-osastoon kuuluu myös mikrotuki, joka vastaa työasemien ja lähiverkon ylläpidosta sekä sovellustuen antamisesta. Lisäksi osastoon kuuluu tietohallintopäällikkö, joka toimii esimiehenä IT-osastolla työskenteleville henkilöille sekä vastaa osaston toiminnasta johtoryhmälle.

2.2 Norpen työkaluvarasto

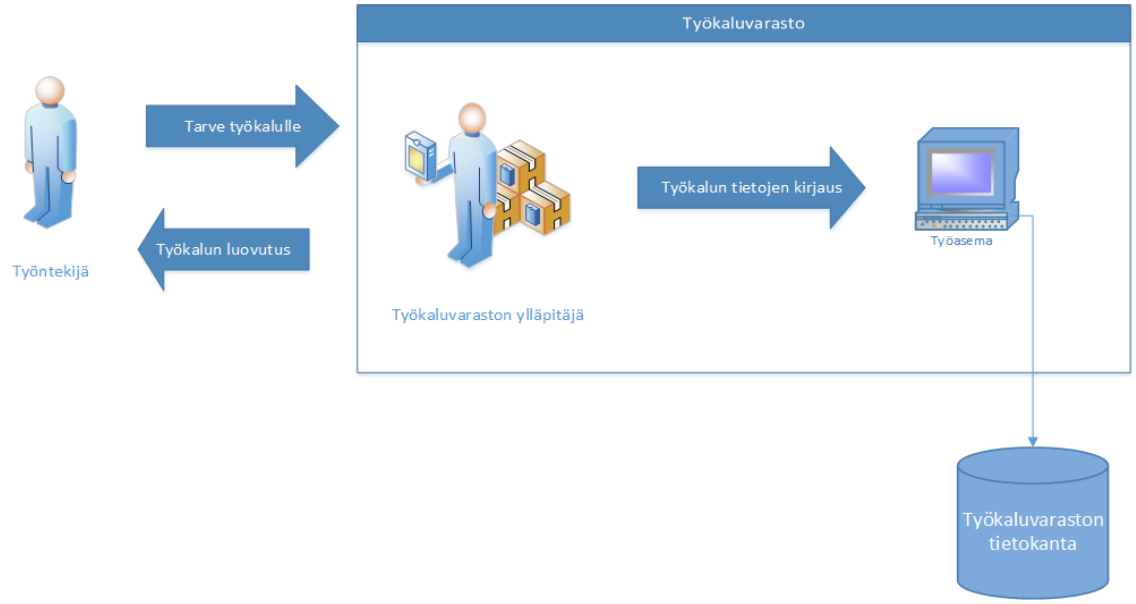
Norpen työkaluvarasto kuuluu Norpen tehdaspalveluosastoon. Tehdaspalvelu huolehtii Norpen tehtaiden ja toimistojen huoltotöistä, kuten esimerkiksi sähkötöistä ja tehdaslaitteiden huoltotöistä. Työkaluvaraston tehtävänä on tarjota Norpen työntekijöille heidän tarvitsemansa työkalut, ja niiden mahdolliset huollot ja kalibroinnit, työtehtäviensä tekemiseen. Työkaluvarasto pitää kirjaa työkaluista ja niille tehdyistä mahdollisista kalibroinneista. Työkaluvaraston asiakkaita ovat siis koko Norpen henkilöstö sekä mahdol-

liset aliurakoitsijat tehtailla. Suurimman osan työkaluvaraston asiakkaista muodostavat kalustekokoajat, jotka tarvitsevat erilaisia työkaluja kylmäkalusteiden kokoamiseen tai niiden osien valmistamiseen.

Työkaluvarasto on siis fyysinen paikka Norpen tehtaalla, eikä käsitettä työkaluvarasto tule sekoittaa työkaluvarastoon liittyvään ohjelmaan. Insinöörityön sovelluksen tekovaiheessa Norpen työkaluvarastolla oli yksi vakituinen työntekijä, joka hoiti uusien työkalujen tilauksen, kirjanpidon ja huollon. Työkalujen kirjanpitoa helpottamaan työkaluvarastolla oli käytössään Norpella itse tuotettu sovellus työkaluvaraston hallintaan. Tämä hallintaohjelma oli tehty 80- ja 90-luvun taitteessa, ja sen käyttöliittymä oli tekstipohjainen, joka toimi näppäinsyötteillä.

2.3 Työkaluvaraston toiminta

Työntekijän tarvitessa uutta työkalua, hän menee työkaluvarastoon pyytämään työkalua. Arvokkaampien työkalujen tapauksissa tarvitaan esimiehen lupa tarpeelle. Työntekijä pyytää uutta työkalua, ja työkaluvaraston ylläpitäjä luovuttaa työntekijälle hänen tarvitsemansa työkalun ja muokkaa työkalun omistajuustiedot työkaluvaraston hallintaohjelmaan. Työntekijän työtehtävien vaihtuessa tehtävään, jossa hän ei enää tarvitse hänelle osoitettuja työkaluja, työntekijä palauttaa hänelle osoitetut työkalut takaisin työkaluvarastoon, ja työkaluvaraston ylläpitäjä kirjaa palautetut työkalut työvaraston hallintaohjelmistoon varastossa oleviksi. Muita työkaluvaraston tehtäviä on tulostaa listauksia työntekijöille heidän työkaluistaan, heidän sitä pyytäessä, esimerkiksi kun työntekijän tarvitsee tietää, mitä työkaluja hänelle on kirjattu. Näin tapahtuu yleensä silloin, kun työntekijä epäilee, että häneltä on hävinnyt työkaluja. Kuva 1 pyrkii havainnollistamaan tapausta, jossa työntekijä tarvitsee uuden työkalun ja asioi työkaluvaraston kanssa.



Kuva 1. Työkalun luovuttamisesta työntekijälle

Työkaluvarastossa kirjanpitoa varten työkalut jaotellaan kahteen ryhmään: kulutustuotteet sekä varaosat ja työkalut. Kulutustuotteisiin ja varaosiin kuuluu esimerkiksi puukot, poranterät ja käsineet ja tuotteet jotka ovat ns. ”kulutustavaraa”. Työkalujen joukkoon kuuluvat jakoavaimet, kiintolenkit, ruuvimeisselit ja tasohiomakoneet jne. Tuotteet, jotka voidaan suoranaisesti luokitella työkaluiksi ja joiden käyttöikä on Kulutustuotteita sekä varaosia pidempi, kuuluvat työkaluihin. Osa työkaluista on erikoistyökaluja, jotka täytyy kalibroida tietyn ajanjakson välein. Tällaisia työkaluja ovat esimerkiksi erilaiset mitat. Kalibroinnin suorittaa useimmiten työkaluvaraston ylläpitäjä tai muu henkilö tehdaspalvelusta. Vanhassa sovelluksessa ei ollut mahdollisuutta kirjata ylös laitteen kalibrointiajankohtaa. Uuden sovelluksen oli tarkoitus ratkaista tämä ongelma. Työkaluista pidetään yksilöllistä kirjanpitoa ja ne numeroidaan numerosarjalla, joka identifioi jokaisen työkalun, numerointi tapahtuu useimmiten kaivertamalla numero työkaluun, joka saadaan työkaluvaraston kirjanpitosovelluksesta. Kulutus- ja varaosista pidetään vain varastosaldon lukumäärätietoa sekä tietoa, kenellä näitä kulutustuotteita on ja monta kappaletta niitä on työntekijälle annettu. Työkalut palautetaan työkaluvarastoon asiakkaan työsopimuksen päättyessä tai työtehtävän vaihdon yhteydessä, mikäli uudet tehtävät eivät vaadi työntekijälle osoitettuja työkaluja. Kun työkalut palautetaan työkaluvarastoon, ne kirjataan takaisin varastossa oleviksi. Kulutustuotteita sekä varaosia ei yleensä palauteta.

Työkaluvaraston, ja sen hallintaohjelmiston, sekä uuden CoolToolin että vanhan sovelluksen näkökulmasta jokaisella työntekijällä on ns. ”työkalupakki” ja ”kulutustuote sekä varaosapakki”. Sovelluksen rakennetta kuvataan myöhemmissä luvuissa. Työkalupakkiin kirjataan työntekijälle luovutetut työkalut ja varaosapakkiin työntekijälle luovutetut kulutustuotteet. Varaosapakista pidetään kirjaa, jotta työkaluvarastolla olisi käsitys työntekijöille luovutetuista kulutustuotteista mahdollisten ylimääräisten luovutusten takia. Työkaluvarastoon tulee usein asiakkaita tiedustelemaan heidän ”pakkinsa” sisältöä, sillä ajansaatossa työkalut saattavat mennä tehtaalla sekaisin ja asiakkaiden on hyvä tietää, mitä työkaluja heille kuuluu, jotta he itse pystyvät pitämään kirjaa omista työkaluistaan esimerkiksi mahdollisten palautusten takia. Työntekijöiden tiedustelut heidän työkaluistaan kuormittavat työkaluvaraston toimintaa ja yksi uuden sovelluksen tarkoituksista oli poistaa tämä ongelma.

2.4 Uuden sovelluksen vaatimukset ja tarve

Työkaluvarastossa oli käytössä noin 20 vuotta vanha oleva FoxProlla toteutettu sovellus, jota ajettiin virtuaalikoneella DOS-käyttöjärjestelmän sisällä. Sovelluksen käyttöliittymä oli tekstipohjainen, ja tietokanta oli alkanut käydä epävakaaksi vuosien varrella. Tietokanta kadotti indeksejä ja linkityksiä taulujen välillä. Lisäksi työkalujen määrän kasvaessa sovelluksen käyttämä juokseva kolminumeroinen työkalujen numerointi oli kohtaamassa tilanteen, jossa uusia numeroita ei ollut enää mahdollista tuottaa. Kaiken tämän seurauksena oli syntynyt tarve uudelle modernimmalle sovellukselle.

Uudelta sovellukselta odotettiin graafista käyttöliittymää, jota olisi mahdollista käyttää web-selaimella. Uuden sovelluksen toivottiin myös tuovan ratkaisun työkalujen numerointiongelmaan. Sovelluksen graafisen käyttöliittymän toivottiin parantavan sovelluksen käyttöä tilanteissa joissa työkaluvaraston työntekijä poissa ja tuuraaja tarvitaan. Vanhan FoxPro-sovelluksen käyttö oli tuttua työkaluvaraston työntekijälle, mutta sen käyttö oli haasteellista muille mahdollisille käyttäjille.

Työkaluvaraston ylläpitäjän kanssa käytiin keskusteluita uuden työkaluvaraston sovelluksen vaatimuksista ja toiminnoista. Työkaluvaraston ylläpitäjän kanssa käytyjen keskusteluiden perusteella, hahmottuivat uuden sovelluksen keskeisimmät ominaisuudet ja tarpeet. Seuraavassa on listaus työkaluvaraston ylläpitäjän kanssa käytyjen keskuste-

luiden pohjalta saatujen tietojen avulla hahmotellut uuden sovelluksen keskeiset ominaisuudet ja toiminnot:

- Työkalujen numerointi: Numero on ”kiertävä” nelinumeroinen (vapautetaan käyttöön, kun työkalu poistetaan)

- Kalibroitavat ja muut työkalutyypit: Kalibroitavalla työkalulla on tieto kalibrointipäivämäärästä ja seuraavan kalibroinnin päivämäärästä.

- Työkalulle on mahdollista kirjata hinta ja takuutiedot.

- Useita eri kriteerejä työkalujen hakuun sekä useita hakutulosten lajittelumahdollisuuksia.

- Henkilön kulutustuotteiden saldomääristä ei tarvitse pitää kirjaa.

- Henkilötietojen muokkaaminen: Työntekijöiden, sovelluksessa olevaa, numeroa täytyy pystyä muuttamaan jälkikäteen.

- Pakki voitava siirtää kokonaan varastoon yhden painikkeen avulla.

- Tulostusmahdollisuuksia: Työkalulistaus, työntekijän pakin sisällön listaus, palauttamattomien työkalujen listaus työntekijäkohtaisesti.

