Web-palvelun toteuttaminen .NET-alustalla

Raimo Giren

12.6.2008

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

TIIVISTELMÄ

Tietoyhteiskunnan tärkeimpänä päivittäisenä resurssina informaatio yhdistää erilaiset toimijat keskenään. Lisääntyvä tiedon määrä asettaa uusia vaatimuksia sen tallettamiselle, organisoinnille ja saatavuudelle. Kun informaatioalalla toimijoita on paljon, helposti käy niin, että tieto hajautuu ympäri maailmaa ja sen saavuttaminen vaikeutuu. Tiedon hankinta useista eri lähteistä on aina ollut vaikeaa ja saanut ohjelmistosuunnittelijat valmistamaan erilaisia tuotteita joiden avulla ongelmaan päästäisiin käsiksi. Ongelma ei kuitenkaan ole ratkennut, koska jokainen on toiminut omalla tavallaan ja yhteinen formaatti on puuttunut.

Informaation saaminen useasta tietolähteestä yhtä aikaa ja nopeasti tulee olemaan tulevaisuuden avainasia tiedonsiirrossa. Useiden erilaisten palveluidentuottajien tuotteet tulee saada helposti sovelluskehittäjien ja heidän tekemiensä sovellusten kautta asiakkaiden käyttöön. Eräänä ratkaisuna ongelmaan on tarjottu web-palveluita. Ne tarjoavatkin uusia näkemyksiä sellaisten toimijoiden taholta, jotka eivät aiemmin ole kiinnostuneet alustariippumattomasta formaatista, joka mahdollistaa erilaisten toimijoiden yhteistyön.

Tässä tutkielmassa on tarkoitus selvittää kirjallisuuden perusteella mitä tarkoitetaan asiakas-palvelin-arkkitehtuurilla Microsoft .NET-ympäristössä ja kuinka alustariippumaton tietojenhankinta verkon yli toteutetaan web-palvelua käyttäen. Lisäksi selvitetään mitä eri tekniikoita tarvitaan web-palveluiden toteuttamiseen ja käyttämiseen. Esimerkkisovelluksen avulla selvitetään kuinka web-palvelua käyttävä tietokantasovellus luodaan sekä kuinka sitä käytetään. Lisäksi tarkastellaan XML-kielen käyttöä nykyaikaisessa sovellusarkkitehtuurissa sekä web-palveluiden toteutuksessa.

ACM-luokat: (ACM Computing Classification System, 1998 version): D.3.2, H.2.3
Avainsanat: .NET-arkkitehtuuri, web-palvelu, XML, SOAP, WSDL.

SISÄLTÖ

1 JOHDANTO

Tietoverkkojen ja Internetin nopea kehitys on tuonut suuren joukon erilaisia sovelluksia, joiden tehtävänä on käsitellä kasvavaa informaation määrää. Jokainen uusi sovellus on kallis investointi ja sen elinikä on rajallinen. Koska kehityksen vauhti on koko ajan vain lisääntynyt, niin ohjelmistojen kiertoaika puolestaan on koko ajan lyhentynyt ja erilaisia palveluita tarjoavien toimittajien määrä lisääntynyt. Kakki suurimmat ohjelmistoalan yritykset ovat tiukasti pitäneet kiinni omista kehittämistään formaateista tehdä sovelluksia ja tuottaa palveluja. Yrityksistä huolimatta sellaista alustariippumatonta ympäristöä ei ole saatu kehitettyä, jossa eri toimijoiden tuotteet ja tuotettu informaatio kohtaisivat.

Kun ensimmäinen merkkauskieli GML (Generalized Markup Language) esiteltiin vuonna 1969, niin se ei ollut kovin kehittynyt ja siitä tuli pienen erityiskäyttäjäkunnan tuote. Sitä merkitystä minkä merkkauskielen rakenteellinen periaate tietojen käsittelyyn toisi, ei varmasti osattu arvata. Tarvittiin usean vuoden kehitystyö ja otollinen aika, että saatiin aikaan jotakin uutta ja yhteistä, kaikille ohjelmistoalan toimijoille sopiva merkkauskieli XML. Kun samanaikaisesti useat erilaiset ohjelmistoalan toimijat alkoivat olla kiinnostuneita XML-kielestä, niin viimein siitä syntyi uusi rakenteisten dokumenttien käsittelyn standardi. Microsoftin viimein julkaistessa .NET-arkkitehtuurinsa havaittiin, että siinä oli toteutettu hämmästyttävän monessa osassa tiukka XML-integraatio. XML on saavuttanut merkittävän aseman paitsi rakenteisten dokumenttien käsittelyssä, niin myös oliopohjaisten tietorakenteiden koodauksessa. XML on kieli, jonka avulla voidaan toteuttaa täysin alustariippumaton tiedonsiirto ja saada erilaiset sovellukset käyttämään kaikkien toimittajien tuottamia palveluita ja informaatiota (Geetanjali, 2002).

Yhteinen kieli helpottaa palveluiden tekemistä, mutta se ei kuitenkaan takaa niiden saatavuutta. Tarvitaan siis myös keinot saada palvelut jokaisen käytettäväksi. Perinteisten kotitietokoneella toimivien tietokoneohjelmien ongelmana on jatkuvasti ollut sovellukset, jotka toimivat rajallisesti. Koska Internet on tullut jäädäkseen ja sen mahdollistama rajaton informaation julkaiseminen on myös pysyvä ilmiö, niin myös informaation monimuotoisuus vain lisääntyy. Perinteisten menetelmien käyttäminen erityisesti verkkojen yli tapahtuvassa tiedonsiirrossa tulee ongelmalliseksi.

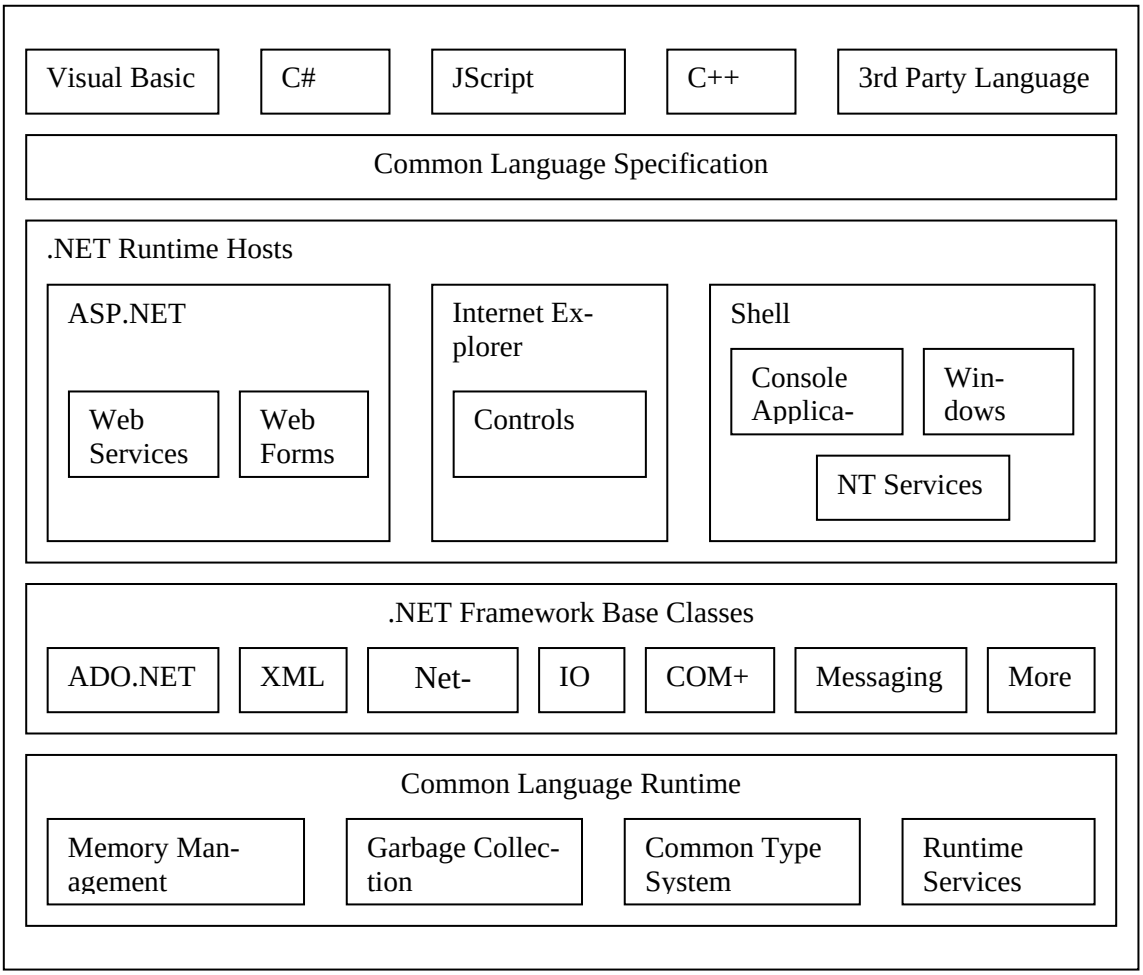
Ratkaisuksi suurten ja eriytyneiden informaatiomäärien käsittelyn ongelmaan on Plattin (2001) mukaan kehitetty viimein aivan uusi tekniikka nimeltään web-palvelutekniikka. Web-palvelutekniikka on kehitetty nimenomaan siksi että sovelluskehittäjät voisivat helposti saavuttaa tarvitsemansa web-palvelut internetistä ja luoda niistä haluamansa kaltaisia ohjelmistoja. Tähän päästään vain kun palveluntarjoajilla on tarjolla palveluja, joko julkisesti tai halutulle ryhmälle, maailman laajuisesti Internetissä, ja jokainen sovelluskehittäjä voi ne tarvitessaan saavuttaa. Palveluiden saavuttamisessa ongelmana on niiden löytäminen ja lisäksi käytössä on oltava jokin sovittu tapa, jolla näitä palveluita voidaan kutsua ja näin yhdistää eri toimijoiden tuotteet. Jos tällainen järjestelmä halutaan, on sen oltava riippumaton käyttöjärjestelmästä sekä ohjelmiston sisäisestä toteutuksesta. Tähän ongelmakenttään web-palvelutekniikka tuo keinot julkaista, etsiä ja käyttää palveluja Internetissä.

Tämän pro gradu -tutkielman tarkoituksena on selvittää millainen on Microsoftin .NET-arkkitehtuuri ja kuinka siihen on liitetty web-palvelutekniikka. Tutkienmassa selvitetään mitä eri tekniikoita web-palvelutekniikan toteutuksessa on käytetty, sekä kuinka asiakas-palvelin-arkkitehtuurin mukainen sovellus rakennetaan Microsoftin .NET Web Services -arkkitehtuuria apuna käyttäen. Tämän lisäksi selvitetään mikä on merkkauskieli ja mikä on sen rooli kun web-palveluja tehdään ja käytetään. Tutkielmaa varten on rakennettu asiakas-palvelin-sovellus käyttäen Visual Studio.NET -sovelluskehitysympäristöä, ja valmiin sovelluksen alustana toimii .NET-arkkitehtuurin Framework.

Tutkielman luvussa 2 esitellään Microsoftin .NET-arkkitehtuuri, jonka tärkeitä osia ovat: CLR (Common Language Runtime), joka on ajonaikainen ympäristö sovelluksille, ja nimiavaruudet sekä luokkakirjastot, jotka tarjoavat palvelut sovellusten tekemiseen. Luvussa 3 esitellään Microsoftin sovelluskehitysympäristö, joka on Visual Studio.NET. Luku 4 käsittelee rakenteellisten dokumenttien merkkauskieltä ja siinä perehdytetään erityisesti XML-kielen rakenteeseen ja sen käyttämiseen. Luvussa 5 on esitelty web-palvelutekniikkaa. Luku 6 käsittelee web-palvelun julkaisemista ja valmiiden palveluiden hankintaa. Luvussa 7 on esitelty web-palvelua hyödyntävä esimerkkisovellus. Luku 8 sisältää yhteenvedon käsitellyistä asioista.

2 .NET-ARKKITEHTUURI

.NET-arkkitehtuuri on Microsoftin uusin ohjelmistoteknologia, jota alettiin kehittää, koska 1990-luvun lopulla tapahtui muutoksia sovelluskehityksen paradigmoissa. Microsoft lähti mukaan kehittämään tuotteitaan kohti olio-ohjelmointia. Useiden eri tekniikoiden ominaisuuksia tutkimalla päädyttiin kehittämään kokonaan uutta arkkitehtuuria. Työnimeksi uudelle sovellusarkkitehtuurille annettiin (NGWS) Next Generation Web Services ja valmistuttuaan sen nimeksi tuli .NET-arkkitehtuuri (Franklin, 2002). .NET-arkkitehtuurin rakenne on esitetty kuvassa 1. Uusina ominaisuuksina se tarjoaa välikielisen käännöksen ja automaattisen roskienkeruun. Molemmat ominaisuudet on jo aiemmin monessa muussa ohjelmointiympäristössä toteutettu. Microsoftin oman kehitystyön tulosta on .NET-arkkitehtuurin tuki useille eri ohjelmointikielille.