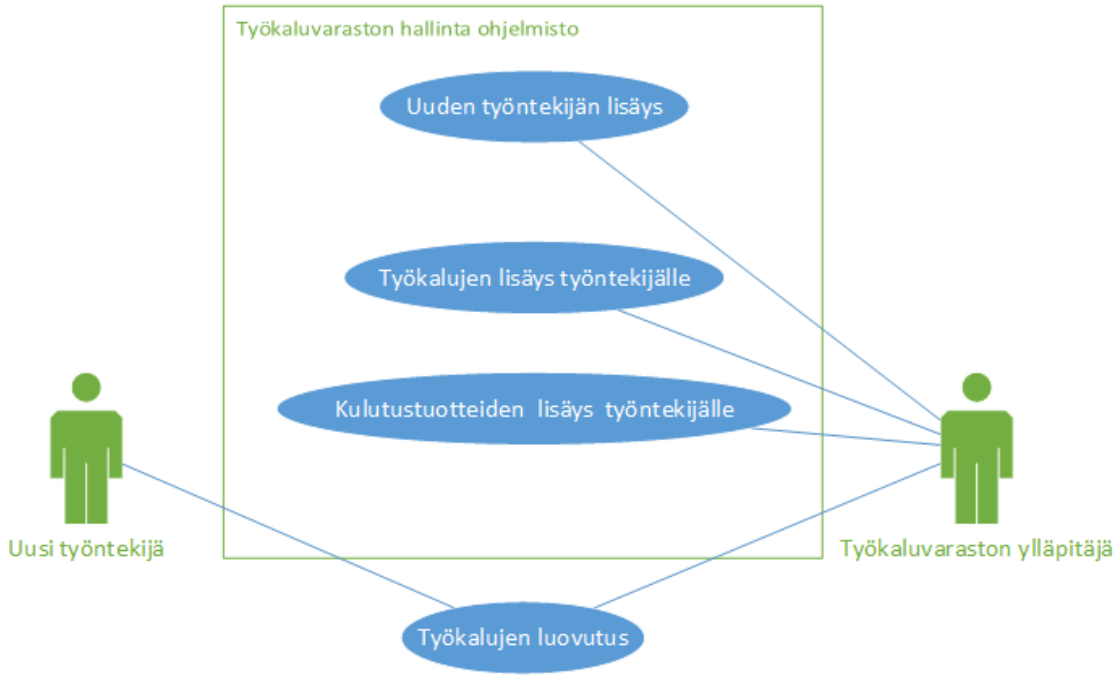
- Työntekijän on mahdollista tarkastella tulostaa omat työkalunsa työasemalta

Uuden sovelluksen teknisenä vaatimuksena oli, että se täytyi toteuttaa .NET-kehyksen tarjoamilla ratkaisuilla käyttäen C#-ohjelmointikieltä. Sovelluksen tietokanta tuli toteuttaa Microsoft SQL Server 2005:lle.

Teknisten vaatimusten lisäksi sovelluksella oli useita käyttötapauksia joiden pohjalta sovelluksen suunnittelua ja toteutusta oli hyvä lähteä ratkaisemaan. Seuraavana on kuvattu neljä keskeistä käyttötapausta, jotka sisältävät joukon toimintoja, jotka sovelluksen täytyi suorittaa.

2.4.1 Käyttötapaus: Uusi työntekijä saapuu yritykseen

Työkaluvarastossa tulee usein tarvetta lisätä uusi työntekijä järjestelmään sekä antaa tälle työkaluja. Uudelta sovellukselta vaadittiin tämänlaisen käyttötapauksen suorittamista. Tilanteessa, jossa uusi työntekijä tulee työskentelemään tehtäviin joissa tarvitaan työkaluja, täytyy työntekijä lisätä työkaluvaraston hallintaohjelmistoon sekä antaa työntekijälle hänen tarvitsemansa työkalut. Lopuksi työkaluvarasto luovuttaa työkalut työntekijälle. Kuvassa 2 on kuvattu työkaluvaraston hallintaohjelmistossa suoritetut tehtävät vihreän laatikon sisään sekä ohjelmiston ulkopuoliset tehtävät laatikon ulkopuolelle.



Kuva 2. Uuden työntekijän lisäys sovellukseen sekä työkalujen ja kulutustuotteiden lisäys uudelle työntekijälle

Uudelta sovellukselta toivottiin, että työntekijöiden tietoja voitaisiin uuden työntekijän lisäyksen yhteydessä hakea myös olemassa olevasta henkilötietojärjestelmästä, jolloin henkilötietoja ei tarvitsisi aina kirjata käsin.

2.4.2 Käyttötapaus: Työkalujen siirto toiselle työntekijälle

Työkaluvarastossa tulee tilanteita, jolloin jonkun yksittäisen työntekijän hallussa olevat työkalut joudutaan siirtämään toiselle työntekijälle. Tämänlaisia tilanteita ovat esimer-

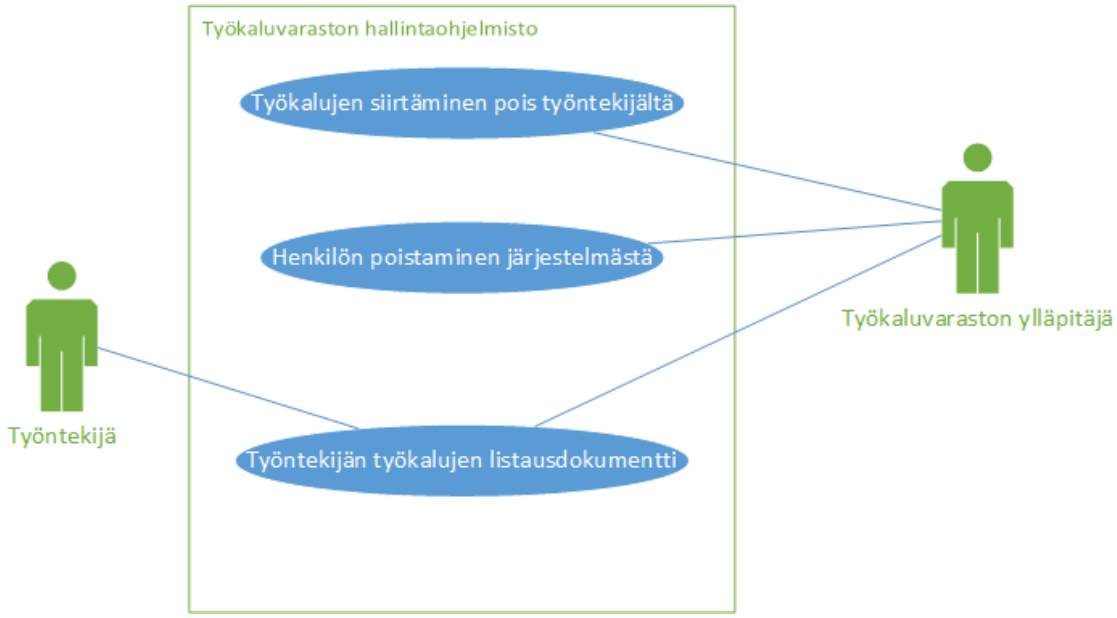
kiksi työntekijän tehtävän kuvan vaihtuminen tai työntekijän työsuhteen päättyminen, jonka johdosta tehtaalle tulee uusi työntekijä vanhan työntekijän tilalle. Joissain tapauksissa vain yksittäisiä työkaluja siirretään esimerkiksi, kun työntekijän tehtävän kuvat vaihtuvat osittain. Uuden työkaluvaraston hallintaohjelmiston yhtenä vaatimuksena oli, että se pystyisi ratkaisemaan tämän ongelman.

Kuva 3. Käyttötapaus: työkalujen siirto toiselle työntekijälle

2.4.3 Käyttötapaus: Työntekijän poistaminen

Henkilöiden poistaminen tulee kyseeseen esimerkiksi silloin, kun työntekijä lähtee yrityksestä. Henkilön poistaminen työkaluvaraston hallintajärjestelmästä vaatii työntekijän työkalujen vapauttamista takaisin varastoon tai siirtoa toiselle työntekijälle. Työntekijän poistuessa yrityksestä työntekijän täytyy saada työkaluvarastolta listaus hänelle jääneistä työkaluista mahdollisia loppupalkasta tehtäviä vähennyksiä varten.

Kuvasta 4 nähdään työkaluvaraston hallintaohjelmistolta vaadittavat tehtävät ja näiden tehtävien toimijat. Työkaluvaraston ylläpitäjä siirtää työntekijän työkalut takaisin varastossa oleviksi (työkalujen siirtäminen pois työntekijältä), jonka jälkeen työkaluvaraston ylläpitäjä tulostaa ja kuittaa lomakkeen, josta käyvät ilmi lähtevän työntekijän hallussa olevat työkalut ja niiden yhteisarvo. Tulostettu dokumentti jää työntekijälle lähtöselvitystä varten.

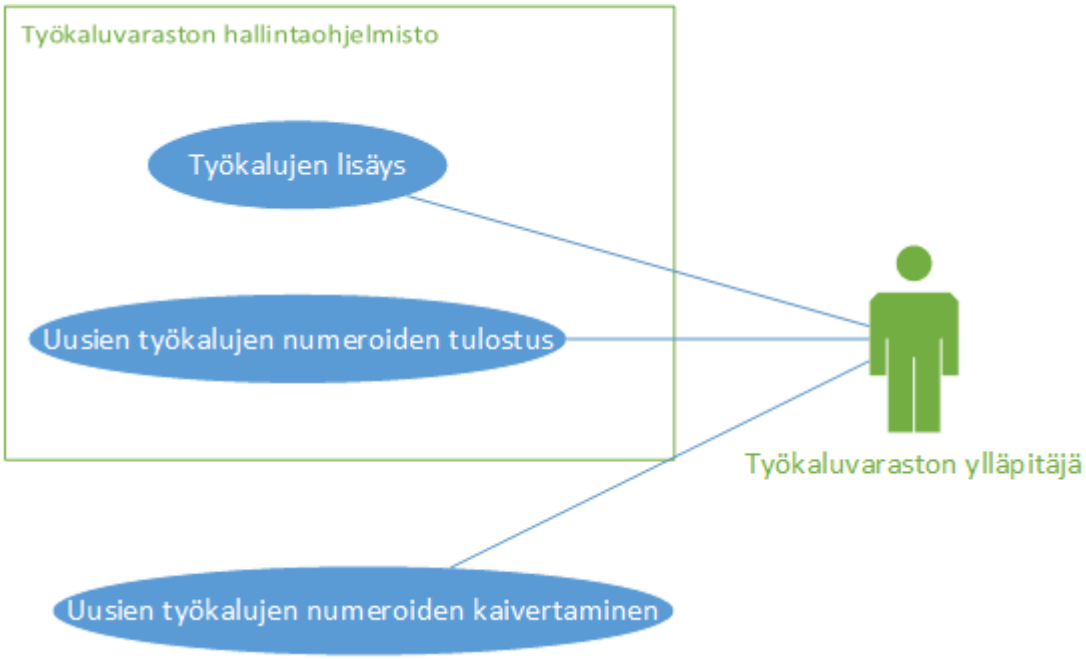


Kuva 4. Käyttötapaus: työntekijän poistaminen työkaluvaraston hallintajärjestelmästä.

2.4.4 Työkaluvarastoon saapuu erä uusia työkaluja

Työkaluvarastoon saapuu uusia työkaluja sekä kulutustuotteita usein. Vanhalla työkaluvaraston hallintasovelluksella jokainen työkalu täytyi erikseen kirjata sovellukseen. Esimerkiksi 20 kpl:n ruuvimeisselierän saapuessa työkaluvarastoon oli jokainen ruuvimeisseli kirjattava erikseen sovellukseen. Uuden sovelluksen toivottiin tuovan ratkaisu usean samanlaisen työkalun lisäykseen.

Kuvassa 5 työkaluvarastoon saapuu n-kappaletta uusia työkaluja, jotka voidaan kirjata järjestelmään syöttämällä tiedot vain kertaalleen. Kun työkalut on kirjattu työkaluvaraston hallintaohjelmistoon, työkaluvaraston ylläpitäjä tulostaa listauksen uusista työkaluista sekä niiden saamista numeroista ja kaivertaa numerot työkaluihin. Itse työkalujen numeroiden kaivertaminen ei kuulu työkaluvaraston hallintaohjelmiston toimintaan, mutta on tärkeä osa uusien työkalujen lisäysprosessia.



Kuva 5. Useiden työkalujen kirjaaminen sovellukseen ja muut toimenpiteet.

3 .NET-sovelluskehyksen arkkitehtuuri

Edellisessä luvussa perehdyimme työkaluvaraston toimintaan ja sen sovelluksen vaatimuksiin sekä käyttötapauksiin. Tässä luvussa käymme läpi tässä työssä käytettyjä keskeisimpiä .NET-sovelluskehyksen ominaisuuksia ja toimintoja, joita käytettiin CoolTool-sovelluksen tekemiseen.

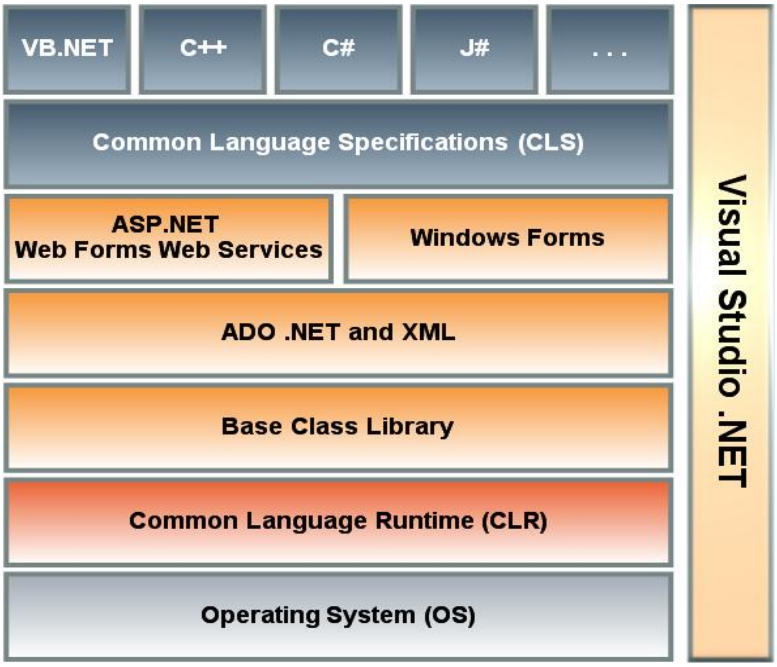
.NET-sovelluskehys on Microsoftin vuonna 2002 julkaisema sovelluskehys / luokkakirjasto Microsoft Windows -käyttöjärjestelmille. Uusin .NET-sovelluskehysversio tätä työtä kirjoittaessa on 4.5. Tässä työssä käytetty versio on 4.0. .NET-luokkakirjasto tarjoaa suuren määrän valmiita ohjelmakomponenttikirjastoja, jotka ovat tiiviisti integroitu .NET:n ajonaikaiseen ympäristöön, Common Language Runtimeen (CLR). [Microsoft, .NET Framework Conceptual Overview, 2012.]

.NET-sovelluskehys tukee lukuisia eri ohjelmointikieliä. Tuettuja ohjelmointikieliä ovat mm. Visual Basic .NET, C#, C++ .NET ja F#, joista yleisimmät ovat Visual Basic ja C#. Suurin osa Microsoftin dokumentaatiosta on tehty C#:lle ja Visual Basicille. Tästä huolimatta dokumentaatio pätee myös muille Frameworkin kielille johtuen frameworkin kielirajoittumattomuudesta.

.NET-sovelluskehys sisältää mittavan määrän ohjelmakomponenttikirjastoja, joita sovelluskehittäjä voi kehittämässään ohjelmassa käyttää hyödyksi. Tässä työssä käytetyimmät kirjastot ovat tietokantakäsittelyihin tarkoitettu ADO.NET sekä web-ohjelmointiin ns. aspx-sivustojen luontiin tarkoitettu ASP.NET, jonka tarjoamien komponenttien ja palveluiden avulla pystytään luomaan dynaamisia web-sivustoja.

Yleisesti käytettävä sovelluskehitin .NET-sovelluskehykselle on Microsoft Visual Studio (VS) -tuoteperhe, johon kuuluu eritasoisia versioita. Tässä työssä on käytetty VS 2010 Professionalia. Eritasoisten versioiden eroina ovat niiden sisältämät ominaisuudet kuten versiot, jotka kattavat useamman ohjelmointikielen, monipuolisemman debuggerin sekä tiimikehitysvälineitä integroituna. Visual Studioihin, 2008:n versioista eteenpäin, on mahdollista ladata ulkopuolisia lisäosia, kuten esimerkiksi rajapinnan svn-versionhallintaan.

Kuvassa 6 nähdään .NET-sovelluskehyksen arkkitehtuuria ja miten Visual Studio siihen liittyy. Visual Studiota ajetaan, kuten tavallistakin työpöytäsovellusta käyttöjärjestelmän päällä. Sen avulla pystytään käyttämään .NET-sovelluskehyksen komponentteja lähdekoodissa.

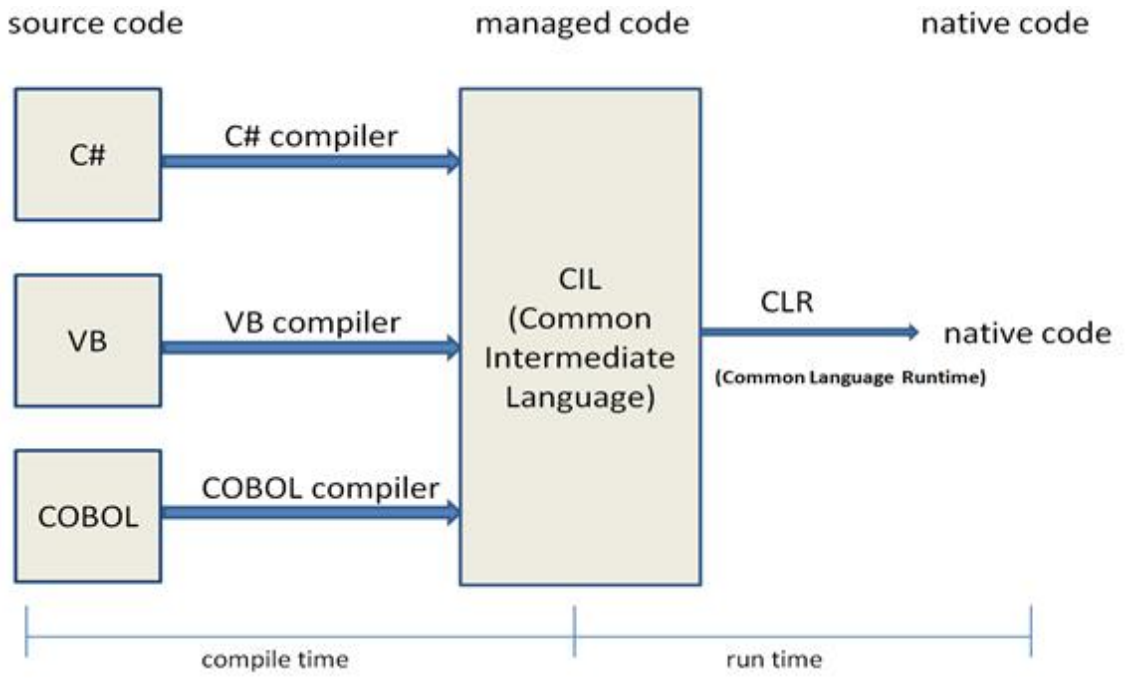


Kuva 6. .NET-sovelluskehyksen arkkitehtuuri

3.1 Common Language Runtime ja .NET-ohjelman kääntäminen

Common Language Runtime (CLR) on ajonaikainen ympäristö, jossa .NET-ohjelmat ajetaan. CLR huolehtii muistinkäytöstä, säikeiden suorituksesta ja poikkeuksien hallinnasta. Java-ohjelmoinnin näkökulmasta CLR:ää voidaan pitää vastaavana kuin Javan virtuaalikonetta. Koodi, jota ajetaan CLR:n alaisuudessa, kutsutaan ”managed codeksi”.

.NET-ohjelmaa käännettäessä kääntäjä kääntää ohjelman CIL-koodiksi (Common Intermediate Language), alustariippumattomaksi kieleksi, jonka CLR kääntää natiivikoodiksi. Tätä vaihetta kutsutaan Common Language Infrastructureksi (CLI), joka on Microsoftin kehittämä avoin spesifikaatio. Ohjelman käännösvaiheessa kääntäjä luo metadatan koodista, jota ajonaikainen ympäristö käyttää etsiäkseen ja ladatakseen viittauksia eri komponentteihin, luokkiin, metodikutsuihin, luokkainstansseihin ajettavassa koodissa. Metadatan avulla CIL-koodi tulkataan ohjelmaa ajettaessa prosessorille sopivaksi natiivikoodiksi (native code). Kuten edellä todettiin, ajonaikaisen ympäristön roskien keruu (Garbage Collector) huolehtii muistinkäytöstä, viittauksista olioihin ja muihin komponentteihin, ja vapauttaa ne, kun niitä ei enää tarvita. Oliot, joiden elinkaarta hallitaan näin, kutsutaan managed dataksi. Kuva 7 havainnollistaa .NET-ohjelman kääntämistä. [Microsoft, Common Language Runtime (CLR), 2012; Microsoft, .NET Framework Conceptual Overview, 2012.]



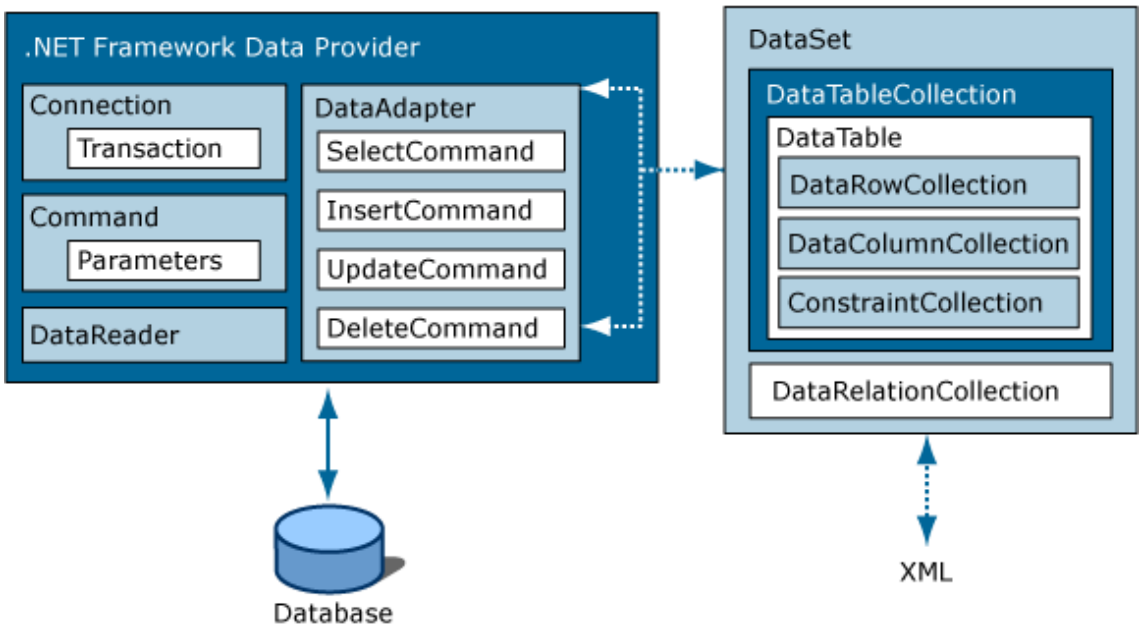
Kuva 7. .NET ohjelman käännös ja ajaminen

3.2 ADO.NET-kehys

ADO.NET on osa .NET-sovelluskehystä, ja se tarjoaa kirjastot tietokantapalveluiden toteuttamiseen .NET-sovelluksissa. ADO.NET tarjoaa tapoja käsitellä dataa erilaisista tietokannoista ja XML-tiedostoista. [Microsoft, ADO.NET Overview, 2013.]

3.2.1 ADO.NET-arkkitehtuuri

ADO.NET on arkkitehtuuriltaan monikerroksinen, ja se voidaan karkeasti jakaa kahteen eri osaan, ”Data Providers” ja ”Data Sets”. .NET Data Providerit ovat komponentteja ,jotka on suunniteltu datan manipulointiin ja nopeaan datan lukuun [Microsoft, ADO.NET Architecture, 2013.]. ADO.NET pyörii muutaman ydinkonseptin ympärillä, kuten Connection, Command ja DataSet-luokat ja niiden instanssit. ADO.NET ei sisällä yleiskäyttöistä ”data provider” -luokkaa, vaan se sisältää erityyppisiä data provider -luokkia, jotka kukin on suunniteltu erityyppisille tiedonlähteille (data source). Jokaisella data provider -luokkainstanssilla on oma toteutus *Connection, Command, DataReader ja DataAdapter* -luokasta, jotka on optimoitu kullekin tietokantahallintajärjestelmälle sopivaksi. Esimerkiksi, jos pitää luoda yhteys SQL Server -tietokantaan, käytetään Connection -luokkaa nimeltä SqlConnection. [MacDonald ym, 2010, 278.]



Kuva 8. ADO.NET -arkkitehtuuri [Microsoft, ADO.NET Architecture, 2013.]

Connection-luokka tarjoaa yhteyden tietokantaan. *Command*-luokka mahdollistaa pääsyn tietokannan komentoihin joiden avulla muokataan tietoa, ajetaan proseduureja ja lähetetään tai vastaanotetaan parametritietoja. *DataReader*-luokka tarjoaa nopean tavan vastaanottaa ja lukea dataa tietokannasta. *DataAdapter* tarjoaa sillan *DataSet*-luokkien ja tiedonlähteen (data source) välille. *DataAdapter*-luokkainstanssit käyttävät *Command*-luokkainstansseja suorittaakseen SQL-komentoja tietokannassa, joiden avulla asetetaan dataa *DataSet*:hin ja heijastetaan mahdolliset muutokset tietokantaan. [Microsfot, ADO.NET Architecture, 2013.]