Kuva 1: .NET-arkkitehtuurin rakenne (Franklin, 2002).

2.1 CLR-ajoympäristö

.NET-arkkitehtuurissa CLR (Common Language Runtime) on hallittu ohjelmistojen ajoympäristö, joka valvoo, että koodi toteuttaa vain sille sallittuja tehtäviä. CLR:n tehtävät on esitetty kuvassa 1, ja niihin kuuluvat: yleisen tyyppijärjestelmän valvonta, roskien keruu, ajonaikainen käännös, versiointi ja tietoturvasta huolehtiminen. Aikaisemmista versioista poiketen se valvoo myös sitä, kuinka koodi on vuorovaikutuksessa käyttöjärjestelmän kanssa (Platt, 2001). Ajoympäristön kautta käytettävä koodi on nimeltään hallittua koodia, ja CLR:n ulkopuolista koodia kutsutaan hallitsemattomaksi koodiksi. VB.NET-kääntäjä tuottaa vain hallittua koodia, mutta C#-kääntäjässä käyttäjä voi itse valita haluaako tuottaa hallittua vaiko hallitsematonta koodia (Franklin, 2002).

Visual Studio.NET kääntää sovelluksen ensin standardoiduksi MSIL (Microsoft Intermediate Language) -välikoodiksi, joka on samanlainen riippumatta siitä, millä kielellä lähdekoodi on kirjoitettu. Välikoodia ei voida suoraan käyttää missään koneessa, vaan se joudutaan vielä käsittelemään ajonaikaisella (Just In Time) -kääntäjällä. Ajonaikainen käännös alkaa silloin, kun sovellus käynnistetään, ja käännöksen alussa CLR ja ajonaikainen kääntäjä tutkivat koodin tyyppiturvallisuuden. Jos koodi läpäisee tarkastuksen, se käännetään kyseiselle alustalle sopivaksi konekieleksi CLR-moottorin suoritettavaksi (Platt, 2001). Kääntäjä pakottaa ohjelmoijan tuottamaan tiukasti tyypitettyä koodia, jonka avulla varmistetaan, että viittaukset tyyppeihin ovat yhteensopivia, oliokutsut vastaavat olioiden toimintoja ja tunnisteet ovat oikeita. Kun koodista poistettaan selvästi epäsopivat osat jo ennen käännöstä, kuten väärät tyyppimuunnokset, saavutetaan vakaampi ja toimivampi käännös.

.NET-arkkitehtuurissa on myös vauhdilla yleistyvien mobiililaitteiden, kuten kannettavien tietokoneiden ja puhelimien, erityisvaatimukset otettu huomioon (Thomsen, 2001). .NET-arkkitehtuuriin kuuluu kaksi erilaista ajonaikaista kääntäjää. Ajonaikaisista kääntäjistä toinen on tavallinen kääntäjä ja toista kutsutaan kevennetyksi kääntäjäksi. Tavallinen kääntäjä toimii perinteisten kääntäjien tavoin ja kääntää välikielen natiivikoodiksi, kun sovellus ajetaan. Kevennetty kääntäjä on suunniteltu sellaisia pieniä laitteita varten, joiden resurssit ovat rajalliset (Franklin, 2002).

Kevennetyn kääntäjän toiminta perustuu käännetyn koodin määrän rajoittamiseen. Vanhinta käyttämätöntä käännettyä koodia poistettaessa saadaan laitteiston resurssit, kuten tarvittavan muistin määrä, rajattua. Poistettu koodi joudutaan uudelleen kääntämään tarvittaessa, ja se syö resursseja. Haitta on ilmeisesti kuitenkin arvioitu hyötyä pienemmäksi. Tavallisen ja kevennetyn kääntäjän lisäksi on saatavilla myös esikääntäjä. Sen avulla sovelluskokonaisuus voidaan kääntää jo ennen asennusta. Tämä antaa lisää nopeutta koodia ensimmäisen kerran ajettaessa. Välikoodi kuitenkin tarvitaan edelleen, koska CLR tekee tarkastukset sen avulla (Franklin, 2002).

2.2 Nimiavaruudet ja luokkakirjastot

Nimiavaruus tarkoittaa loogista ohjelmiston osaa, joka sisältää joukon nimiä, joista jokaisen tulee olla yksilöllinen ja helposti tunnistettavissa (Thomsen, 2001). Nimiavaruuksien avulla muodostetaan hierarkia, joka auttaa hallitsemaan ja käyttämään komponentteja, jolloin sekaannusten ja ristiriitaisuuksien mahdollisuus vähenee. Vaikka nimiavaruuksien käyttämisen perussyy onkin virheiden välttäminen, niin loogisten ryhmien avulla myös helpotetaan ja nopeutetaan halutun kohteen etsimistä. Komponentit nimetään loogisilla nimillä ja ne järjestetään nimen mukaisiin ryhmiin nimiavaruuksissa. COM-mallissa nimeämisessä käytettään vain kahta tasoa, jotka ovat komponentin nimi ja luokan nimi (Franklin, 2002).

.NET-arkkitehtuurissa tasoja on useita ja sisäkkäisistä nimiavaruuksista voi muodostua melko syviä kokonaisuuksia, jotka koodissa käytettäessä vievät tilaa ja vaativat paljon kirjoittamista. Tämä ongelma on ratkaistu siten, että koodiin voidaan liittää nimiavaruuksia juuresta lähtien Visual Basicissa Imports-komennolla ja C#:ssa avainsanalla using. Esimerkiksi Imports System -komento liittää System-nimiavaruuden, jolloin sitä ei tarvitse erikseen kirjoittaa jokaiseen System-nimiavaruuteen sisältyvän nimen eteen (Franklin, 2002).

Taulukossa 1 on Franklinin (2002) mukaisesti esitelty .NET-arkkitehtuurin luokkakirjasto, jonka tehtäviä ovat mm. seuraavat:

- Korkeimman tason eli ytimen muodostaa System-nimiavaruus, joka on juuriluokka ja sen tehtävänä on tarjota yleisimmin käytetyt tyypit kuten Object.
- System.Data-nimiavaruus sisältää luokat, joiden avulla käsitellään tietokantojen tietoa. Nimiavaruus sisältää luokat tietokantayhteyksien muodostamista ja hallintaa varten.
- Tietojen käsittelyyn kuuluu myös System.XML-nimiavaruus, joka tarjoaa palvelut XML-muotoisen datan käsittelylle, ja on nykyisin tärkeä osa myös muiden palveluiden toimintoja.
- Arkkitehtuurin palveluja tarjoaa System.Diagnostics-nimiavaruus, jossa sijaitsevat luokat koodin seuraamiseen sekä debuggaukseen.
- Perusohjelmointiin kuuluva System.Collections-nimiavaruus tarjoaa luokat joiden avulla toteutetaan tietorakenteet, kuten taulukot, jonot ja listat.
- Perusohjelmointiin liittyvät toiminnot on sijoitettu System.IO-nimiavaruuteen, joka tarjoaa tyyppejä ja luokkia tietojen kirjoittamiseen ja lukemiseen.
- System.Threading-nimiavaruudessa on luokat joiden avulla säikeiden käyttö ohjelmissa on mahdollista.
- Reflektio sisältää System.Reflection-nimiavaruuden, jonka luokkien avulla voi tutkia metadataa.
- Graafisen käyttöliittymän palvelut tarjoaa System.Windows.Forms-nimiavaruus.
- Tietoturvaan kuuluva System.Security-nimiavaruuden avulla on mahdollista toteuttaa salaukseen ja pääsylupiin liityviä toimintoja.
- Web-palveluiden nimiavaruus on System.Web, joka tarjoaa palvelut selaimen ja palvelimen väliselle kommunikoinnille.
- System.Web-nimiavaruus pitää sisällään nimiavaruuden System.Web.Services, joka tarjoaa luokat web-palveluiden tekemiseen.
- System.Web.UI-nimiavaruus mahdollistaa tavan tehdä Windowsin käyttöliittymätekniikan mukaisia www-pohjaisia sovelluksia.

Taulukko 1: .NET-arkkitehtuurin luokkakirjasto (Franklin, 2002).

Palvelu	**Nimiavaruus**
Ydin	System
Tiedot	System.Data System.XML
Komponenttimalli	System.CodeDom System.ComponentModel System.Core
Konfigurointi	System.Configuration
Arkkitehtuurin palvelut	System.Diagnostics System.DirectoryServices System.ServiceProcesses System.Messaging System.Timers
Globalisointi	System.Globalization System.Resources
Verkot ja internet	System.Net
Perusohjelmointi	System.Collctions System.IO System.Text System.Threading
Reflektio	System.Reflection
Graafinen käyttöliittymä (GUI)	System.Drawing System.Windows Forms
Ajonaikainen infrastruktuuri	System.Runtime.InteropServices
Palvelut	System.Runtime.Remoting System.Runtime.Serialization
Tietoturva	System.Security
Web-palvelut	System.Web System.Web.Services System.Web.UI

Olio-paradigmaan sisältyy ajatus luokkien uudelleenkäytöstä ja niiden jatkokehityksestä. Kun jokin asia on kerran keksitty ja hyväksi todettu, niin sitä kannattaa käyttää useamman kuin yhden kerran ja näin välttää turhaa itseään toistavaa työtä, joka aiheuttaa aina uusia virheitä. Menettelyn katsotaan nostavan tuotavuutta ohjelmistotuotannossa, sekä auttavan vakaampien ja monipuolisempien sovellusten kehittämistä (Koskimies, 1998). .NET-arkkitehtuuri tarjoaa sovelluskehittäjien käyttöön luokkakirjastot, joita on mahdollista laajentaa. Samalla tavalla koko käyttöjärjestelmän toiminnallisuus

on organisoitu käyttäen nimiavaruuksia. Kaikki .NET CLR-objektit ja funktiot kuuluvat System-nimiavaruuteen, joka on hyvin suuri ja siksi se on toteutettu useassa erillisessä DLL:ssä (Platt, 2001). Esimerkiksi System.Windows.Forms-nimiavaruuteen kuuluva MessageBox-objekti, löytyy vastaavan nimisestä DLL:stä. Ohjelman tekijän tulee huolehtia, että tämä DLL tulee mukaan Component Library Project-tyyppiseen projektiin.

3 VISUAL STUDIO.NET

Visual Studio.NET on integroitu ohjelmankehitysympäristö (Integrated Development Environment), jonka periaatteet oli toteutettu jo Visual Studio 6 -versiossa, jonka mallina oli ollut VB 5:n kehitysympäristö (Microsoft Press, 1999). VB 6 ei ollut täysin integroidun ohjelmankehitysympäristön mallin mukainen, vaan sen suunnittelussa poiketttiin hieman mallista. Koska integroidun ohjelmankehitysympäristön pariaatteet oli kuitenkin todettu hyvin toimivaksi malliksi, niin se otettiin myös käyttöön kun Visual Studio.NET oli suunnitteilla (Franklin, 2002).