DataSet-luokat ovat suunniteltu nimenomaan tiedon tallentamiseen ja käsittelemiseen riippumatta tiedonlähteestä, joka voi olla Sql- tai xml-tietokanta tai xml-tiedosto. *DataSet*-luokan ilmentymä sisältää kokoelman yhden tai useampia *DataTable*-luokkainstansseja, jotka sisältävät rivi ja saraketietoja sekä primääri- ja vierasavain tietoja. [Microsfot, ADO.NET Architecture, 2013.]

3.2.2 Yhteyden muodostaminen tietokantaan ADO.NET:in avulla

Kuten edellä todettiin, käytetään tietokantayhteyden luomiseen *Connection*-luokkien ilmentymiä. Ennen kuin voidaan työskennellä tietokannan kanssa, esimerkiksi lukea tietoa tai muokata tietoa, täytyy yhteys tiedonlähteeseen (”Data Source”) muodostaa.

Keskeisimmät tietokantayhteyden luomiseen tarvittavat ominaisuudet ja metodit on määritelty *IDbConnection*-rajapinnassa, jonka jokainen *Connection*-luokka toteuttaa. [MacDonald ym, 2010, 283.]

Kun *Connection*-luokasta luodaan instanssi, täytyy sille välittää ”connection string”, joka on yhteyden merkkijono, johon on sisällytetty nimi/arvo-asetuksia eroteltuna puolipisteellä (;). Merkkijonossa isoilla ja pienillä kirjaimilla on väliä, mutta asetusten järjestyksellä ei. Yhdessä asetukset muodostavat perustiedot, joita yhteyden muodostamiseen tarvitaan. Siitä huolimatta, että yhteyden merkkijonot vaihtelevat riippuen tietokannanhallintajärjestelmästä ja minkälaista data provideria käytetään, muutamat tiedot ovat lähes aina vaadittuja tietoja kuten: palvelimen nimi, jossa tietokanta sijaitsee, tietokanta, jota halutaan käyttää sekä autentikointitiedot. Koska tämän työn sovellus käyttää tietokantanaan Microsoft Sql Server -tietokantaa, on tässä työssä käytetty SqlConnection -tyyppistä data provider -luokkaa. SqlConnection on tarkoitettu Sql Server tietokannoille. [MacDonald ym, 2010, 282–283.]

```
String connectionStr = "Data Source=palvelin;Initial Cata-
log=tietokanta;User
Id=user;Password=passwd;Trusted_Connection=true;";
SqlConnection connection = new SqlConnection(tyokaluStr);
connection.Open();
```

Esimerkkikoodi 1. Connection stringin muodostaminen sekä SqlConnection luokkainstanssin luominen muodostetulla connection stringillä.

3.2.3 Komentojen suorittaminen ADO.NET:in avulla

Ennen kuin voidaan suorittaa komentoja, täytyy valita komennon tyyppi (*command type*) ja asettaa komentoteksti (*command text*) sekä sitoa komento *Connection*-luokan ilmentymään. Komennon tyyppi asetetaan antamalla arvot *CommandType*, *CommandText* ja *Connection*, ominaisuuksille. Komentoteksti (*CommandText*) voi olla Sql-lause, tietokannan proseduuri tai tietokannan taulun nimi. Tämä riippuu siitä, minkä tyyppistä komentoa käytetään. *Command*-luokalla on kolme eri enumeroitua tyyppiä: CommandType.Text, *CommandType.StoredProcedure* ja *CommandType.TableDirect*.

CommandType.Text -komento suorittaa Sql-lauseen, joka on välitetty *CommandText* -parametrissa. *CommandType.StoredProcedure* suorittaa tietokantaproseduurin, jonka

nimi on välitetty *CommandText* -parametrissa. *CommandType.TableDirect* ajaa kyselyn kaikista tietokannan taulun tietueista. Taulun nimi välitetään *CommandText* -parametrissa.

Command-luokka tarjoaa kolme eri metodia, joiden avulla voidaan suorittaa haluttu komento. Nämä metodit ovat *ExecuteNonQuery()*, jonka avulla suoritetaan insert, update ja delete Sql-komentoja. *ExecuteNonQuery()* palauttaa kokonaisluvun, joka kertoo, kuinka moneen riviin komento kohdistui tietokannassa. *ExecuteScalar*() –metodin avulla suoritetaan select-kysely tietokantaan, joka palauttaa ensimmäisen kentän ensimmäisen rivin arvon, jonka kysely luo. Tätä metodia käytetään usein Sql-funktioiden *Count()* tai *Sum()* kanssa. *ExecuteReader()* suorittaa Select-kyselyn tietokannassa, ja se palauttaa *DataReader*-luokan ilmentymän, johon on talletettu kursorimuodossa kyselyn tulos vain lukutilassa. [MacDonald ym, 2010, 290-291.]

3.2.4 Tiedon lukeminen ja tallettaminen ADO.NET:n avulla

Tiedonlähteestä saatava tieto voidaan tallettaa .NET:n tarjoamiin erilaisiin listoihin ja taulukoihin. Yksi ADO.NET:n tarjoama kokoelmatyyppi on *DataSet*-luokat, joita käytetään DataAdaptereiden kanssa. *DataSet*-luokan ilmentymät sisältävät kaksi tärkeää osaa: kokoelman 0:sta tai useammasta taulusta ja kokoelman 0:sta tai useammasta yhteydestä joita voidaan käyttää taulujen linkittämiseen. Kuviossa 4 on hahmotettu *DataSet*-luokan rakennetta.

ADO.NET:ssä tiedon lukemiseen voidaan käyttää *DataReader*-luokkaa. *DataReader*-luokka instantioidaan usein käyttämällä *Command*-luokan metodia *ExecuteReader(),* joka palauttaa DataReader-luokan instanssin. *DataReader*-luokan avulla luetaan dataa yksi rivi kerrallaan, joka on saatu SELECT-komennolla. *DataReader*-luokan ilmentymän käyttäminen on yksinkertaisin tapa saada data ulos datan lähteestä. DataReaderluokan yksi keskeisimmistä metodeista on *Read(),* jonka avulla siirretään kursori seuraavalle riville datavirrassa. *Read()*-metodi palauttaa boolean tyyppisen arvon siitä, onko datavirrassa seuraavaa riviä luettavaksi. [MacDonald ym, 2010, 291.]

```
SqlCommand command = new SqlCommand("SELECT id, nimi FROM
dbo.tyokalu_ryhma", connection);

SqlDataReader reader = command.ExecuteReader();
```

```
List<String[]> tulokset = new List<String[]>();

while (reader.Read())
{
          tulokset.Add(new String[] { reader[0].ToString(),
reader[1].ToString() });
}
```

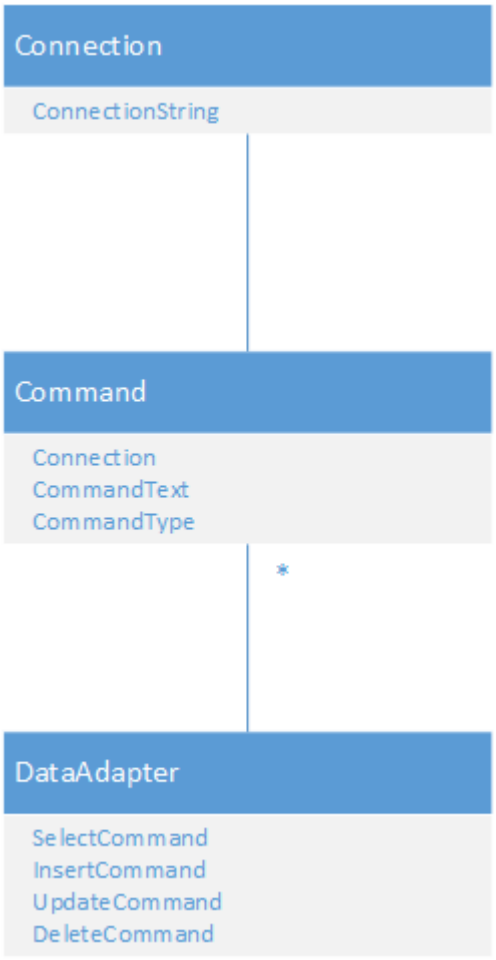
Esimerkkikoodi 2. SqlDataReaderin käyttö C#-koodissa.

Toinen mahdollinen tapa lukea tieto on käyttää *DataAdapter*-luokkia. Tässä työssä on käytetty *SqlDataAdapter*-luokan ilmentymiä, jotka on tarkoitettu käytettäväksi Sql Server -tietokannan kanssa. *DataAdapter*-luokkia käytetään purkamaan tietokannasta saadut tiedot *DataSet*-taulukokoelman tauluihin. Jokaisella data providerilla on oma dataadapterinsa, esimerkiksi *SqlDataAdapter* ja *OracleDataAdapter* jne. Kuten kohdassa 3.2.1 todettiin, *DataAdapter* toimii siltana yksittäisen *DataSet*in *DataTable*-ilmentymän ja datan lähteen välillä. Se sisältää tarvittavat komennot kyselyiden ja päivitysten suorittamiseen datan lähteellä. *DataSet*in täyttämistä DataAdapterin avulla tarvitaan määritellä *DataAdapter*-luokan ilmentymälle asettaa sen *SelectCommand*-ominaisuudelle arvo, joka on *Command*-luokan ilmentymä. *DataAdapter*-luokalla on neljä erilaista *Command*-tyyppistä muuttujaa: *InsertCommand*, *UpdateCommand*, *SelectCommand* ja *DeleteCommand*, joihin määritellään suoritettava komento. *DataAdapter*-luokalla on kolme keskeistä metodia: *Fill(),* jonka avulla lisätään *DataSetin DataTable*-ilmentymiin *SelectCommand*in tuottamat tulokset. Toinen on *FillSchema(),* joka alustaa *DataTable*-ilmentymät vastaamaan tietokannan tauluja, kuten tiedot sarakkeiden nimistä, tietotyypeistä ja primääriavaimista. Kolmas metodi on *Update(),* joka tutkii kaikki *DataTable*-instanssien muutokset ja tekee muutokset datan lähteelle suorittamalla sopivat *InsertCommand*, *UpdateCommand* ja *DeleteCommand* operaatiot. [MacDonald ym, 2010, 337-338.]

```
DataSet ds = new DataSet();
SqlCommand command = new SqlCommand("SELECT * FROM
dbo.kulutus_varaosa", connection);
SqlDataAdapter sda = new SqlDataAdapter(command);
sda.Fill(ds);
```

Esimerkkikoodi 3. DataSet-luokan ja SqlDataAdapterin käyttö C#-koodissa

Kuva 9 pyrkii havainnollistamaan, kuinka *DataAdapter* on yhteydessä tiedonlähteeseen (data source). Yhteen *DataAdapter*-luokkainstanssiin voidaan liittää useita *Command*-luokan instansseja. [MacDonald ym, 2010, 338.]

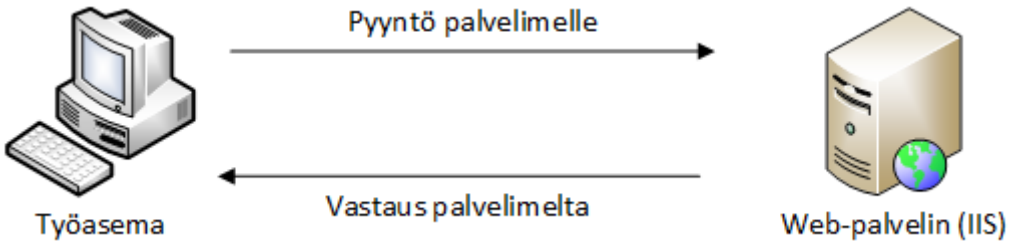


Kuva 9. Connection-, Command- ja DataAdapter-luokkien yhteistoiminta

3.3 ASP.NET-sovelluskehys

ASP.NET-sovelluskehys on osa .NET-sovelluskehystä, jolla voidaan kehittää dynaamisia Web-sovelluksia, -sivuja sekä -palveluja. Koska ASP.NET on osa .NET-sovelluskehystä. ASP.NET-sovellukset ajetaan CLR:ssä, eli toisin sanoen myös ASP.NET web-sivustojen ajettava koodi on managed codea ja käytössä on .NET:n tarjoamat kirjastot. Kuten muutkin web-sivustot ja sovellukset, ASP.NET:llä tuotetut sivustot ajetaan web-palvelimella, ASP.NET-sivustot eivät poikkea käyttäjän näkökulmasta mitenkään muista

Web-sivustoista. ASP.NET-sivu avataan verkkoselaimella. Työasemalla ajettava verkkoselain lähettää pyynnön web-palvelimelle, ja palvelin palauttaa html-merkkausta sisältävän sivun, jonka selain näyttää käyttäjälle. Yleisesti käytetty web-palvelin ASP.NET-sivustojen isännöimiseen (hosting) ja ajamiseen on Internet Information Services (IIS). IIS on kehitetty Windows-käyttöjärjestelmille. Vaihtoehtoinen ratkaisu IIS:lle olisi ollut käyttää apache web -palvelinta ja monoa, joka on avoimen lähdekoodin projekti .NET-ohjelmien tuottamiseen ja ajamiseen [Troelsen, 2010, 1564.]. Tässä työssä on päädytty käyttämään IIS-palvelinta, koska toimeksiantajalla oli käytössään IIS-pavelinympäristö. Jatkossa mainittaessa web-palvelimesta, tarkoitetaan IIS web-palvelinta.