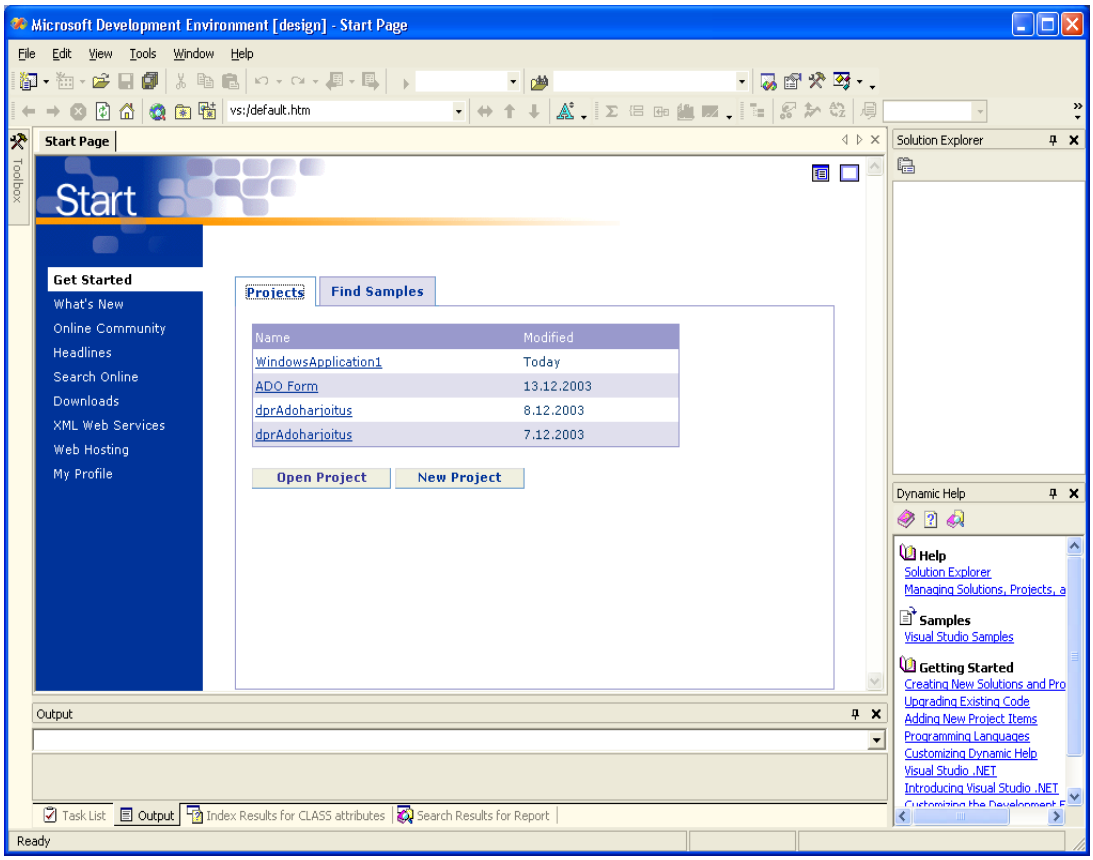
Visual Studio.NET on kehitysympäristö, joka on tarkoitettu usean eri kielen käsittelyyn, ja se mahdollistaa nopean siirtymisen kielestä toiseen, jolloin ohjelmointikielten erilaisten ominaisuuksien hyväksikäyttö sovelluksia kehitettäessä tehostuu. Yhden kehitysympäristön käyttämisestä seuraa myös, että ohjelmointikielen vaihtaminen ei aiheuta ympäristön muuttumista, jolloin uusien asioiden opettelu vähenee ja työskentely tehostuu. Kehitysympäristöjen yhdistämisen tarkoituksena on myös ollut, että ohjelmistojen kehityksessä käytetään aina sitä kieltä, joka parhaiten tukee juuri kehitettävää ohjelmiston osaa (Platt, 2001).

3.1 IDE (Integrated Development Environment)

.NET-arkkitehtuuri ja Visual Studio.NET on luotu tukemaan web-teknologiaa (Franklin, 2002). Nimen loppuosasta .NET tulee myös helposti mielikuva, joka yhdistää sen internetiin ja nettisivujen tekovälineeksi. Internet onkin ollut näyttävästi mukana jo .NET-

arkkitehtuurin suunnitteluvaiheessa ja se näkyy lopputuloksessa. ADO.NET käyttää yhteydetöntä menetelmää internetin yli tietokantaan ja osansa internetin hyväksikäytöstä .NET-kehitysympäristössä hoitaa ASP.NET, joka on aktiivisten palvelinta käyttävien web-sovellusten (Active Server Pages) kehitysympäristö.

Edelleenkin Visual Studio.NET sisältää tuen myös perinteisten Windows-sovellusten tekoon, samoin kuin aikaisemmatkin Visual Studion versiot. Web-sovelluksien tekeminen on tullut lähemmäksi perinteisiä Windows-sovelluksia, kun niiden toteutuksessa on samat menetelmät otettu käyttöön, josta esimerkkinä mainittakoon ASP.NET:iin sisältyvä Web Forms (Franklin, 2002).

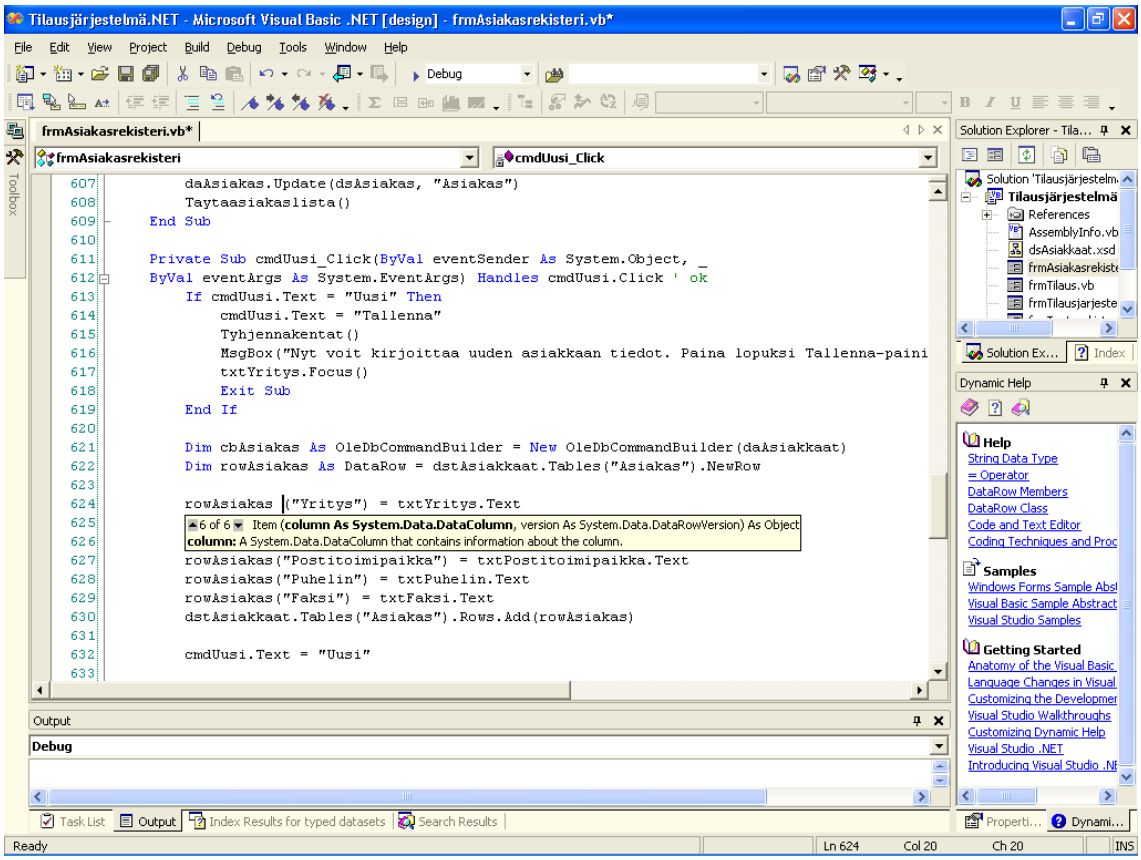


Kuva 2: Visual Studion aloitussivu.

Kuvassa 2 näkyy Visual Studio.NET avattuna. Uudenlainen ulkoasu näkyy web-muotoisena aloitussivuna, josta voi luoda erikielisiä uusia projekteja tai avata vanhoja.

Tässä vaiheessa määritellään uudelle projektille ohjelmointikieli ja projektityyppi. Ohjelmointiympäristö mukautuu kulloisenkin projektityypin ja kielen mukaan. Uusi Dynamic Help -toiminto näkyy vasemmassa alakulmassa. Sen avulla voidaan ratkaista erilaisia ohjelmointiin tai ohjelmointiympäristön käyttöön liittyviä ongelmia. Dynamic Help on aktiivinen avustaja, joka seuraa ohjelmoijan tekemisiä ja hakee tarvittavia tietoja valmiiksi käyttöä varten. Se on myös internetin yli toimiva Help-toiminto, joka hakee tarvittaessa tiedot vaikka Microsoftin palvelusivuilta (Microsoft, 2001).

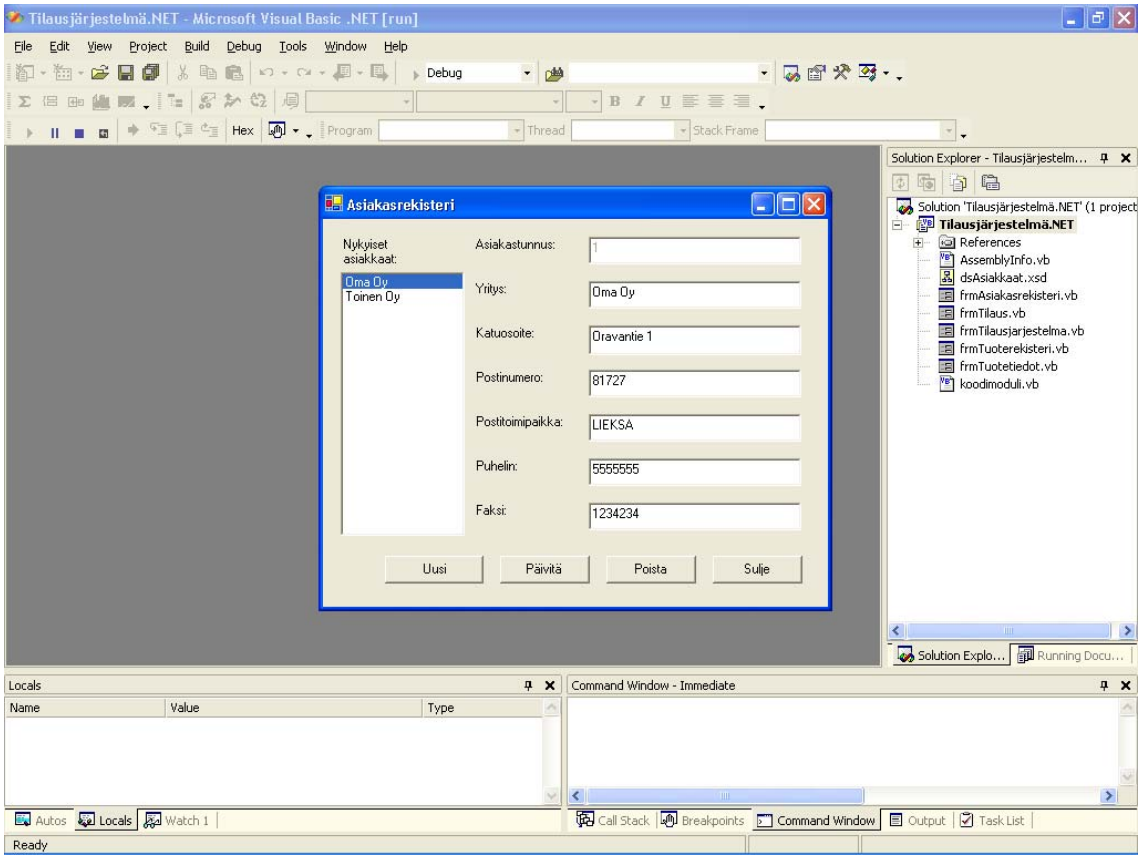
Visual Studio.NET on kasvanut sekä kokonsa että ominaisuuksiensa puolesta melko suureksi ja tehokkaaksi työvälineeksi, joka asettaa myös laitteistolle vaatimuksia. Jos resurssit eivät riitä esimerkiksi Dynamic Help -toiminnon käyttämiseen, kannattaa se poistaa käytöstä, koska se tarvitsee melko paljon resursseja toimiakseen tehokkaasti.



Kuva 3: Visual Studion koodinäkymä.

Uusi Visual Studio.NET sisältää myös useita pieniä yksityiskohtia, jotka auttavat sovelluskehittäjiä työssään. Kuvassa 3 näkyy, että käyttäjä saa rivinumerot halutessaan näkyviin, kuten vanhassa Basic-kielessä oli. Niiden tarkoitus on virheenetsinnän helpottaminen. Numerot ovat vain ohjelmointiympäristön ominaisuus, joka ei tule käännöskoodiin. Koodin kirjoittamista on helpotettu erilaisilla rakenteen näyttävillä ikkunoilla, joista uutta ovat avustavat ikkunat, jotka näyttävät koodin rakenteen vaiheittain.

Visual Studion ajonaikainen tila on saanut lisää ominaisuuksia, ja oletusasetukset ovat nyt selkeämmät kuin ennen. Virheiden etsinnässä käytettävät ikkunat tulevat nyt kaikki näkyviin päällekkäisinä, jolloin niitä on helppo käyttää. Kuvassa 4 näkyy Command Window, jonka avulla voi ajon aikana esimerkiksi muuttaa parametrien arvoja.



Kuva 4: Visual Studion ajonaikainen näkymä.

4 XML

Kaikki tietojenkäsittely perustuu jonkinlaiseen sopimukseen siitä, kuinka tietoa esitetään ja merkataan. Ensimmäinen merkkauskieli oli GML (Generalized Markup Language), joka esiteltiin jo vuonna 1969. Koska minkäänlaista standardia merkkauskielelle ei vielä ollut, sen käyttäminen oli hankalaa. GML vaati oman erikoistuneen kääntäjänsä, ja niitä oli käytössä useita eri versioita. Kun jokainen dokumentti oli tehty hieman eri periaatteita noudattaen, niiden yhteensopivuudesta ei ollut varmuutta (Geetanjali, 2002).