Kuva 10. ASP.NET-sivun avaus työasemalta

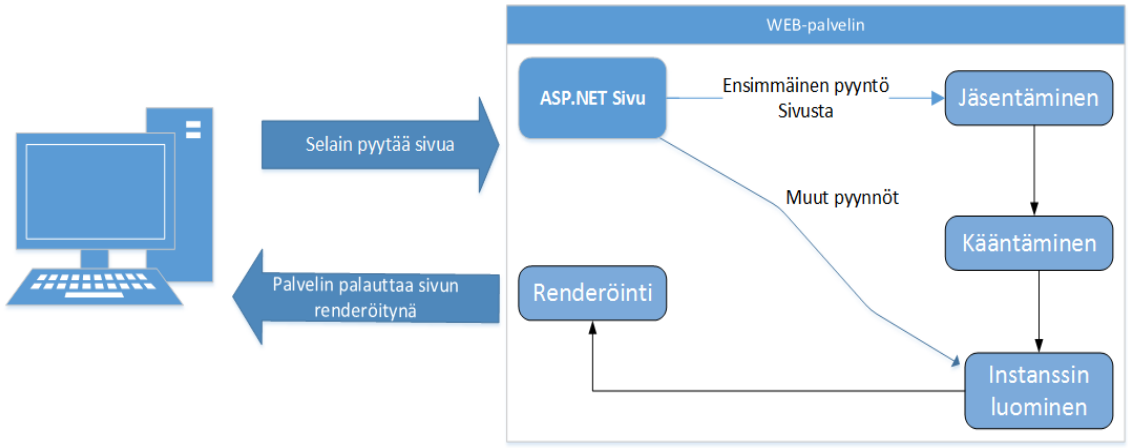
ASP.NET-sovellukset ajetaan siis web-palvelimella, mikä tarkoittaa sitä, että ohjelmakoodikin ajetaan palvelimella käyttäen palvelimen resursseja. Myös ohjelman käyttämä .NET-ohjelmistokehys ja CLR, jossa ohjelmaa ajetaan, sijaitsee palvelimella. Tätä ohjelmointitapaa kutsutaan ”server side programmingiksi”. Tämän ansiosta ei käyttäjällä itse tarvitse olla .NET-frameworkia asennettuna omalle työasemalleen ASP.NET-sivustojen näyttämiseen verkkoselaimella.

ASP.NET-sivustojen luomista voidaan lähestyä kahdella eri tavalla. Vanhempi malli, on Web Forms. Uudempi malli on Microsoftin ASP.NET MVC Framework, joka toteuttaa yleisesti käytetyn MVC-ohjelmistoarkkitehtuurimallin. Tässä työssä on käytetty Web Forms -mallia. Toimeksiantajalla on käytössään jo useita Web Formsiin perustuvia web-sovelluksia, joten sen käyttäminen tässä työssä oli luonteva ja tehokas ratkaisu.

ASP.NET:llä toteutetut sivut näkyvät internetselaimessa. Sivun lähdekoodia tutkittaessa html:nä, jossa toiminnot suoritetaan JavaScript-koodina. ASP.NET-sivujen tiedostopääte on ”.aspx”.

3.3.1 ASP.NET-ohjelman kääntäminen

ASP.NET-sivu käännettään palvelimella, kun asiakas pyytää sitä ensimmäistä kertaa. Tällöin sivu käy läpi seuraavat vaiheet, joita ovat jäsentäminen, kääntäminen ja uuden instanssin luominen. Sivun ollessa jo kertaalleen käännetty luodaan pyynnöstä uusi instanssi. Instanssin luomisen jälkeen sivu käännetään (renderöidään) verkkoselaimelle sopivaksi. Web-palvelin tarkkailee lähdekoodin tiedostoja mahdollisten muutosten varalta ja suorittaa uuden käännöksen, jos muutoksia on tehty. [Bochicchio ym, 2011, 3-9.]



Kuva 11. ASP.NET sivun kääntäminen. [Bocchichio ym, 2011, 6.]

3.3.2 ASP.NET Web Forms-malli

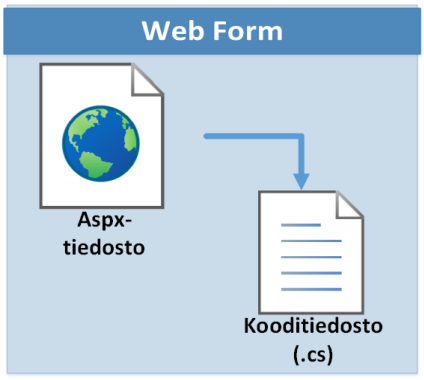
Kuten edellä todettiin, yksi lähestyttävä tapa luoda yksittäisiä web-sivuja ASP.NET:llä on Web Forms, joka on Microsoftin 2000-luvun alussa kehittämä teknologia luoda dynaamisia web-sivuja. Yksi Web Form kuvastaa yhtä web-sivua. Web Forms:iin pohjautuvia ASP.NET-sivuja voidaan luoda Visual Studiolla kahdella eri tavalla. Yksi vaihtoehto on luoda oma projekti ASP.NET-sivulle ja toinen vaihtoehto on luoda ns. ”Web Site”. Suurin ero näiden vaihtoehtojen välillä on se, että ”Web Site”-tyyppisessä sivussa jokaisella sivulla on oma käännöksensä, kun taas projektityyppisessä sivussa luodaan käännös koko projektista. Näin ollen ”Web Site” tyyppisellä ASP.NET-sivulla jokainen yksittäinen sivu käy läpi kuvan 14 vaiheet, kun taas projektityyppisellä ASP.NET-sivulla tehdään käännös koko sivustosta kuvan 14 osoittamalla tavalla.

Web form on yksi ASP.NET:n tarjoamista käyttöliittymäkomponenttivaihtoehdoista, joka koostuu aspx-tiedostosta, joka sisältää html-merkkausta ja siihen liittyvästä koodi-tiedostosta. Aspx-tiedosto, tai aspx-sivu, kuvastaa näytettävää html-sivua.

```
<asp:Content runat="server" ID="FeaturedContent"
ContentPlaceHolderID="FeaturedContent">
    <section class="featured">
        <div class="content-wrapper">
            <hgroup class="title">
                <h1><%: Title %>.</h1>
                <h2>Modify this template to jump-start your
ASP.NET application.</h2>
            </hgroup>
            <p>
                To learn more about ASP.NET, visit <a
href="http://asp.net" title="ASP.NET Web-
site">http://asp.net</a>.
                The page features <mark>videos, tutorials, and
samples</mark> to help you get the most from
                ASP.NET. If you have any questions about ASP.NET
visit
                <a href="http://forums.asp.net/18.aspx" ti-
tle="ASP.NET Forum">our forums</a>.
            </p>
        </div>
    </section>
</asp:Content>
```

Esimerkkikoodi 4. Aspx-tiedoston sisältämä, html-merkkausta muistuttava koodi.

Kooditiedoston tiedostopääte, esimerkiksi käytettäessä C#-ohjelmointikieltä, on aspx.cs. Kooditiedostoja kutsutaan ”code-behind” tai ”code-beside” -tiedostoiksi Visual Studion projektin tyypistä (”Web site” tai Web projekti) riippuen. [MacDonald ym, 2010, 5; Bocchichio, 2011, 6.]



Kuva 12. Web Form:in käsittämät tiedostot

Aspx-sivu kostuu html-koodin lisäksi ns. ”server controlleista”, joita on html-koodin seassa. Jokaisesta kontrollista (”server control”) muodostetaan oma olioinstanssi, jota voidaan hallita kooditiedostossa. Kooditiedoston avulla saadaan .NET-sovelluskehyksen ohjelmointiominaisuudet käyttöön sivulla. Aspx-tiedostoon on myös mahdollista upottaa ohjelmakoodia, mutta näin saadaan helposti aikaan huonosti luettavaa sekä ylläpidettävää koodia. [Bochicchio ym, 2011, 6-7.]

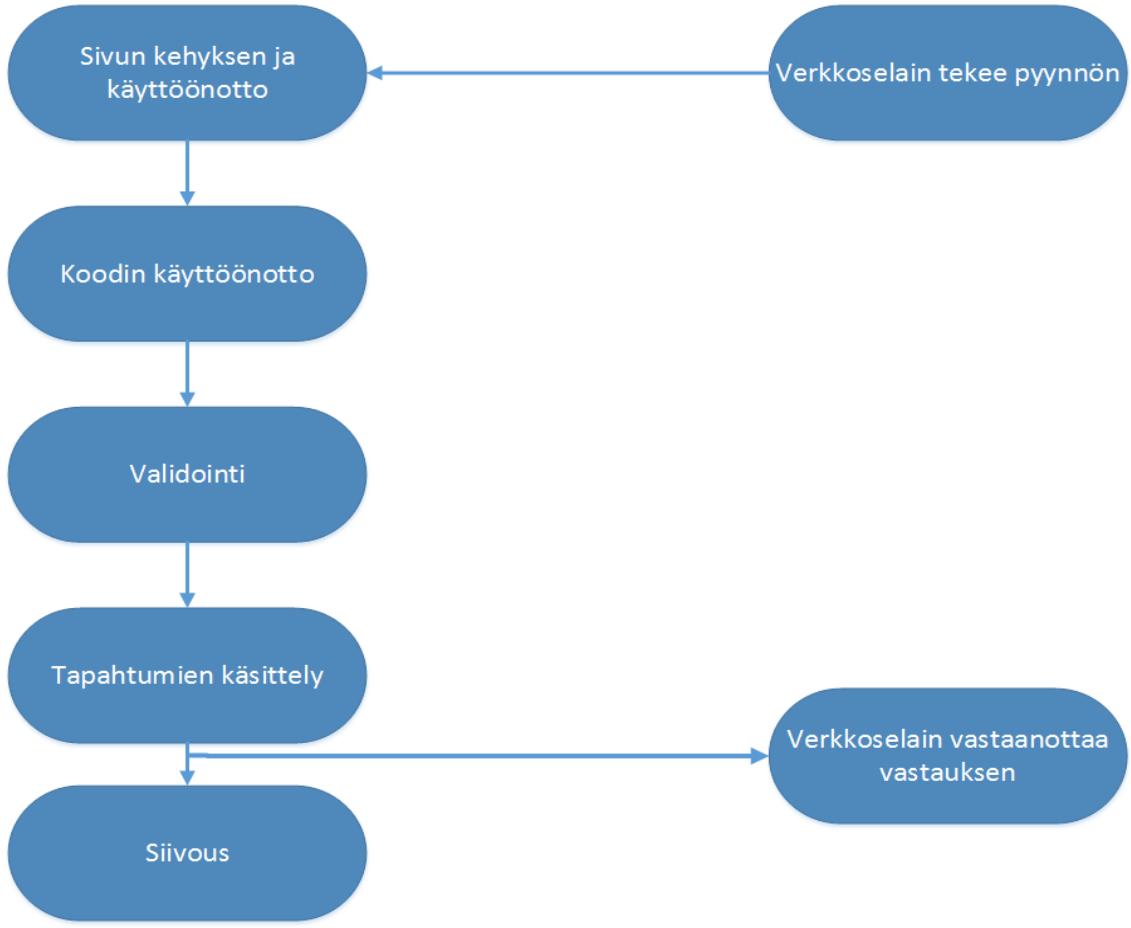
Web Form perustuu renderöintiin. *Web Form* on itsessään ns. kantakontrolli, jonka kääntäjä käsittelee omana kokonaisuutena. Sen sisältö generoidaan käyttäen *Render*-metodia. Tätä metodia kutsutaan rekursiivisesti ja se on jaettu kaikkien kontrollien kanssa, joten jokainen sivun osa renderöidään. ASP.NET sivu käsittää tapahtumia, joita tarvitaan sivun käsittelemiseen sekä sen olioinstanssien käsittelemiseen koodissa. *OnLoad*-tapahtuma on usein tarvittava tapahtuma ja tämän tapahtuman yksinkertaistamiseen ASP.NET määrittelee erityisiä tapahtumakäsittelijöitä joissa on käytetään etuliitettä *Page_*. Näitä metodikutsuja kutsutaan automaattisesti. Näistä useimmin tarvittavat tapahtumat ovat *Page_Load*, joka nostetaan, kun sivu ja sen kontrollit ovat valmiina käyttöön; Page_Init, jota kutsutaan, kun luokka joka on yhteydessä sivuun, on ladattu; *Page_LoadComplete*, joka nostetaan, kun *Page_Load*-tapahtuman käsittely on päättynyt; *Page_PreRender* on viimeinen tapahtuma, joka nostetaan ennen kuin ASP.NET renderöi sivun sisällön. [Bochicchio ym, 2011, 14–15.]