SGML (Structured Generalized Markup Language), joka esiteltiin 1986, oli GML:n jatkokehityksen tulos. Se oli ensimmäinen kansainvälinen standardi dokumenttien käsittelemiseen merkkauskieltä käyttäen. SGML oli ensimmäinen merkkauskieli, joka vietiin standardoitavaksi, ja jonka ISO (International Organization for Standardization) hyväksyi standardinumerolla ISO 8879 (W3C, 2008).

SGML sallii käyttäjän luoda erilaisia alustasta riippumattomia dokumentteja, joita on helppo siirtää verkon yli siten, että informaatio ja sen muoto säilyy alkuperäisenä. Vaikka SGML on hyvin määritelty ja toimiva merkkauskieli, on SGML-dokumenttien rakenne monimutkainen, ja niiden tekemiseen sekä hallintaan tarkoitetut ohjelmistot ovat suurelle yleisölle liian monimutkaisia ja raskaita käyttää (Geetanjali, 2002).

SGML ei ole koskaan saavuttanut suurta suosiota sovelluskehittäjien piirissä, vaan se on jäänyt joidenkin suurten toimijoiden käyttöön. Koska yleinen tarve oli selvästi kevyemmälle merkkauskielelle, niin eri toimijoiden yhteistyön tuloksena viimein syntyi XML (Extensible Markup Language). Microsoft on ollut XML:n alulle paneva voima ja kehityksessä mukana alusta asti. XML ei kuitenkaan myöskään ole aivan uusi merkkauskieli. Se on evoluution ja pitkällisen kehitystyön tuloksena syntynyt SGML:n kevennytty versio, jossa on säilynyt SGML:n menetelmät rakenteisten dokumenttien kirjoittamiseen ja merkkaamiseen. XML on niin kutsuttu metakieli, jonka avulla kuvataan tietoa tiedosta. Se toimii ja sitä käytetään niin, että se sisältää itsessään sekä varsinaisen tiedon että myös tietoa tiedosta, kuten esimerkiksi tiedon nimen, ominaisuuksia ja tietotyypin (Geetanjali, 2002).

XML:n sisältämä tieto koostuu tekstimuotoisesta informaatiosta, joka on aina järjestetty jonkin sovitun periaatteen mukaan. Tiedot voidaan säilöä joko säännöllisesti tai epäsäännöllisesti järjestettynä. XML on myös metakieli, jonka avulla voidaan määritellä merkkauskieliä ja se mahdollistaa myös valitun merkkauskielten laajentamisen ja eri merkkauskielten yhdistämisen. XML on oleellinen osa web-palvelutekniikkaa, koska sitä käytetään apuna useissa sen eri elementeissä, mm. SOAP ja WSDL perustuvat XML-kielelle (Geetanjali, 2002).

Käyttääkseen XML dokumenttia sovellus tarvitsee jäsentäjän tulkitsemaan dokumentin sisältöä. Jäsentäjä koostuu joukosta API-kutsuja, jotka sallivat ohjelmoijalle mahdollisuuden valita, mitä dokumentin elementtejä hän käyttää ja miltä ne näyttävät. Microsoftin .NET-arkkitehtuuri tarjoaa ohjelmistosuunnittelijoille joukon luokkia, eli luokkakirjaston. Luokkakirjastojen avulla voidaan rakentaa ja käyttää XML-dokumentteja erilaisissa sovelluksissa (Hochgurtel, 2003).

Microsoft on käyttänyt XML:ää useassa eri tarkoituksessa, kun .NET-arkkitehtuuria on luotu, ja samalla se integroitu tiukasti uuteen arkkitehtuuriin. Esimerkiksi ADO.NET ja web-palvelut käyttävät hyväkseen XML-integraatiota. Taulukossa 1 näkyvä System.XML on se nimiavaruus, josta .NET-arkkitehtuurissa löytyvät palvelut, joiden avulla tietoa voi tallettaa ja käsitellä XML-muodossa (Jorgensen, 2002).

XML-kieltä on käytetty mm. .NET-arkkitehtuurin seuraavissa osissa:

- olioiden välisessä tiedon siirrossa
- olioiden muuntamisessa sarjamuotoon
- sovellusten hallinnoimisessa
- kansainvälistämisessä
- dokumentoinnissa
- web-palveluissa
- tietokantayhteyksissä.

XML mahdollistaa paitsi alustariippumattoman tavan tiedonsiirtoon, niin myös muita uusia tapoja käsitellä tietoa. Microsoftin .NET-arkkitehtuurissa XML:ää on käytetty siten, että web-palveluissa olioita voidaan palauttaa XML-muodossa. Kun .NET-arkkitehtuurin asiakassovellus tekee pyynnön palvelimelle ja saa XML-muotoisen palautteen, niin se voi joko käyttää sitä sellaisenaan tai rakentaa siitä olion uudelleen. Microsoftin web-palvelutekniikka on rakennettu siten, että se on perimmältään luokkakirjasto, jota käytetään XML-kielen välityksellä (Franklin, 2002).

4.1 XML-kielen rakenne

XML-kielen suunnittelijoilla oli tavoitteena, paitsi alustariippumattomuus, niin myös yksinkertaisuus ja selkeys. Siksi XML-kielen idea on melko yksinkertainen. Suunnittelijoiden tarkoitus on ollut organisoida tiedot niin, että ne muodostavat loogisen kokonaisuuden, jolloin myös ihmisen on helppo niitä lukea. Tavoitteena on myös ollut, että jokainen XML-dokumentti olisi itseään tulkitseva ja sillä voisi myös määritellä rakenteita. XML-kieli koostuu siksi erilaisista osista, kuten elementeistä ja attribuuteista, jotka erottuvat toisistaan ja joilla on oma käyttötarkoituksensa (Jorgensen, 2002).

XML-dokumentissa jokainen osa on ympäröity omilla merkeillään, joita kutsutaan tageiksi. Merkintätapa on samankaltainen kuin HTML-kielessä on ollut käytössä. Aloitustageina käytetään nuolimerkkejä <> ja lopetustagi on samanlainen, mutta sisältää vinoviivan ja on muotoa </> (Hochgurtel, 2003).

Ensimmäinen ja jokaisen XML-dokumentin aloittava osa on PI (Processing Instruction). PI kertoo jäsentäjälle, että kyseessä on nimenomaan XML-dokumentti, ja samalla se kertoo, mikä on kyseisen dokumentin XML-kielen versio. XML-dokumentti alkaa kuvassa 5 esitellyllä tavalla. Kyseisen dokumentin XML-kielen versionumero on 1.0, koska versioita ei ole aiemmin ollut muita, ja samalla rivillä on esitetty merkkien koodaustyyppi, joka on tässä tapauksessa utf-8. Koodaustyyppi ei ole pakollinen, vaan valinnainen osa XML-dokumenttia (Jorgensen, 2002).

```
<?xml version="1.0" encoding="utf-8" ?>
```

Kuva 5: XML-dokumentin versiotiedot.

Jokaisessa hyvin suunnitellussa merkkauskielessä, samoin kuin ohjelmointikielessä, on oltava mahdollisuus kommentoida koodia. Kommentointi merkitään XML-kielessä huutomerkin ja kahden lyhyen viivan sisään samalla tavalla kuin HTML-kielessä. Kuvassa 6 on esitetty esimerkki kommentin käytöstä (Jorgensen, 2002).

```
<!--Muista tarkistaa tämä osa koodista -->
```

Kuva 6: XML-dokumentin kommentointi.

Dokumentin tulee olla helposti tunnistettavissa ja käytettävissä, jolloin se tarvitsee esittelyn. XML-dokumentin alussa on juurielementiksi kutsuttu elementti. Juurielementissä koko dokumentti esitellään antamalla sille sitä kuvaava nimi, esimerkiksi TUOTE, kuten kuvan 7 esimerkissä on tehty (Jorgensen, 2002).

```
<<?xml version="1.0" encoding="utf-8" ?>
<TUOTE>
</TUOTE>
```

Kuva 7: XML-dokumentin juurielementti.

XML-kielen joustava rakenne antaa mahdollisuuden käyttää myös niin sanottuja tyhjiä elementtejä. Tätä ominaisuutta käytetään, kun kyseessä on pienen tietomäärän esittely. Tyhjä elementti ei sisällä erillistä sulkevaa tagia, jolloin koko rakenne saadaan tiiviim-

pään muotoon. Edellinen esimerkki rakennetaan tyhjää elementtiä käyttäen kuvassa 8 esitetyllä tavalla (Hochgurtel, 2003).

```
<?xml version="1.0" encoding="utf-8" ?>
<TUOTE TUOTTEENNIMI =”Laser tulostin”/>
```

Kuva 8: XML-dokumentin tyhjä elementti.

XML:ssä tietoa täydentävä elementti on niin sanottu attribuutti. Se on toinen esimerkki XML-standardin joustavuudesta, mikä antaa ohjelmistosuunnittelijoille erilaisia tapoja muotoilla XML-dokumentteja. Attribuutit voivat olla joko osa avaavaa elementtiä tai toinen tapa on sijoittaa se minkä tahansa elementin sisälle. Attribuutit voivat joko määritellä nimiavaruuksia tai sijaintia. Tällä tavalla attribuutteja käytetään silloin, kun tehdään web-palveluja ja käytössä on SOAP (Simple Object Access Protocol). Kuvassa 9 on esimerkkikoodi siitä kuinka attribuuttia käytetään avaavassa XML-elementissä (Hochgurtel, 2003).

```
<?xml version="1.0" encoding="utf-8" ?>
<TUOTE TUOTTEENNIMI =”Laser tulostin”>
</TUOTE>
```

Kuva 9: XML-dokumentin attribuutti.

Pelkkä aloittava attribuutti ei voi itsessään sisältää paljonkaan tietoa. Niin kutsuttu attribuuttikeskeinen tieto talletetaan useisiin erillisiin ja peräkkäisiin attribuutteihin, jotka ovat elementin sisällä. Näin voidaan tallettaa paljon tietoja ja samalla jokainen tiedon osa voidaan yksilöidä erikseen, jolloin tiedot myös tarkentuvat. Kuvassa 10 on esimerkki attribuuttikeskeisestä tiedon käsittelystä XML:ssä (Hochgurtel, 2003).

```
<?xml version="1.0" encoding="utf-8" ?>
< TUOTE TUOTTEENNIMI =”Laser tulostin”
      TUOTENUMERO=”501”
      AHINTA=”399”
      KPL=”100”/>
```

Kuva 10: Attribuuttikeskeinen XML-dokumentti.

Attribuuttikeskeinen malli antaa kyllä mahdollisuuden tallettaa paljon erillistä tietoa, mutta ei mahdollisuutta yksilöidä niitä tarkasti. Kun jokainen tiedon osa talletetaan omaan elementtiinsä, niin tämä ongelma poistuu (Hochgurtel, 2003). Kuvassa 11 on esimerkki, kuinka elementtikeskeinen formaatti toimii. Jokainen tuote sisällytetään omaan elementtiinsä ja jokainen uusi elementti on juurielementin lapsielementti.

```
<?xml version="1.0" encoding="utf-8" ?>
<TUOTE
<TUOTTEENNIMI> Laser tulostin </TUOTTEENNIMI>
      <TUOTENUMERO>501</TUOTENUMERO>
      <AHINTA>399</AHINTA>
      <KPL>100</ KPL>
</ TUOTE>
```

Kuva 11: Elementtikeskeinen XML-dokumentti.

XML-standardi ei vaadi käyttämään joko attribuutti- tai elementtikeskeistä formaattia. Elementtejä ja attribuutteja käytetään yleensä yhtä aikaa samassa dokumentissa, koska se antaa parhaan informaatioarvon käyttäjälle. Kuvassa 12 on esimerkki siitä, kuinka attribuutit joustavasti sisällytetään elementteihin täsmentämään elementtien tietoja (Hochgurtel, 2003).

```
<?xml version="1.0" encoding="utf-8" ?>
<TUOTE TUOTEENNIMI =”Laser tulostin”>
    <TUOTENUMERO>501</TUOTENUMERO>
    <AHINTA>399</AHINTA>
    <KPL>100</KPL>
</TUOTE>
```

Kuva 12: Attribuutit ja elementit XML-dokumentissa.

Edellisissä esimerkeissä on kaikissa ollut kyseessä vain yksi tuote, josta on muodostettu elementti, ja jolla on ollut joko attribuutteja tai lapsielementtejä. Käytännössä tilanne on kuitenkin lähes aina sellainen, että useita samanlaisia kohteita, jotka sisältävät omat tietonsa, joudutaan säilömään monta yhtä aikaa. Silloin tarvitaan enemmän sisäkkäisiä elementtejä. Sisäkkäiset elementit lisäävät tiedon määrää ja parantavat tulkitsemista (Hochgurtel, 2003).