Web-palvelimella, ASP.NET Web Formin käsittely tapahtuu vaiheittain. Tämä mahdollistaa sivun liittämisen prosessointiin, missä vaiheessa tahansa sekä pystytään muokkaamaan sitä, kuinka sivu käyttäytyy. Tärkeimmät ASP.NET-sivun prosessointivaiheet ovat:

- sivun kehyksen käyttöönotto
- koodin käyttöönotto
- validointi
- tapahtumien käsittely

- automaattinen datan sidonta (data binding)
- roskienkeruu.

Nämä vaiheet tapahtuvat jokaisella kerralla, kun selain tekee pyynnön sivusta. Havainnollistava kuvio sivun prosessoinnista on kuvassa 13. [MacDonald ym, 2010, 97–98.]



Kuva 13. ASP.NET-sivun prosessointi ja elinkaari (MacDonald ym, 2010, 98.)

3.3.3 Server-kontrollit

ASP.NET server -kontrollit (“server controls”) ovat olennainen osa ASP.NET-arkkitehtuuria. Server-kontrollit ovat .NET-luokkia, jotka havainnollistavat visuaalisia elementtejä Web Formissa. Osa näistä luokista on suoraviivaisia ja kuvastaa tiettyä

HTML-elementtiä. Toiset taas ovat paljon suurisuuntaisempia abstraktioita, jotka kuvastavat useaa HTML-elementtiä. Erilaisia server control -tyyppejä ovat *Html server controls, Web controls, Rich Controls, Validation Controls, Data Controls, Navigation controls, Login controls, Web parts controls, ASP.NET ajax controls ja ASP.NET Dynamic Data controls*. Tässä työssä on hyödynnetty Web Controls- ja Data Controls -kontrolleja. [MacDonald ym, 2010, 129–130.]

Web-kontrolliluokat (*web controls*) on jatkettu html-elementeistä, jotka sisältävät joukon ominaisuuksia ja metodeita, joiden avulla niiden tilaa voidaan muokata koodissa. Joitakin esimerkkejä ovat *HyperLink*-, *ListBox*- ja *Button*-kontrollit. Lisäksi on useita muita ASP.NET-kontrolleita, jotka määritellään web-kontrolleiksi. [MacDonald ym, 2010, 130.]

Jokainen web-kontrolli on määritelty *System.Web.UI.WebControls*-nimiavaruudessa ja ne on periytetty *WebControl*-luokasta, joka tarjoaa käsitteellisemmän ja johdonmukaisemman mallin kuin HTML server -kontrollit. Web kontrollit mahdollistavat mm. lisäominaisuudet kuten automaattiset *postback*-toiminnot, jonka avulla kontrollit lähettävät tietoa web-palvelimelle. [MacDonald ym, 2010, 143.]

3.3.4 Tiedon sidonta

Tiedon sidonta on ominaisuus, jonka avulla voidaan näyttää sovelluksissa dataa siihen liitetystä tiedonlähteestä (data source). Esimerkiksi käyttöliittymän komponenttiin on mahdollista liittää muuttuja tai muuttujia, jotka sidotaan haluttuun tiedonlähteeseen esimerkiksi ruudukko, joka on sidottu tietokannan näkymään. Olennaista tiedon sidonnassa on se, että se on deklaratiivista eikä ohjelmallista. Tämä tarkoittaa sitä, että datan sidonta voidaan määritellään koodin ulkopuolella mukana kontrollien määrittelyssä .aspx-sivulla. [MacDonald ym, 2010, 354.]

ASP.NET:ssä useimmat web-kontrollit tukevat yhden arvon datan sidontaa (single-value data binding). Yhden arvon sidonnalla voidaan sitoa kontrollin ominaisuus tiedonlähteeseen, mutta kontrolli voi näyttää vain yhden arvon. Sidottavan omaisuuden ei tarvitse kuvastaa mitään näkyvää sivulla. Useat web-kontrollit tukevat usean arvon sitomista (repeated value binding), mikä tarkoittaa sitä, että ne voidaan renderöidä

näyttämään joukollista dataa. Erilaiset taulukot ja listat käyttävät hyödykseen tämän tyyppistä sidontaa. [MacDonald ym, 2010, 354.]

3.3.5 Data-kontrollit

Data-kontrollijoukko sisältää kehittyneitä taulukoita ja listoja, jotka on suunniteltu esittämään suurta määrää dataa sisältäen tuen eri esitystyyleille, muokkaukselle, lajittelulle ja sivutukselle. Data-kontrollit käyttävät hyödykseen tiedon esittämistä varten datan sidontaa. Tämä joukko sisältää myös datan lähteeseen liittyviä kontrolleja joiden avulla voidaan sitoa datan lähteet deklaratiivisesti ilman, että tarvitsisi kirjoittaa ylimääräistä koodia. [MacDonald ym, 2010, 130.]

Data-kontrolli säilöö tarpeeksi informaatiota näyttääkseen vain sen datan, mikä on sillä hetkellä näkyvissä. Jos tarvitaan vuorovaikuttaa DataSetteihin käyttäen useita postbackejä, tarvitsee ne säilöä *ViewState*-, *Session*- tai *Cookie*-kokoelmaan käsin (joka suurentaa sivuston kokoa huomattavasti) [MacDonald ym, 2010, 340.]

Yksinkertaisten listakontrollien lisäksi ASP.NET sisältää muutamia ns. Rich Data -kontrolleja, jotka tukevat ”repeated value binding” -sidontaa. Rich data -kontrollit ovat hyvin erilaisia kontrolleja verraten yksinkertaisiin listakontrolleihin. Esimerkiksi ne ovat suunniteltu nimenomaan datan sitomiseen (data binding). Niillä on myös kyky näyttää useita ominaisuuksia tai kenttiä kustakin data itemistä usein taulukko-pohjaiseen malliin, joka tukee korkeamman tason ominaisuuksia kuten muokkausta. [MacDonald ym, 2010, 366.]

Rich data -kontrollit sisältävät *GridView*-kontrollin. *GridView* on yleiskäyttöinen taulukko-kontrolli, jonka avulla näytetään suuria taulukkoja. Se tukee valintaa, muokkausta, lajittelua ja sivutusta. [MacDonald ym, 2010, 366.]

Ymmärtääksemme, kuinka data-kontrollit (data controls) toimivat, täytyy tietää kuinka ne liittyvät sivun elinkaareen. Tämä tieto on tärkeää silloin, kun joudutaan tilanteisiin, joissa täytyy laajentaa tiedon sidontaa. Nämä vaiheet tapahtuvat sivua pyydettäessä. Sivusta luodaan olioinstanssi perustuen .aspx-tiedostoon, sivun elinkaari alkaa ja *Page.Init* ja *Page.Load* -tapahtumat laukeavat, kaikkien muiden kontrollien tapahtumat laukeavat, data source -kontrollit suorittavat mahdolliset päivitykset, *Page.PreRender*

tapahtuma laukeaa, data source -kontrollit suorittavat mahdolliset kysely- ja tallennusoperaatiot haettuun dataan niihin linkitettyihin kontrolleihin, sivu renderöidään, jonka jälkeen se joutuu roskien keruuhun. [MacDonald ym, 2010, 369.]

4 CoolTool-sovelluksen tekninen arkkitehtuuri

Edellisessä luvussa käytiin läpi .NET-sovelluskehyksen tarjoamia kirjastoja, joita on hyödynnetty tämän CoolTool-sovelluksen tekemisessä. Tässä luvussa perehdytään CoolTool-sovelluksen tekniseen toteutukseen.

CoolTool on Norpen työkaluvaraston hallintaa varten kehitetty web-selaimessa toimiva sovellus, joka hyödyntää tietokantanaan Microsoft SQL Server 2005:llä rakennettua tietokantaa.

4.1 CoolTool-sovelluksen toteutus .NET-sovelluskehyksellä

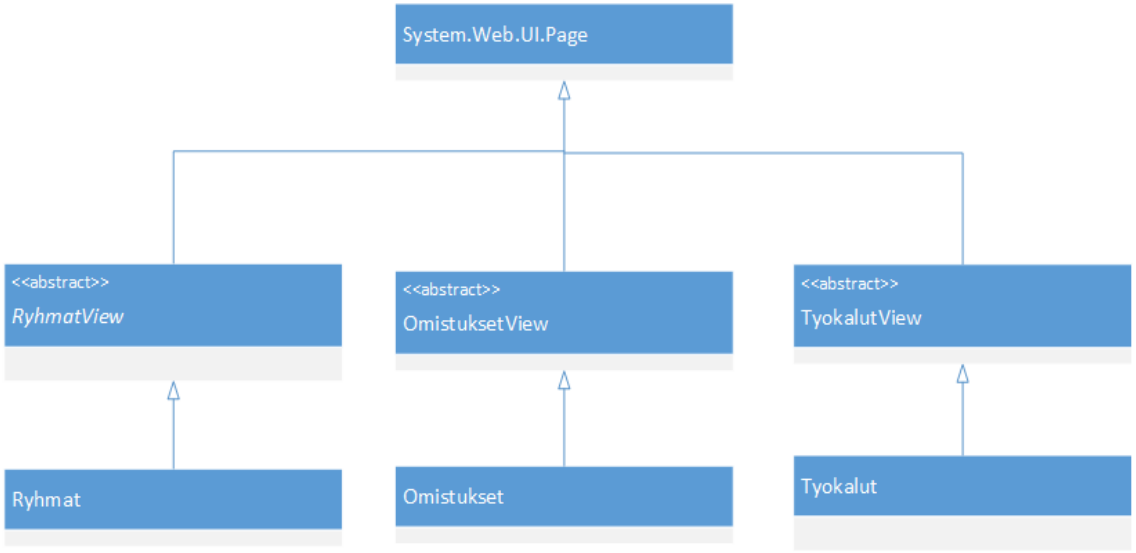
CoolTool-sovelluksen käyttöliittymä ja suurin osa bisneslogiikasta on toteutettu .NET-sovelluskehyksen avulla. CoolTool-sovellus käyttää hyväkseen .NET:n perusluokkakirjastoja sekä .NET:n sisältämiä ASP.NET- ja ADO.NET-sovelluskehyksiä. Sovelluksen käyttöliittymä on verkkoselaimessa toimiva Web Formsilla toteutettu web-sivusto. Sovellus on toteutettu Visual Studio 2010:llä, ja Visual Studion projekti on Web Site. Sovelluksen käyttöliittymäarkkitehtuurissa on hyödynnetty MVC-mallia.

Tietokannan käsitteitä vastaamaan on mallikerrokseen (model) tehty POCO-luokkia (Plain Old CLR Object), jotka kuvastavat tietokannan tauluja. Lisäksi mallikerros sisältää DAO-luokan, jonka avulla tehdään tietokantaoperaatiot sekä Validator-luokan, jossa tehdään entiteeteille (POCO-luokille) tarkistuksia. Käyttöliittymäkerroksessa on yhteensä kolme aspx-sivuista muodostuvaa näkymää (views) sekä jokaisella kolmella näkymällä tulostusnäkymä, johon tulostettavat sivut avataan. Lisäksi jokaisella kuudella käyttöliittymänäytöllä on oma kontrolleri-luokka, joka yhdistää luokat mallin välille. Kuvassa 14 on kaavio CoolTool-sovelluksen eri kerroksista ja niiden yhteyksistä.

Kuva 14. CoolTool-sovelluksen sovelluskerrokset

4.1.1 Web Forms -näyttöjen rakenne ja toiminta sovelluksessa

Kuviossa alla näkyvät kolme näyttöä: *RyhmatView, OmistuksetView ja TyokalutView* ovat itse asiassa abstrakteja luokkia. Nämä luokat toimivat yliluokkina sovelluksen, *Ryhmat, Omistukset, Tyokalut,* Web Forms -sivujen kooditiedostoille, joiden avulla aspx-sivuja hallitaan koodista käsin. Abstraktit näyttöluokat perivät ASP.NET:n Page-luokan, joka on oletuksena Web Formsien kooditiedostojen yliluokka. Abstraktit luokat toimivat siltana aspx-sivujen ja AppCoden välillä. AppCode on hakemisto, johon käyttöliittymän ulkopuoliset luokat ja luokkarakenteet luodaan Visual Studion Web Site -tyyppisessä projektissa.



Kuva 15. Luokkakaavio sovelluksen näyttöjen periytymisestä

Jokaista näyttöä ohjaa sille tarkoitettu kontrolleriluokka. Nämä kontrolleriluokat ottavat kiinni näytöillä tapahtuvat tapahtumat. Kontrolleriluokka toimi yhdessä tietokantakerroksen välillä, jonka avulla tietokannasta haetaan tiedot näytölle. Esimerkiksi jokaisesta napin painalluksesta syntyy *Postback*-tapahtuma, jonka kontrolleriluokka käsittelee. Tapahtumakäsittelyn lisäksi kontrolleriluokat tallentavat ja käsittelevät tietoa aspx-sivun sessioihin, joita tarvitaan tilan säilyttämiseen *Postback*-tapahtumien sattuessa. Sillä kuten kohdassa 3.3.2 todettiin, jokaisella pyynnöllä sivusta luodaan uusi instanssi ja vanha joutuu roskien keruuhun.

4.1.2 Tietokantaoperaatiot ja tiedon käsittely sovelluksessa

Sovelluksen olennainen osa on tietokanta. Sovellus näyttää ja käsittelee tietokannassa olevaa tietoa. Tiedonkäsittelyä varten sovelluksessa on oma kerros, jonka luokat hyödyntävät ADO.NET:n tarjoamia luokkia. Nämä luokat ovat *DAO* ja *Validator*. *DAO*-luokka suorittaa SQL-komentoja, joita tarvitaan tiedon hakemiseen ja muokkaamiseen. Validator-luokka sisältää joukon metodeita, joiden avulla tarkistetaan ja korjataan syötteitä sekä luodaan uusia POCO-luokkien ilmentymiä tallennusta varten.

DAO-luokan käyttämät keskeisimmät ADO.NET:n tarjoamat luokat ovat *DataSet*, *SqlConnection*, *SqlCommand* ja *SqlDataAdapter*. Suurin osa DAO-luokan metodeista, jotka kysyvät joukollista dataa tietokannasta, palauttavat datan *DataSet*iin talletettuna,

joka kuljetetaan näyttöluokalle. Yhteys tietokantaan muodostetaan käyttäen *SqlConnection*-luokkaa ja SQL-komennot suoritetaan *SqlCommand*-luokan ilmentymillä. Koodiesimerkissä 5 tieto täytetään *DataSet* instanssiin käyttäen *SqlDataAdapteria*.

```
public DataSet getKulVaraAll()
{
          DataSet ds = new DataSet();
          makeConnection();
          SqlCommand command = new SqlCommand("SELECT * FROM
dbo.kulutus_varaosa", connection);
          SqlDataAdapter sda = new SqlDataAdapter(command);
          sda.Fill(ds);
          closeConnection();
          return ds;
}
```

Esimerkkikoodi 5. DAO-luokan metodi, jossa haetaan tietoa tietokannasta käyttäen hyväksi DataSet-, SqlCommand-, SqlConnection- ja SqlDataAdapter-luokkia.

Sovelluksessa on siis kolme kerrosta, johon kuuluu erilaisia luokkia, jotka sisältävät erilaisia toimintoja. Eri sovelluskerroksiin kuuluvat luokat ovat:

- Tiedon näyttäminen: Näytöt, jotka on nimetty päätteellä ”View”. Edellä mainitut Web Forms -näytöt.

- Logiikka:

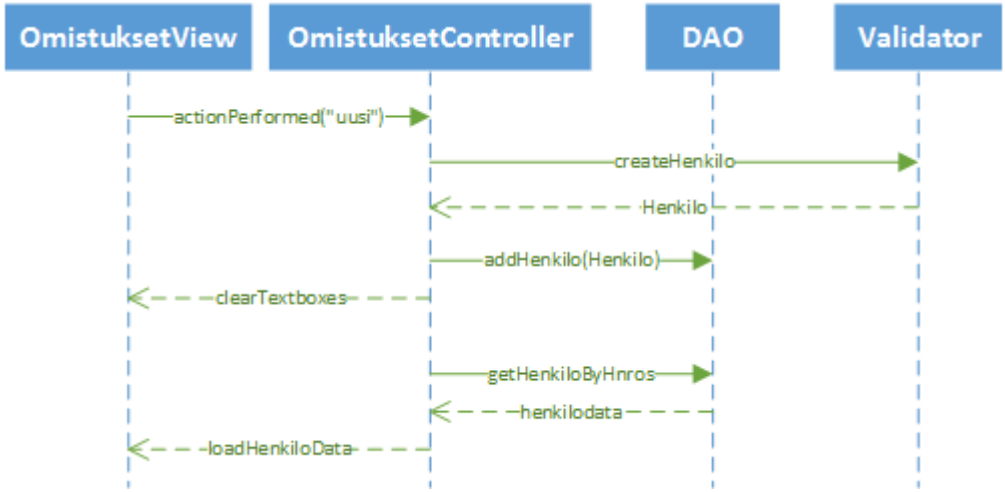
    - Kontrollerit, jotka on nimetty päätteellä ”Controller”. Jokaisella näytöllä on oma kontrolleri-luokka, joka kommunikoi tietokantakerroksen sekä Validator-luokan kanssa. Kontrolleriluokka vastaa myös pitkälti käyttöliittymälogiikasta.

    - Validator-luokka sisältää POCO-luokkien luonnin sekä joukon tarkistuksia POCO-luokille. Validator-luokka kommunikoi myös tietokantakerroksen kanssa.

- Tietokantakerros: Tietokantakerros sisältää yhden luokan DAO, jonka tehtävänä on suorittaa kaikki sovelluksen tietokantaoperaatiot, kuten haut, poistot ja muokkaukset.

4.1.3 Uuden työntekijän lisääminen

Kun omistukset näytöllä, jonka tehtävänä on näyttää työntekijät (henkilöt) ja heille kuuluvat työkalut, luodaan uusi työntekijä, *OmistuksetController* ottaa komennon vastaan ja pyytää *Validator*-luokkaa luomaan uuden henkilön, jonka jälkeen kontrolleriluokka lisää Validator-luokan luoman henkilön tietokantaan. Tämän jälkeen näytöllä tyhjennetään tekstikentät, joihin on syötetty uuden henkilön tiedot, ja haetaan henkilödata näytölle tietokannasta. Kuvassa 16 on kuvattu uuden työntekijän lisäys sekvenssikaaviona.

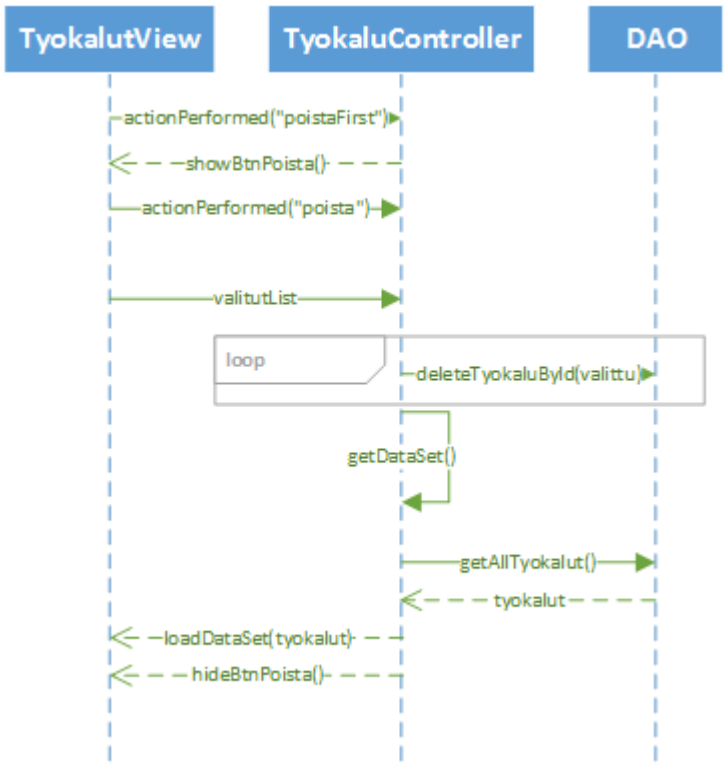


Kuva 16. Sekvenssikaavio uuden työntekijän lisäämisestä.

4.1.4 Työkalun poistaminen

Kun työkalunäytöllä halutaan poistaa yksi tai useampi työkalu, *TyokaluController* pyytää työkalunäyttöä näyttämään vahvistuspainikkeen, joka aloittaa työkalujen poiston. Kun vahvistuspainiketta painetaan, kontrolleriluokka pyytää näytöltä kaikki valitut työkalut, jotka halutaan poistaa. Tämän jälkeen käydään silmukassa yksitellen läpi silmukassa jokainen työkalu, joka on valitut työkalut listalla ja kutsutaan *DAO*-luokan pois-

tometodia. Poiston jälkeen pyydetään *DAO*-luokalta työkalut tietokannasta ja ladataan ne näytölle. Kuvassa 17 on havainnollistettu työkalujen poisto sekvenssikaaviona.



Kuva 17. Sekvenssikaavio työkalun poistamisesta

4.2 CoolTool-sovelluksen tietokanta

Sovelluksen keskeinen tekijä on sovelluksen tietokanta. Sovelluksen käsittelemä data on talletettuna SQL Server 2005 -tietokantaan. Sql Server 2005 on Microsoftin kehittämä relaatiotietokantojen hallintajärjestelmä. Yrityksellä oli jo entuudestaan käytössään SQL Server -tietokantoja, joten oli loogista, että CoolTool-sovellus käyttää tietokantanaan SQL Server -tietokantaa.

4.2.1 Tietokannan rakenne

Sovelluksen tietokanta on olennainen ja keskeinen osa sovellusta. Sovelluksen tietokannassa säilytään kaikki olennainen informaatio työkaluista, työntekijöistä ja niiden keskinäisistä suhteista. Tauluja sovelluksessa on seitsemän: *työkalu, henkilo, tyokalu_ryhma, henkkulu, kalibroitavat, poistetut_tyokalut* ja *vapaat_numerot*.

- *tyokalu*: Sisältää tiedon esimerkiksi työkalun nimestä, merkistä, ostopäivästä ja takuuajasta sekä tiedon, onko työkalu poistettu sovelluksesta. Sovellus näyttää vain työkalut, joilla on arvo 0 poistettu kentässä.

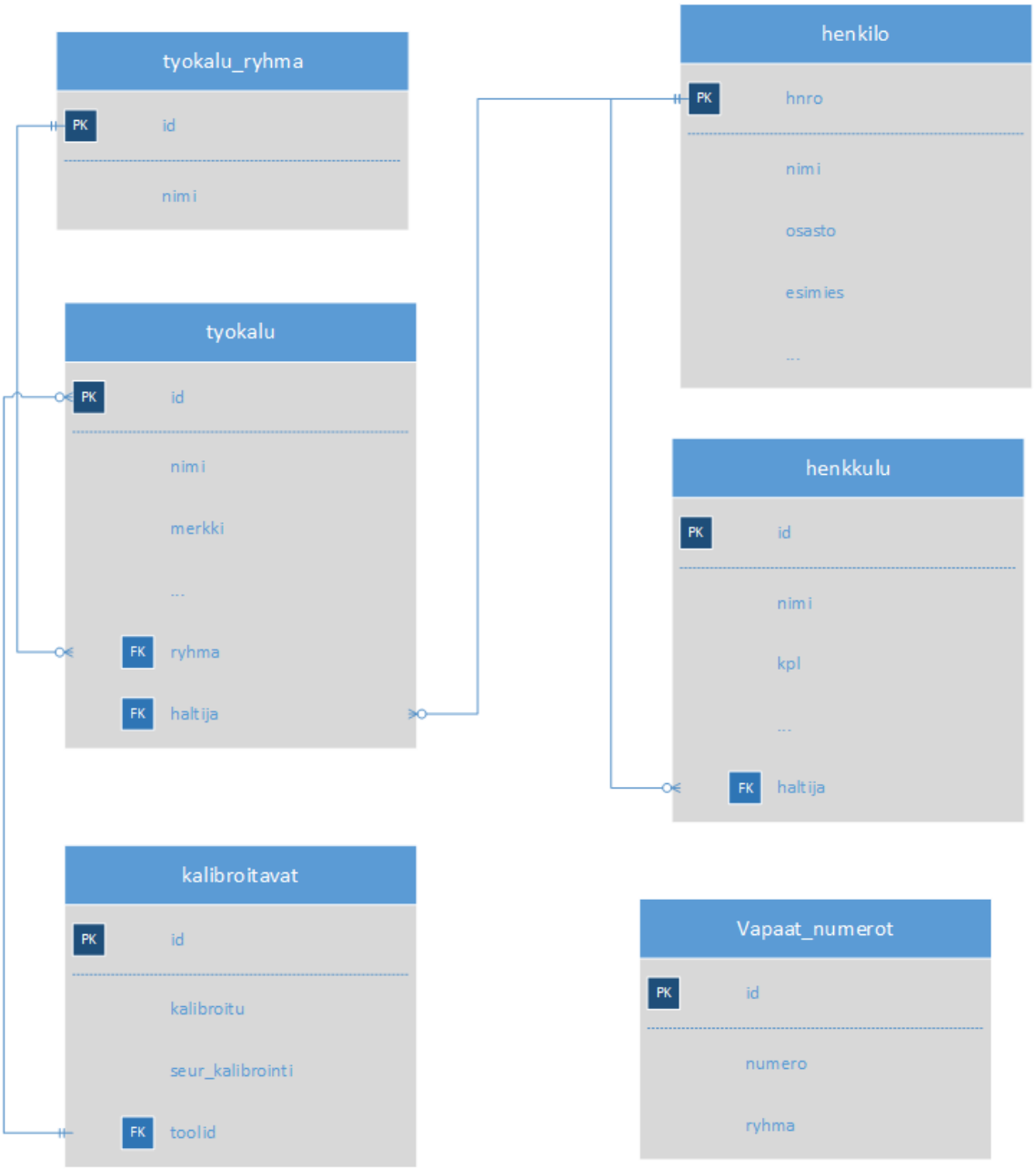
- *tyokalu_ryhma*: työkalujen ryhmittelyä varten, jota käytetään työkalujen numeroinnissa.