Useita sisäkkäisiä elementtejä uusille tuotteille saadaan aikaan niin, että edellisten esimerkkien XML-dokumenttiin lisätään uusi eriniminen juurielementti. Uuden juurielementin sisään talletetaan jokainen kohde, eli tässä tapauksessa tuote, omaksi elementiksi, Tuote on silloin juurielementin lapsielementti, joka sisältää omat lapsielementtinsä. XML:n sallima sisäisten elementtien rakennemalli voidaan kirjoittaa esimerkiksi kuvan 13 esittämällä tavalla. Koko rakenne on nyt paketoitu kokonaisuutta kuvaavalla nimellä, joka on Tuoterekisteri. Samalla tavalla kuin oikeassa tuoterekisterissäkin, kaikki tuotteet on sijoitettu rekisterin sisälle. Jokaisella tuotteella on omat ominaisuutensa, jotka se tarvitsee. Jokaisella tuotteella on yksilöllinen ja helposti tunnistettavissa oleva nimi. Koska nimikään ei vielä takaa yksilöitävyyttä, niin jokaisella tuotteella on sen lisäksi vielä kuvassa 13 myös tuotenumero.

```
<?xml version="1.0" encoding="utf-8" ?>
<TUOTEREKISTERI>
    <TUOTE TUOTTEENNIMI =”Laser tulostin”>
        <TUOTENUMERO>501</TUOTENUMERO>
        <AHINTA>399</AHINTA>
        <KPL>100</ KPL>
    </TUOTE>
    < TUOTE TUOTTEENNIMI =” Mustavalko tulostin”>
        <TUOTENUMERO >502</TUOTENUMERO>
        <AHINTA >299</AHINTA>
        <KPL >200</KPL>
    </TUOTE>
</TUOTEREKISTERI>
```

Kuva 13: XML-dokumentin sisäkkäinen rakenne.

4.2 XML-kielen nimiavaruudet

XML:ssä käytetään nimiavaruuksia melko samalla tavoin kuin Microsoftin .NET-ympäristössä ja muissa olioperustaisissa ohjelmointiympäristöissä. Nimiavaruus tarkoittaa, että jokainen XML-dokumentissa käytetyt elementti on nimetty yksilöllisesti ja se käyttää niitä samoja ominaisuuksia, mitä kyseisessä nimiavaruudessa on käytetty. Nimiavaruuden esittelyt tehdään juurielementissä, kuten kuvasta 14 käy ilmi, ja hyödyntää yksilöinnissä URL:ää, mutta ei ole pakko sisältyä siihen, koska skeeman URL-viittaus voidaan tehdä ilman nimiavaruuden määrittelyä (Hochgurtel, 2003).

```
<TUOTE xmlns:WEBSERVICES=”www.yritysinfo.com/XML”>
</TUOTE>
```

Kuva 14: Nimiavaruuden esittely XML-dokumentissa.

Nimiavaruudet ovat käytössä myös osana web-palvelun rakennetta. Nimiavaruudet otetaan käyttöön xmlns (XML namespace) -määrittelyn avulla. Jos xmlns-määrittelyä ei tehdä ollenkaan, niin käytössä on silloin oletusnimiavaruus (Default Namespace) (Hochgurtel, 2003). Seuraavassa esimerkissä on näytetty, kuinka nimiavaruuden esittely tehdään juurielementissä ja sitten kokonainen elementtirakenne, josta näkyy, että nyt jokainen uusi lapsielementti alkaa myös nimiavaruuden nimellä (kuva 15).

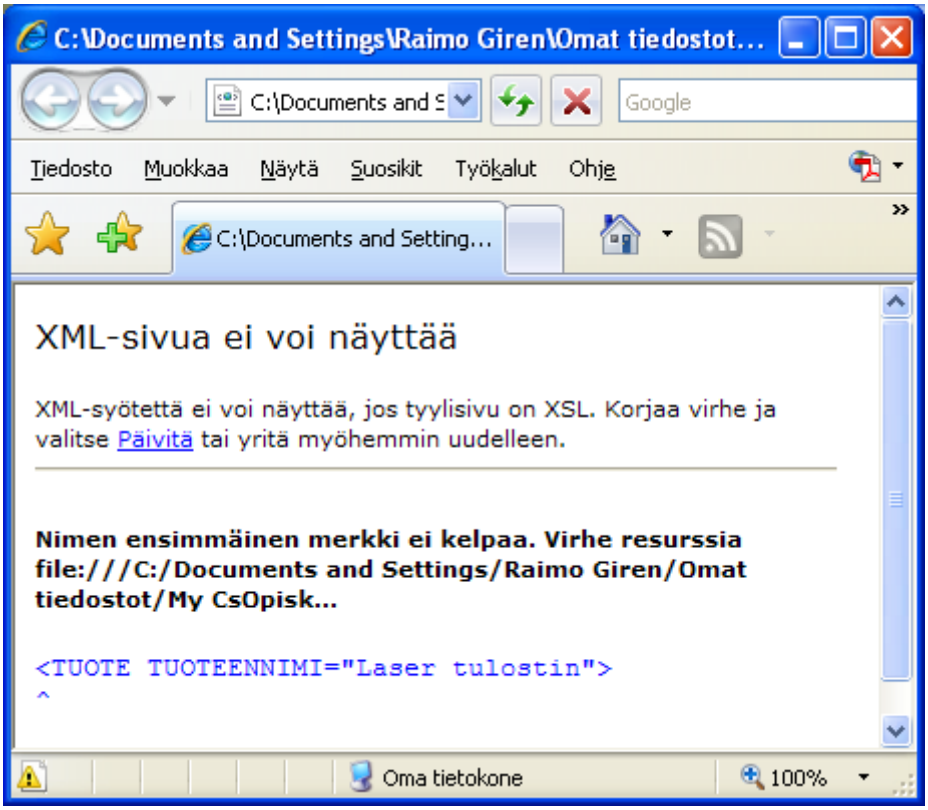
```
<?xml version="1.0" encoding="utf-8" ?>
<TUOTE xmlns:WEBSERVICES=”www.yritysinfo.com/XML”>
    <WEBSERVICES:TUOTEENNIMI>Laser tulostin</
        WEBSERVICES:TUOTEENNIMI>
    <WEBSERVICES:TUOTENUMERO>501</
        WEBSERVICES:TUOTENUMERO>
    <WEBSERVICES:AHINTA>399</
        WEBSERVICES:AHINTA>
    <WEBSERVICES:KPL>100</ WEBSERVICES:KPL>
</TUOTE>
```

Kuva 15: XML-dokumentin nimiavaruus.

4.3 XML-dokumentin muotoilu

Kun XML-dokumenttia katsoo ensimmäistä kertaa, niin kieli muistuttaa hyvin paljon HTML-kieltä. Suurin ero on kuitenkin XML-kielen paljon tiukemmassa määrittelyssä. XML-dokumentin tulee olla hyvin muotoiltu (Well-Formed) ja hyväksyttävä (Valid). Hyvän muotoilun ja hyväksyttävyyden W3C on määritellyt ja kirjannut XML-standardiin. XML-dokumentissa tulee Jorgensin (2002) mukaan olla seuraavat elementit:

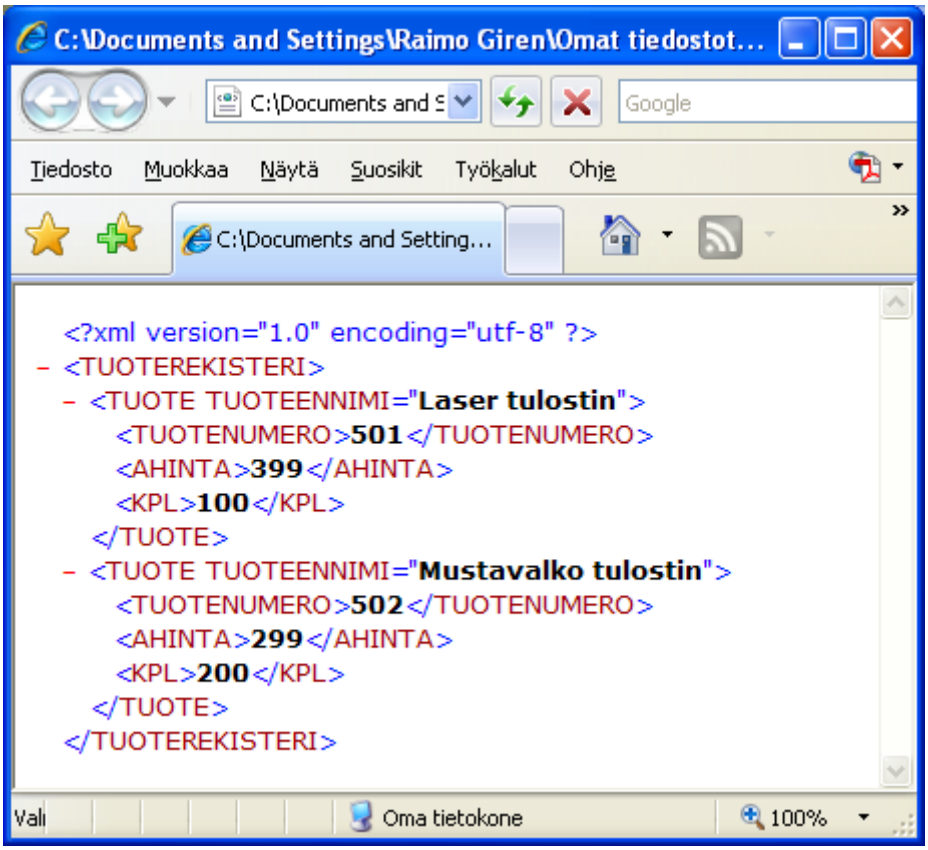
- Dokumentissa tulee olla vain yksi juurielementti.
- Jokainen aloittava tagi tulee loppua lopetustagiin. Tämä sääntö ei kuitenkaan koske tyhjää elementtiä (Hochgurtel, 2003).
- Dokumentin elementeillä tulee olla sisäkkäinen rakenne.
- Attribuutin ensimmäisen merkin tulee olla joko kirjain tai alaviiva.
- Yksilöivän attribuutin nimi saa esiintyä vain kerran samassa tagissa.



Kuva 16: Virheellinen XML-dokumentti Internet-selaimessa.

XML-dokumentin muotoilun voi tarkastaa nykyisin internetpalveluna tai käyttäen apuna erilaisia työkaluohjelmistoja, joilla voi kirjoittaa ja tarkastaa dokumentin rakenteen. Ilman näitä helpoin tapa tarkastaa, onko dokumentti hyvin muotoiltu, on avata se Internet-selaimeen. Uudet selaimet tarkastavat XML-muotoilut ja raportoivat niissä esiintyvistä virheistä (Hochgurtel, 2003).

Kuvassa 16 on yritetty avata virheellistä XML-dokumenttia ja siitä on seurannut virhesanoma, ja kuvassa 17 näkyy kuinka kyseisen XML-dokumentin sisältö näkyy silloin, kun virhe on korjattu (Hochgurtel, 2003).



Kuva 17: Hyvin muotoiltu XML-dokumentti Internet-selaimessa.

4.3.1 DTD

DTD (Document Type Defination) on vanha mutta vielä nykyisin käytössä oleva dokumentin rakennemäärittelytapa. Se määrittelee rakenteisen dokumentin sallitut ilmenemismuodot elementeille ja attribuuteille. DTD ei ole XML-kieltä vaan sillä tehdystä määrittelystä muodostuu uusi merkintäkieli. DTD on peräisin SGML-standardista ja sitä kautta tullut käyttöön myös XML-kielen tyylikieleksi (Hochgurtel, 2003). Kuvassa 18 on esitelty Hochgurtelin (2003) mukaisesti DTD-dokumentti, jonka avulla voidaan määritellä muotoilut kuvan 19 yksinkertaiselle muotoiltavalle XML-dokumentille.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- This is a comment -->
<!-- The following code is the DTD -->
<!-- The PI and the DTD are the prolog of the document
   -->
<!DOCTYPE TUOTE [
<!ELEMENT TUOTE (TUOTENUMERO?, AHINTA +, KPL +)>
<!ATTLIST TUOTE TUOTEENNIMI CDATA #REQUIRED>
<!ELEMENT TUOTENUMERO (#PCDATA)>
<!ELEMENT AHINTA (#PCDATA)>
<!ELEMENT KPL (#PCDATA)>
 ]>
<TUOTE TUOTEENNIMI =”Laser tulostin”>
      <TUOTENUMERO>501</TUOTENUMERO>
      <AHINTA>399</AHINTA>
            <KPL>100</KPL>
 </ TUOTE >
```

Kuva 18: DTD-dokumentti.