- *henkilo*: taulu, johon talletetaan työntekijöiden perustiedot: nimi, tehdas, esimies.

- *henkkulu*: henkilön kulutustuotteet ja niiden kappalemäärä.

- *kalibroitavat*: kalibroitavat työkalut. Sisältää tiedon suoritetun kalibroinnin päivämäärästä ja seuraavasta kalibroinnista.

- *vapaat_numerot*: jokaisen työkaluryhmän vapaat numerot. Sisältää identifioivan id-kentän sekä työkalun numeron ja työkalun ryhmän numeron.



Kuva 18. CoolTool-sovelluksen tietokannan rakenne

4.2.2 CoolTool-tietokannan automaattiset proseduurit

Triggerit ovat tietokannan suorittamia automaattisia proseduureja, jotka on kytketty suoriutumaan, kun tietty tapahtuma laukeaa tietokannassa. Tällaisiä tapahtumia voivat esimerkiksi olla *update*-, *insert*- tai *delete*-tapahtumat tietokanassa. Automaattiset proseduurit määritellään taulukohtaisesti, eli kun määriteltyyn tauluun tehdään yllämainit-

tuja *update-, insert-* tai *delete*-operaatioita. Esimerkkikoodi 6 on esimerkki automaattisen proseduurin luomisesta.

```
CREATE TRIGGER reminder2
ON Sales.Customer
AFTER INSERT, UPDATE, DELETE
AS
   EXEC msdb.dbo.sp_send_dbmail
        @profile_name = 'AdventureWorks2012 Administrator',
        @recipients = 'danw@Adventure-Works.com',
        @body = 'Don''t forget to print a report for the sales
force.',
        @subject = 'Reminder';
GO
```

Esimerkkikoodi 6. Automaattisen proseduurin luominen MS SQL -serverissä [Microsoft, Create Trigger (Transact-SQL), 2013.]

Vapaat_numerot -taulu sisältää siis työkalun numeron ja työkaluryhmän numeron, johon numero kuuluu. Tämä taulu on apuna ratkaisemassa työkalujen numerointiongelmaa, jossa työkaluille pitää antaa nelinumeroinen numero. Ongelmallista on se, että numeron ei pidä kasvaa jokaisella kerralla, kun tietokantaan lisätään uusi työkalu. Kun tietokannasta poistetaan työkaluja, lisätään työkalun sisältämä numero ja ryhmä vapaat_numerot -tauluun. Näin ollen nelinumeroinen numerointi riittää hyvin pitkälle, sillä numeroita voidaan kierrättää. Numeroiden lisääminen ja poistaminen on kytketty automaattisiin proseduureihin, jotka tarkkailevat työkalutauluun tehtäviä *delete-, update-* ja *insert* sql-komentoja.

Lisättäessä työkalua tyokalu-tauluun, annetaan työkalulle numero vapaat_numerot-taulusta, joka vastaa työkalun ryhmää. Lopuksi annettu numero poistetaan vapaat_numerot-taulusta.

```
ON [dbo].[tyokalu]

FOR INSERT

AS

DECLARE @ryhma int
DECLARE @numero int
DECLARE @id

SELECT @id = (SELECT id FROM Inserted)
```

```
SELECT @ryhma = (SELECT ryhma FROM Inserted)
SELECT @numero = (SELECT MIN(numero) FROM dbo.vapaat_numerot
WHERE ryhma = @ryhma)

UPDATE dbo.tyokalu SET numero = @numero WHERE id = @id
DELETE FROM dbo.vapaat_numerot WHERE ryhma = @ryhma AND numero =
@numero
```

Esimerkkikoodi 7. Automaattinen proseduuri, joka suoritetaan työkalun lisäämisen jälkeen

Poistettaessa työkalua CoolTool-sovelluksesta tietokantaan tehdään update-komento eikä delete-komentoa. Update-komento päivittää valittujen työkalujen poistettu-kentän arvon arvoon 1, jonka jälkeen automaattinen proseduuri, joka tarkkailee update-komentoja työkalutaulussa, suoritetaan. Esimerkkikoodissa 8 on automaattinen proseduuri, joka tarkkailee työkalutaulun update-komentoja. Automaattinen proseduuri tarkistaa, onko työkalu poistettu vai palautetaanko työkalu takaisin sovellukseen. Jos työkalun poistettu kentän arvo on 1, lisätään työkalun numero ja ryhmän numero vapaat_numerot-tauluun. Jos työkalun poistettu kentän arvo puolestaan on 0, niin annetaan työkalulle uusi numero *vapaat_numerot*-taulusta ja poistetaan numero ks. taulusta.

```
FOR UPDATE

AS

IF UPDATE (poistettu)
BEGIN
DECLARE @ryhma int
DECLARE @numero int
DECLARE @id int
DECLARE @poistettu bit

SELECT @id = (SELECT id FROM inserted)
SET @ryhma = (SELECT ryhma FROM inserted)
SELECT @poistettu = (SELECT poistettu FROM inserted)
SELECT @numero (SELECT numero FROM inserted)

IF (@numero IS NOT NULL AND @ryhma IS NOT NULL AND @poistettu =
1)
BEGIN
          INSERT INTO dbo.vapaat_numerot (numero, ryhma)
          VALUES(@numero, @ryhma)
          UPDATE dbo.tyokalu SET numero = null WHERE id = @id
          DELETE FROM dbo.kalibroitavat WHERE toolid = @id
END
```

```
ELSE IF (@numero IS NULL AND @ryhma IS NOT NULL AND @poistettu =
0)
BEGIN
DECLARE @new_numero int
SET @new_numero = (SELECT MIN(numero) FROM dbo.vapaat_numerot
WHERE ryhma = @ryhma)
UPDATE dbo.tyokalu SET numero @new_numero WHERE id = @id
DELETE FROM dbo.vapaat_numerot WHERE ryhma = @ryhma AND numero =
@new_numero
END
```

Esimerkkikoodi 8. Automaattinen proseduuri, joka tarkkailee työkalutaulun update-komentoja.

5 Yhteenveto

Tässä työssä käsiteltiin yksinkertaisen kirjanpitosovelluksen toteuttamista Microsoftin .NET-sovelluskehyksellä. Työssä käytiin läpi .NET-sovelluskehystä, siltä osin miten sitä oltiin työssä käytetty, keskittyen lähinnä ADO.NET-sovelluskehykseen ja ASP.NET-sovelluskehyksen tarjoamaan Web Forms -malliin. .NET-sovelluskehystä kehitetään jatkuvasti ja siihen tehdään uusia kirjastoja sekä päivitetään olemassa olevia kirjastoja. Aiheesta löytyy lukuisia kirjoja, jotka usein keskittyvät yhteen .NET:n osa-alueeseen. Näin ollen tietoa on runsaasti sekä verkossa että painetussa muodossa, joka toi omat haasteensa tiedon rajaukseen työssä. Työkaluvaraston sovelluksen toteuttaminen tapahtui kesällä 2011, jonka aikana opettelin, Java-ohjelmointiin tottuneena, peruspiirteet .NET-sovelluskehyksestä.

Työn tekeminen aloitettiin käymällä keskusteluita työkaluvaraston ylläpitäjän kanssa. Keskustelut koskivat työkaluvaraston toimintaa yleisesti sekä vanhan sovelluksen heikkouksia sekä etuja. Keskusteluiden pohjalta sain hyvän käsityksen työkaluvaraston toiminnasta sekä miten vanha hallintasovellus liittyy työkaluvaraston toimintaan. Tämän käsityksen ansiosta oli helppo lähteä suunnittelemaan uutta sovellusta, jonka oli hyvä vastata käsitemaailmaltaan vanhan sovelluksen käsitteitä. Sovelluksen käsitemaailman pysyessä samana oli työkaluvaraston ylläpitäjän helpompi omaksua uuden sovelluksen käyttö.

CoolTool-sovelluksen avulla työkaluvaraston hallintasovellus on viety nykypäivän tasolle web-pohjaisella käyttöliittymällä, jolloin työkaluvaraston ylläpito ei ole sidottu työpöytäsovellukseen, joka mahdollistaa sovelluksen käyttämisen useilta työpisteiltä. Sovelluk-

sen käyttöliittymän ollessa graafinen, on sen omaksuminen myös helpompaa uusilta ylläpitäjiltä sekä mahdollisilta tuuraajilta. CoolTool-sovelluksessa täysin uusia ominaisuuksia, jotka helpottavat työkaluvaraston ylläpitäjän tehtäviä, kuten useiden työkalujen lisäys kerralla, työntekijän työkalujen siirto toiselle henkilölle ja työntekijöiden mahdollisuus tulostaa omat työkalunsa omalta työasemaltaan, jolloin työntekijöiden asiointi työkaluvarastossa vähentyy huomattavasti, sillä suuri osa työkaluvaraston asioinneista koski työntekijöiden tiedusteluita omista työkaluistaan.

Lähteet

Bochicchio, Daniele; Mostarda, Stefano;De Sanctis, Marco 2011; ASP.NET 4.0 In Practise, Manning, USA.

Davidson, Louis; Kline, Kevin; Klein, Scott; Windisch, Kurt 2008; Pro Sql Server 2008 Relational Database Design and Implementation, Apress, USA.

MacDonald, Matthew; Freeman, Adam; Mario Szpuszta 2010; Pro ASP.NET 4 in C# 2010, Apress, USA.

Microsoft, .NET Framework Conceptual Overview , Verkkodokumentti <http://msdn.microsoft.com/library/zw4w595w.aspx>, Luettu 15.3.2012.

Microsoft, ADO.NET Architecture, Verkkodokumentti <http://msdn.microsoft.com/en-us/library/27y4ybxw(v=vs.100).aspx>, Luettu 3.3.2013.

Microsoft, ADO.NET Overview, Verkkodokumentti < http://msdn.microsoft.com/en-us/library/h43ks021(v=vs.100).aspx>, Luettu 3.3.2013.

Microsoft, Common Language Runtime (CLR), verkkodokumentti <http://msdn.microsoft.com/en-us/library/8bs2ecf4.aspx>, Luettu 15.3.2012.

Microsoft, CREATE TRIGGER (Transact-SQL), Verkkodokumentti, <http://msdn.microsoft.com/en-us/library/ms189799.aspxp://msdn.microsoft.com/en-us/library/ms189799.aspx>, Luettu 18.4.2013.

Microsoft, Pages and Extents, Verkkodokumentti, < http://msdn.microsoft.com/en-US/library/ms190969(v=sql.90).aspx >, Luettu 3.3.2013.

Microsoft, Physical Database Architecture, Verkkodokumentti, <http://msdn.microsoft.com/en-us/library/ms179276(v=sql.90).aspx>, Luettu 3.3.2013.

Norpe, Laatua jo lähes 60 vuotta, verkkodokumentti < http://dasp2.sulatto.fi/norpe/fi/cms.nsf/pages/5CAFCC2AEB9A79F1C2257417003E4926?opendocument>, Luettu 25.2.2013.

Norpe, Norpe tänään, Verkkodokumentti, <http://www.norpe.fi/yritys/norpe-taenaeaen>, Luettu 25.2.2013.

Oppel, Andrew J 2004; Databases Demystified; McGraw-Hill/Osborne, USA.

Simple-Talk, Understanding Garbage Collection in .NET, Verkkodokumentti <http://www.simple-talk.com/dotnet/.net-framework/understanding-garbage-collection-in-.net/>, Kirjoitetu 2009 Luettu 22.3.2012.

Troelsen, Andrew 2010; Pro C# 2010 and the .NET 4 Platform, Apress, USA.

Wikipedia, C Sharp (Programming language), Verkkodokumentti <http://en.wikipedia.org/wiki/C_Sharp_(programming_language)>, Luettu 22.3.2013