```
<?xml version="1.0" encoding="utf-8" ?>
<TUOTE TUOTEENNIMI =”Laser tulostin”>
    <TUOTENUMERO>501</TUOTENUMERO>
    <AHINTA>399</AHINTA>
    <KPL>100</KPL>
</TUOTE>
```

Kuva 19: Muotoiltava XML-dokumentti.

4.3.2 XSD

XML-skeemoilla tarkoitetaan DTD:tä vastaavaa teknologiaa, jonka avulla kuvataan XML-dokumentin rakennetta. Erona kuitenkin on se, että DTD määritellään omalla kielellään ja skeemat määritellään XML:llä. XML Schema -teknologia on W3C:n hyväksymä vuonna 2001, ja siten myös standardoitu tapa, jonka avulla muotoillaan ja kuvataan XML-dokumentteja (Geetanjali, 2002). Skeemoilla on mahdollista kuvata esimerkiksi XML-muotoisen tuoterekisterin sallittu rakenne, jossa määritellään millaista yhteistä mallia ja kieltä siinä käytetään. Sen avulla tuoterekisterissä käytettävät nimet ja koko sanasto voidaan kuvailla niin tarkasti, että myös tietokoneet ymmärtävät sitä.

XML-skeemoja käytetään web-palveluiden yhteydessä, koska niillä on useita hyviä ominaisuuksia, joita DTD ei tue. Näistä jo aiemmin oli esillä määrittelyssä käytetty kieli. Koska XML-skeeman kirjoittamisessa käytetään XML-kieltä, niin se tekee käyttäjälle niiden luomisen ja ymmärtämisen helpommaksi. Skeemoilla voidaan elementin tietosisältöä rajoittaa tiukemmin kuin DTD:llä, koska skeemat tukevat useita tietotyyppejä, kuten kokonaislukuja, totuusarvoja, päivämääriä jne. Näiden lisäksi myös käyttäjän määrittelemät tietotyypit ovat sallittuja. XML-skeemoilla on myös periytyvyys ominaisuutena, jolloin aiempien dokumenttien käytettävyys paranee. Skeemat tukevat nimiavaruuksia ja ne mahdollistavat useiden eri nimiavaruuksia käyttävien elementtien käyttämisen samassa dokumentissa (Geetanjali, 2002).

Koska XSD-dokumentti kirjoitetaan XML-kielellä niin se alkaa esittelyllä, samalla tavalla kuin muutkin XML-dokumetit (kuva 20). Myös elementtien käyttö on samanlaista kuin elementtikeskeisessä XML-dokumentissa. XSD-dokumenttissa juurielementtinä käytetään schema-elementtiä, jonka sisälle sijoitetaan muut elementit, jotka puolestaan sisältävät tiedon siitä, kuinka XML-dokumentti muotoillaan. Schema-elementtissä esitellään nimiavaruus, jota muotoiltavassa dokumentissa käytetään. Nimiavaruus esitellään käyttäen avainsanaa xmlns (Geetanjali, 2002).

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
</xsd:schema>
```

Kuva 20: XSD-dokumenttin syntaksi.

XSD-dokumentin elementti on myös oma tyyppinsä eli elementtityyppi. Elementtityyppi on tarkoitettu määrittelemään, kuinka XML-dokumentti muotoillaan. Sillä on kaksi attribuuttia name ja type. Kuvan 21 esimerkissä name määrittelee nimen elementille ja type määrittelee sille tietotyypin (Geetanjali, 2002).

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
    <xsd:element name=<name> type=<data type>>
    </xsd:element>
</xsd:schema>
```

Kuva 21: XSD-dokumenttin elementtityyppinen elementti.

Web-palvelussa SOAP, XML ja XSD kytkeytyvät toisiinsa tiiviisti. Web-palvelu käyttää XML-kieltä ja XML-dokumentteja määrittelemään palvelun rakennetta ja toimintaa. XML-dokumenttien rakenne ja muotoilu puolestaan määritellään XSD-dokumentissa.

Nykyaikaisille ja kehittyneille ohjelmointikielille on ominaista tiukka tyyppisidonnaisuus, koska se parantaa koodin tarkistettavuutta. Koska XML-kielessä on mahdollista tyyppien määrittely, niin samoin ne voidaan määritellä myös XSD-dokumentissa ja sen sisältämiä skeema-tyyppejä käytetään usein identifioimaan primitiivi-tyyppejä SOAP-viesteissä. Uusien tyyppien määrittelyä voidaan tehdä kahdella erilaisella määrittelyllä, jotka ovat simpleType ja complexType. SimpleType määrittely voi sisältää vain tekstiä, eikä sillä voi määritellä attribuutteja tai elementtejä. Sen sijaan uusia rakenteellisia tyyppejä voidaan määritellä complexTypen avulla (kuva 22), jolla voi määritellä uusia attribuutteja ja elementtejä. Elementtien esiintymiskertoja voidaan myös rajoittaa määrittelemällä minOccurs ja maxOccurs arvot (Hochgurtel, 2003).

Olioperustaisissa ohjelmointikielissä sovellussuunnittelijan työtä on jo pitkään helpottanut mahdollisuus luoda uutta vanhasta periyttämällä. Myös XML-skeema antaa keinot saman tekniikan käyttöön, koska se sallii tyyppien määrittelemisen periyttämisen avulla. Erona olio-ohjelmoinnin periyttämismalliin on laajempi perinnän käsite. XML-skeemoissa periminen voi tapahtua laajentamalla periytyviä ominaisuuksia samoin kuin olio-ohjelmoinnissa. Tämän lisäksi perittäviä ominaisuuksia voidaan myös rajoittaa, eli sulkea pois niitä olemassa olevia ominaisuuksia, joita ei tarvita (Geetanjali, 2002).

Kuten kuvasta 22 käy ilmi, XML-skeema on melko monimutkainen ja vaikeaselkoinen rakenne. Sovelluskehittäjien ei kuitenkaan tarvitse XML-skeemaa aivan manuaalisesti kirjoittaa. XML-dokumentteja varten on luotu useita erilaisia kehitysympäristöjä, jotka tekevät skeeman määrittelevän XSD-dokumentin itsenäisesti, kun niille annetaan XML-dokumentti, josta skeema tehdään. Muun muassa XML Spy ja Visual Studio.NET generoivat XSD-dokumentin automaattisesti. Kuvan 22 XML-skeeman on generoinut Visual Studio.NET. Valmista dokumenttia on ohjelmoijan sitten helpompi muokata tarvitsemallaan tavalla (Hochgurtel, 2003).

```
<?xml version="1.0" ?>
<xs:schema id="NewDataSet"
    targetNamespace=http://www.advocatemedia.com/~vs1C0.xsd
    xmlns:mstns=http://www.advocatemedia.com/~vs1C0.xsd
    xmlns=http://www.advocatemedia.com/~vs1C0.xsd
    xmlns:xs=http://www.w3.org/2001/XMLSchema
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    attributeFormDefault="qualified"
    elementFormDefault="qualified">
<xs:element name="BOOK">
    <xs:complexType>
        <xs:sequence>
        <xs:element name="PAGECOUNT" type="xs:string" minOccurs="0"
            maxOccurs="3" msdata:Ordinal="0" />
        <xs:element name="AUTHOR" type="xs:string" minOccurs="0"
            msdata:Ordinal="1" />
        <xs:element name="PUBLISHER" type="xs:string" minOccurs="0"
            msdata:Ordinal="2" />
        </xs:sequence>
        <xs:attribute name="TITLE" form="unqualified" type="xs:string" />
    </xs:complexType>
</xs:element>
<xs:element name="NewDataSet" msdata:IsDataSet="true"
    msdata:EnforceConstraints="False">
    <xs:complexType>
        <xs:choice maxOccurs="unbounded">
            <xs:element ref="BOOK" />
        </xs:choice>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Kuva 22: XSD-dokumentti ja tyyppimärittelyt (liite 2).

5 WEB-PALVELU

Web-palveluiden kehitystä eteenpäin vievänä voimana on ollut ajatus, että sovelluskehittäjät voisivat helposti saavuttaa tarvitsemansa palvelut internetistä ja luoda niistä haluamansa kaltaisia ohjelmistoja. Jotta tähän päästäisiin on palveluntarjoajilla oltava palveluita maailman laajuisesti Internetissä, joita jokainen sovelluskehittäjä voi saavuttaa. Toisaalta on myös oltava jokin tapa, jolla näitä palveluita voidaan kutsua ja yhdistää eri toimijoiden tuotteet. Jos tällainen järjestelmä halutaan, on sen oltava riippumaton käyttöjärjestelmästä sekä ohjelmiston sisäisestä toteutuksesta (Platt, 2001).

Erilaisten järjestelmien yhteensovittaminen on aikaisemmin ollut erittäin työlästä. Web-palvelu antaa viimein helpon tavan yhdistellä informaation tasolla erilailla toimivia ohjelmistotuotteita keskenään. Web-palvelut tarjoaa palvelinobjektien avulla ratkaisut vastaanottaa asiakassovelluksilta palvelupyyntöjä riippumatta siitä, millä alustalla sovellus toimii ja miten se on toteutettu. Web-palvelussa tiedonsiirto toteutetaan internetin yli siten, että käytössä on HTML (HyperText Markup Language) tai SOAP (Simple Object Access Protocol) sekä XML (Extensible Markup Language), jolla koodataan verkossa liikuteltavat tiedot (Jorgensen, 2002).

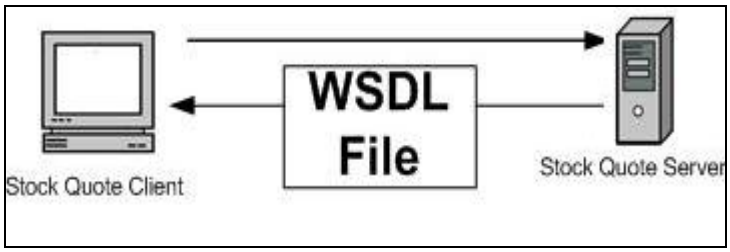
Web-palvelu on joustava ratkaisu tiedonsiirtoon, koska se hyväksyy kolme protokollaa, joiden avulla tiedot käsitellään, kun palvelupyyntöjä käytetään verkon yli. Käytössä olevat protokollat ovat HTTP GET, HTTP POST ja SOAP. Vaikka jokaista näistä voidaankin käyttää, niin SOAP on käyttökelpoisin protokolla silloin, kun web-palveluita ja niitä käyttäviä sovelluksia tehdään Microsoft Visual Studio.NET -ympäristössä. Visual Studio.NET tarjoaa sovelluskehittäjille tuen juuri SOAP protokollalle. Web-palvelu voidaan julkaista asiakkaille UDDI (Universal Discovery Descriptionand Integration) -rekisterin avulla, jossa säilytetään palvelun tietoja. UDDI tarjoaa osoitteen, jonka avulla sovelluskehittäjä voi löytää tarvitsemansa palvelun. Web-palvelusta voi noutaa WSDL (Web Services Description Language) -dokumentin, jossa on kuvaus palvelusta ja sen käyttämiseen tarvittavista metodeista. Näiden avulla voi rakentaa asiakassovelluksen, joka käyttää web-palvelua lähes samalla tavalla kuin tavallista ohjelmistokomponenttia (Platt, 2001).

5.1 WSDL

Sovelluskehittäjät, jotka tekevät ohjelmistoja, jotka käyttävät hyväkseen web-palveluja, tarvitsevat kuvauksen eli tiedot siitä, mitä kukin palvelu niille tarjoaa. Kuvauksen tulee myös neuvoa, mitä asiakassovelluksen tulee tehdä, että se saa palveluntarjoajan tekemään haluamansa toiminnot. Kuvaus web-palvelusta toteutetaan dokumentissa nimeltä WSDL (Platt, 2001).

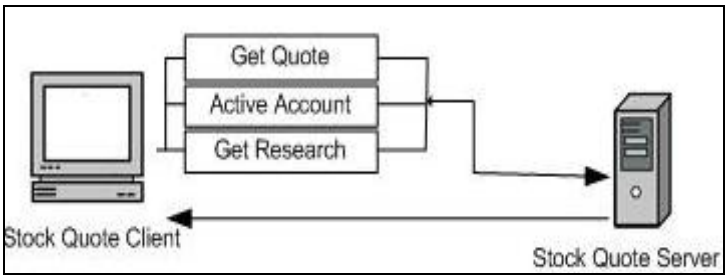
WSDL (Web Services Description Language) on web-palvelujen kuvaamiseen tarkoitettu, XML-pohjainen kuvauskieli (Hochgurtel, 2003). WSDL-kielen avulla kuvataan palvelun toiminnallista tietoa, kuten web-palvelun käyttäjälle tarjoamat rajapinnat ja metodit, joiden avulla palvelua käytetään. Asiakassovelluksen kannalta tärkeää on tietää web-palvelun tarjoamat funktiot ja niiden parametrit sekä niiden tukemat protokollat. Tiedot, jotka WSDL kuvaa, sisältää melko samat asiat kuin COM-komponenttien tyyppikirjasto. Erona kuitenkin on, että COM-komponenttien tyyppikirjasto on käytännössä vain Microsoftin Windows-tuotteita varten, web-palvelu ja WSDL sen sijaan on suunniteltu yleispäteväksi ratkaisuksi ja siten tarkoitettu toimimaan millä tahansa alustalla. WSDL on sovelluskehittäjän kannalta samanlainen käyttää kuin tyyppikirjasto.

Asiakas-palvelin-arkkitehtuureissa käytetään yleisesti periaatetta, jossa palvelua käyttävä asiakassovellus pyytää tarvitsemansa tiedot palvelimelta ja noutaa ne luvan saatuaan. Web-palvelu ei tee poikkeusta tästä käytännöstä, mutta palvelua käyttävä asiakassovellus voi noutaa ensin myös WSDL-tiedoston palvelimelta (kuva 23). Asiakassovelluksen tulee sitten vain pystyä tulkitsemaan saamansa WSDL-tiedoston XML-muotoinen sisältö oikein. WSDL-tiedostosta käytetään joskus myös nimeä sopimus (Contract), koska siinä luetellaan ne asiat, joita web-palvelu tekee ja kuinka palveluja tulee pyytää (Platt, 2001).



Kuva 23: Web-palvelun käyttäminen WSDL-tiedoston avulla (Hochgurtel, 2003).

Kun asiakassovellus on saanut WSDL-tiedoston palvelimelta, se tulkitsee dokumentin. Sillä on mahdollisuus käyttää kolmea palvelimen tarjoamaa metodia, jotka WSDL-dokumentti määrittelee. Metodit ovat: Get Quote, Account and Get Research (kuva 24). Asiakassovelluksen käyttäjällä on nyt rajapinta ja oikeus näiden metodeiden käyttämiseen.



Kuva 24: Web-palvelun metodit (Hochgurtel, 2003).

Web-palvelun WSDL-määrityksen komponentteja ovat:

- message
- portType
- binding
- service
- types.

Message-elementin tehtävänä on kuvailla web-palvelun lähettämä tai vastaanottama sanoma. PortType ryhmittelee web-palvelun sanomat loogisiksi tapahtumiksi eli operaatioiksi, jotka koostuvat useista, toisiinsa liittyvistä sanomista. PortType-komponentti kuvaa miltä web-palvelun rajapinta todellisuudessa näyttää käyttäjälle päin. Binding kuvaa web-palvelun rajapinnan sidonnan käytettävään tietoliikenneprotokollaan ja tiedon esitystapaan. Se määrittää sanomien siirrossa käytettävän protokollan ja tiedon formaatin. Servicen-elementin tehtävänä on määrittää web-palveluun kuuluvat päätepisteet eli portit (port), joita voi olla yksi tai useampia. Portti tarkoittaa palveluportin verkko-osoitetta ja palvelun käyttämää tietoliikenneprotokollaa sekä lisäksi käytettävää tiedonsiirtoformaattia. Types määrittää sanomien käyttämät tietotyypit XML-skeemasyntaksin avulla ilmaistuna (Hochgurtel, 2003).

Kuvassa 25 on esitetty TilausjarjestelmaWS-palvelun WSDL-dokumentin osa (ks. myös liite 2). Dokumentti sisältää paljon pitkiä lauseita ja on erittäin monimutkaisen näköinen. Alussa on jo tutut XML-dokumentin määritykset, koska myös WSDL-dokumentti on XML-dokumentti. Sen jälkeen tulevat WSDL-määritykset, joista ilmenee käytettävät versiot tekniikoista, joita dokumentin hallinnassa käytetään. Näitä tietoja ovat mm. SOAP-tiedot, XML-skeeman tiedot sekä kyseisen dokumentin käytössä oleva nimiavaruus (Platt 2001).

WSDL-dokumentin loppupuolella oleva <service>-elementti määrittelee web-palvelun nimen ja siihen kuuluvat päätepisteet. Elementti pitää sisällään <port>-elementin kutakin sellaista protokollaa kohden, jota tämä web-palvelu voi käsitellä. Mahdollisia vaihtoehtoja olisivat HTTP GET, HTTP POST JA SOAP. TilausjarjestelmaWS-palvelu on rakennettu käyttämään vain SOAP-protokollaa, eikä se hyväksy muilla protokollilla tehtyjä palvelupyyntöjä. Dokumentissa esiintyy vain yksi <port>-määrittely, TilausjarjestelmaWSSoap, josta käy ilmi nimi ja protokolla. Kyseisen <port>-elementtin tehtävänä on pitää sisällään määrittelyjä käytössä oleviin muihin dokumentteihin. Esimerkkisovelluksen WSDL-tiedoston <port>-elementin osoite on: http://pomi1/TilausjarjestelmaWSNET/TilausjarjestelmaWS.asmx. Kyseessä on ASMX-tiedosto, jossa vuorostaan on varsinaisen palvelun toteuttavan koodin sisältävään dokumenttiin viittaavan tiedoston osoite.

```
<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" targetName-
space="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
- <wsdl:types>
- <s:schema elementFormDefault="qualified" targetName-
space="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS">
<s:import namespace="http://www.w3.org/2001/XMLSchema" />
- <s:element name="dstAsiakkaat">
<s:complexType />
</s:element>
- <s:element name="dstAsiakkaatResponse">
- <s:complexType>
- <s:sequence>
- <s:element minOccurs="0" maxOccurs="1" name="dstAsiakkaatResult">
- <s:complexType>
- <s:sequence>
<s:element ref="s:schema" />
<s:any />
</s:sequence>
</s:complexType>
</s:element>
</s:sequence>
</s:complexType>
</s:element>
- <s:element name="dstAsiakkaatTallenna">
- <s:complexType>


- <wsdl:service name="TilausjarjestelmaWS">
<documentation xmlns="http://schemas.xmlsoap.org/wsdl/" />
- <wsdl:port name="TilausjarjestelmaWSSoap" bind-
ing="tns:TilausjarjestelmaWSSoap">
<soap:address loca-
tion="http://pomi1/TilausjarjestelmaWSNET/TilausjarjestelmaWS.asmx" />
</wsdl:port>
</wsdl:service>
```

Kuva 25: Tiedoston TilausjarjestelmaWS.WSDL osa.

5.2 HTTP-GET

Web-palvelussa on mahdollista käyttää yksinkertaista HTTP GET -pyyntöä, ja lähettää tarvittavat parametrit URL-merkkijonon mukana. Edellytyksenä tälle on, että ohjelmoitu web-palvelu tukee tätä protokollaa. Palvelun tukemat protokollat määritellään WSDL-tiedostossa <port>-elementin sisällä (kuva 25). Jokaista tuettua protokollaa kohti on yksi määrittelevä <port>-elementti. HTTP GET -pyyntö toimii siten, että URL-merkkijonon saapuessa palvelimelle siitä erotellaan parametrit. Tämän jälkeen palvelinsovellus luo tarvittavan objektin ja kutsuu sen metodia välittäen sille sen tarvitsemat parametrit. Saatuaan objektilta vastauksena paluuarvon palvelinsovellus muotoilee sen XML-kielelle ja palauttaa asiakassovellukselle (Platt 2001).

5.3 HTTP-POST

Web-palvelu hyväksyy kutsuina käytettäväksi myös HTTP POST -pyyntöjä. Samalla tavalla kuin HTTP GET -pyynnöillä, myös HTTP POST -pyynnöillä tulee olla tuki määriteltynä kyseiselle web-palvelulle. Määrittely tehdään myös samalla tavalla kuin HTTP GET -pyynnöllä. Kun palvelupyyntöjen välityksessä käytetään HTTP POST -metodia, niin esimerkkisovelluksen voi tehdä siten, että luo tavallisen web-lomakkeen, jossa on tarvittavat lomakekomponentit ja toiminnot (Platt 2001). Palvelupyyntö pitää vain paketoida <FORM METHOD>-elementin sisälle (kuva 26).

```
<form METHOD=”POST”
      ACTION=”http://localhost/TilausjatjestelmaService.asmx. /GetTilaus”
      <input TYPE=”TEXT” NAME=”Tuotenumero” VALUE=”Text”>
      <input TYPE=”SUBMIT” VALUE=”Submit Form”>
      <input TYPE=”RESET” VALUE=”Reset Form”>
</form>
```

Kuva 26: GET POST-palvelupyyntö.

5.4 SOAP

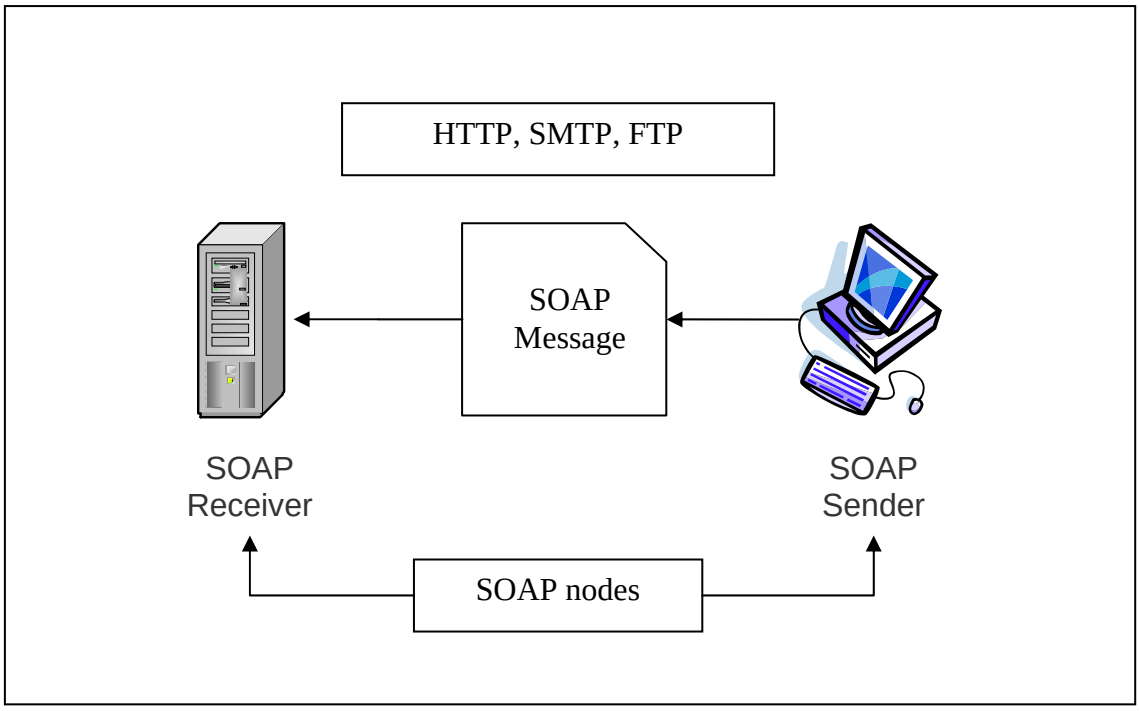
SOAP (Simple Object Access Protocol) on verkossa tapahtuvan kommunikoinnin protokolla, jonka perusta on XML. Se on avoin Internet-standardi, jonka alkuperäisiä kehittäjiä ovat olleet Microsoft, IBM ja Ariba. Kehitystyön aloittajien lähtökohtana kehitystyössä oli Internet-liikenteen optimointi, protokollan helppokäyttöisyys ja tehokas XML-integraatio (Jorgensen, 2002).

SOAP käyttää XML-kieltä ja HTTP-protokollaa tehtäessä RPC-kutsuja (Remote Procedure Calls). Ensimmäinen 1.0-versio julkistettiin, kun Visual Studio.NET esiteltiin ensimmäisen kerran suurelle yleisölle syksyllä 1999. Myöhemmin samana syksynä SOAP esiteltiin myös W3C-yhteistyöelimille. Seuraavana vuonna Microsoft julkaisi työkalun, nimeltään Visual Studio 6 SOAP Toolkit, joka käytti tätä uutta teknologiaa. Samana vuonna 2000 tuli valmiiksi myös seuraava versio SOAP 1.1, esiteltäväksi W3C-yhteistyöelimille. Useat suuret ohjelmistoalan toimijat, kuten esimerkiksi Sun Mirosytems, lähtivät nyt mukaan tukemaan Web Services -mallia, jonka etäkutsut on toteutettu käyttäen SOAP-kuuntelijaa (Franklin, 2002).

5.4.1 SOAP-arkkitehtuuri

SOAP on XML-rakenne, jonka avulla kuvataan funktioiden kutsut ja niiden parametrit. SOAP-standardia ei pidä sekoittaa XML-standardiin, koska kyse on eri asiasta. SOAP standardi määrittelee, miltä XML näyttää SOAP-dokumentissa, kuinka tiedot lähetetään ja kuinka tietojen käsittely tapahtuu sekä lähettäjän että vastaanottajan osalta. SOAP voidaan jakaa kahteen eri osaan, joita ovat tietojen siirto (Transmission) ja viestien käsittely (Message). Ennen kuin SOAP keksittiin, monet ohjelmistojen kehittelijät tekivät itse tarvitsemansa menetelmät XML-dokumenttien siirtämiseen verkossa. SOAP helpotti ohjelmoijien työtä ja yhtenäisti perin erilaiset käytännöt XML-tyyppisten tietojen siirrossa. Nykyisin käytössä olevan SOAP-standardin versionumero on 1.2, ja se on julkaistu vuonna 2003 (Hochgurtel, 2003).

SOAP mahdollistaa alustariippumattomat etäkutsut verkon yli, jolloin ei tarvitse välittää minkä tyyppinen prosessi esimerkiksi Web-palvelun toisessa päässä on. Kun etäkutsut toteutetaan käyttäen apuna SOAP-protokollaa, yhteys ei katkea palomuuriin, koska sen käyttämänä tiedonsiirtoprotokollana voi olla HTTP, SMTP tai FTP. Kun palvelin käyttää SOAP-kuuntelija, joka kuuntelee saapuvia SOAP-kutsuja, voidaan viestit välittää suoraan asianomaisille komponenteille. SOAP-kuuntelija on vain rakennettava hyväksymään web-palveluiden kutsuja (Franklin, 2002).



Kuva 27: SOAP-transaktio (Hochgurtel, 2003).

Kuvassa 27 on esitetty yksinkertainen SOAP-transaktio. SOAP antaa sovelluskehittäjille mahdollisuuden tehdä ohjelmistoja, joiden tiedonsiirrossa voidaan käyttää useita eri protokollia, mikä antaa lisää joustavuutta varsinkin silloin, kun tiedonsiirtoyhteys ei ole paras mahdollinen (Hochgurtel, 2003).

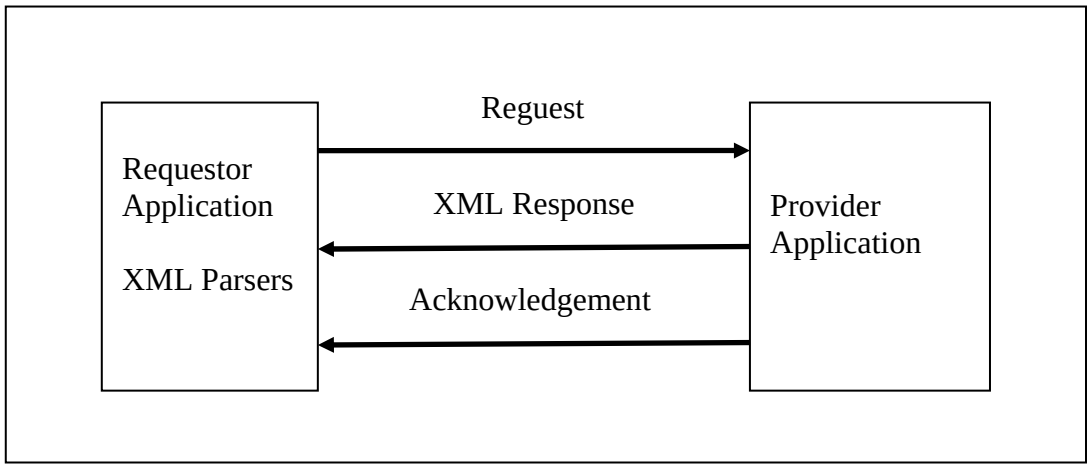
5.4.2 SOAP ja .NET.SDK

.NET SDK sisältää Soap-Proxy-luokat, jotka helpottavat asiakasohjelmien kirjoittamista. Proxy-luokka periytyy System.Web.Services.Protocols.SoapHttpClientProtocol-kantaluokasta ja se sisältää sekä funktioiden synkronisen että asynkronisen version.

Proxy-luokan voi kirjoittaa millä tahansa työkalulla, mutta vähemmällä koodin kirjoittamisella pääsee, kun antaa .NET SDK sisältämän Wsdl.exe-nimisen apuohjelman tehtäväksi luokan generoinnin. Proxy-luokan luominen tapahtuu siten, että apuohjelma lukee WSDL-tiedostossa olevan web-palvelun kuvauksen ja muodostaa sen perusteella Proxy-luokan sen metodien käyttämiseen. Metodit voidaan toteuttaa useimmilla nykyisin käytössä olevilla ohjelmointikielillä, kuten C#, Visual Basic .NET, Perl, Java, Python jne. (Platt, 2001).

5.4.3 SOAP-komponentit

SOAP-arkkitehtuuri koostuu kolmesta pääkomponentista, joita ovat: SOAP-viesti eli SOAP-message, SOAP etäproseduurikutsut (SOAP Remote Procedure Calls, RPC) ja SOAP-koodaussäännöt (SOAP encoding rules).



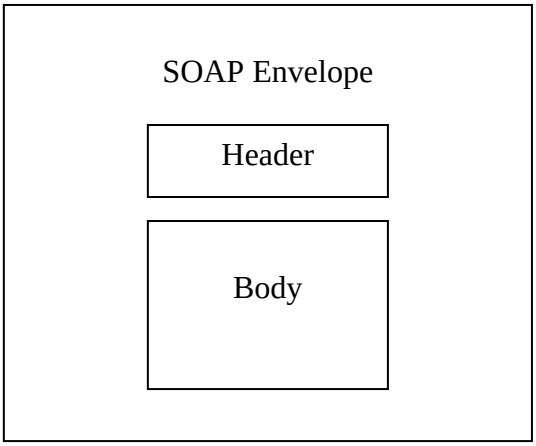
Kuva 28: SOAP-tiedonsiirtoprosessi (Hochgurtel, 2003).

SOAP-viesti on yhdensuuntaista viestintää lähettävältä sovellukselta vastaanottavalle sovellukselle. Se voi sisältää lähetyspyynnön tarvitsemistaan tiedoista vastaanottavalta sovellukselta. Kun vastaanottava sovellus on käsitellyt pyynnön, niin se lähettää vastauksen toisessa SOAP-viestissä pyytävälle sovellukselle. SOAP-viestit mahdollistavat eri koneissa olevien sovellusten keskustelun verkon yli. Kuvassa 28 on esitetty, kuinka tietojen vaihto tapahtuu SOAP-viestien avulla (Geetanjali, 2002).

Aikaisemmin käytössä olleet RPC-kutsuja välittävät protokollat kuten DCOM (Distributed Component Object Model), eivät aina toimineet, koska palomuuri esti niiden liikenteen. SOAP käyttää porttia 80, samalla tavalla kuin HTML, jolloin tietoliikenne kulkee myös palomuurin läpi (Jorgensen, 2002).

5.4.3.1 SOAP-message

SOAP-viestin (SOAP-message) komponentteja ovat: SOAP Envelope, SOAP Header ja SOAP Body (Geetanjali, 2002). Envelope-elementin voi käsittää kirjekuoreksi, jonka sisällä viestit lähetetään, Header-elementti pitää sisällään otsikkorakenteen tiedot ja kolmas elementti Body koostuu varsinaisen viestin sisällöstä. SOAP-viestin komponentit muodostavat sisäkkäisen rakenteen, jossa SOAP-Envelope on juurielementti ja sen sisällä olevat Header sekä Body ovat sen lapsielementtejä (kuva 29).



Kuva 29: SOAP-viestin komponentit (Hochgurtel, 2003).

5.4.3.2 SOAP-Envelope

SOAP Envelope on elementti, jonka tehtävä on olla toimeksiantajana SOAP-viestissä. Nimensä mukaisesti se luo kehykset SOAP-viestille ja sisältää tärkeää tietoa viestistä, kuten lähettävän sovelluksen ja vastaanottavan sovelluksen tiedot. Kuvassa 30 on esitetty esimerkki Envelope-elementistä.

```
<SOAP ENV:Envelope xmlns:SOAP ENV=
      ”http://shemas.xmlsoap.org/soap/envelope/”>
</SOAP ENV:Envelope>
```

Kuva 30: SOAP-viestin Envelope-elementti (Hochgurtel, 2003).

5.4.3.3 SOAP-Header

SOAP Header on valinnainen elementti, joka tehtävä on toimia lisäyksenä Envelope-elementille ja se on myös suoraan Envelope-elementin lapsielementti. SOAP-viesti sisältää niin monta Header-elementtiä kuin tiedonvälityksen kannalta on tarpeen. Heder-elementti puolestaan voi pitää sisällään esimerkiksi ohjeita siitä, kuinka vastaanottavan sovelluksen tulee käsitellä SOAP-viestiä (Hochgurtel, 2003).

Kuvassa 31 on esitelty, kuinka SOAP on sisäkkäinen rakenne samalla tavalla kuin XML, ja siksi sen sisältämiä tietoja voidaan myös sijoittaa Header-elementin alielementteihin, jotka voivat sisältää attribuutteja. Ehtona kuitenkin on, että niiden tulee olla suoraan Header-elementin alielementeissä.

```
<SOAP HEADER>
        <m:l xmlns:m=”http://www.someUrl/locale/”>
                <m:currency>dollars</m:currency>
                <m:country>US</m:country>
        </m:l>
</SOAP HEADER>
```

Kuva 31: SOAP-viestin Header-elementti (Hochgurtel, 2003).

5.4.4 Rajoitteet

Vaikka SOAP on hyvin käyttökelpoinen protokolla, on sillä myös joitakin rajoitteita jotka on otettava huomioon. Kun tietoja lähetetään verkossa SOAP-protokollaa käyttäen, on tiedot ensin pakattava SOAP-paketiksi. Koska paketissa tiedot esitetään XML-muodossa, joudutaan dokumentti muokkaamaan käyttäen apuna XML-jäsennintä. Tämä menetelmä käyttää aika paljon muistia ja prosessoritehoa, mikä on otettava huomioon prosessien aikavaativuutta laskettaessa. SOAP ei salli virheiden tarkastuksia käännöksen aikana, vaan testaus joudutaan tekemään ajon aikana. Kun eri organisaatiot käyttävät XML-skeemaa, toiminnan kannalta on aina huomioitava, että se on aina standardin mukainen (Geetanjali, 2002).

5.5 Synkroninen ja asynkroninen web-palvelu

Verkoissa tapahtuvalle tietoliikenteelle on ominaista, että palveluiden vasteaika vaihtelee. Palvelimet voivat tarjota vain rajallisen määrän toimintoja ja siten vastata vain tiettyyn määrään palvelupyyntöjä. Palvelinten resurssit ovat koko ajan lisääntyneet, mutta edelleen tietoliikennettä rajoittavat ne resurssit, jotka muodostavat järjestelmän heikoimman lenkin. Kun palveluiden käyttäjät ovat tottuneet nopeaan palveluun, niin jo muutamien sekuntien odottelu tuntuu ikuisuudelta.

Web-palveluja kehitettäessä on katsottu tarpeelliseksi luoda kaksi erilaista toimintatapaa tietojen siirrossa. Proxy-luokka sisältää nämä molemmat tavat, jotka ovat funktiokutsujen synkroninen ja asynkroninen versio. Proxy-luokan synkronisessa funktiokutsussa tarvittavat parametrit välitetään verkon yli palvelimelle ja asiakassovellus jää odottamaan paluuviestiä. Jos palvelupyyntöön ei heti vastata, silloin voi käydä niin, että sovellus jähmettyy paikalleen useiksi sekunneiksi (Platt, 2001). Synkroninen funktiokutsujen versio ei siis voi toimia kovinkaan hyvin vilkkaassa Internet-liikenteessä, koska siellä viivettä tulee joka tapauksessa.

Hyvin tehty asiakassovellus ei voi odotella jähmettyneenä paikallaan, vaan toiminnan on jatkuttava, vaikka palvelupyyntöön ei heti vastattaisikaan. Tähän päästään kun asiakassovelluksen toiminta ja palvelupyynnöt voitavat toimia eri säikeissä. Tämän kaltainen asiakas-palvelin-sovellus on ollut hankala kirjoittaa, koska sekä palvelimella että asiakassovelluksessa on oltava oma koodi pyyntöjen käsittelyyn.

Ratkaisuna ongelmaan web-palvelutekniikka tarjoaa Proxy-luokassa asynkronisen menetelmän tiedonsiirrolle. Se toimii niin, että asiakassovellus lähettää palvelupyynnön omassa säikeessä palvelimelle niin, ettei sen tarvitse odottaa paluuviestiä, vaan se voi palata jatkamaan omia toimintojaan. Palvelupyyntöjä voidaan lähettää useita, vaikka peräkkäin ja palvelimen tehtäväksi jää niihin vastaaminen. Kun palvelin on saanut tehtävänsä suoritettua ja vastaukset ovat valmiina, asiakassovellus voi kutsua metodia, joka osaa hakea paluuviestin. Tämä voidaan tehdä joko heti, kun paluuviesti on valmis, tai sitten kun asiakassovellus vastausta tarvitsee. Enää ei tarvitse kirjoittaa aikariippuvaisia metodeja (Platt, 2001).

6 UDDI

Ennen kuin sovelluskehittäjät voivat käyttää web-palvelua se täytyy löytää. UDDI (Universal Discovery Descriptionand Integration) tarjoaa standardoidun tavan julkaista näitä palveluja suurelle yleisölle. UDDI kuvaa ja määrittää rekisterin, jonne sovelluskehittäjät voivat rekisteröidä tarjolla olevia palvelujaan (Hochgurtel, 2003).

UDDI, joka kehitystyön aloittajina ovat olleet Ariba, IBM ja Microsoft, on teollisuusstandardi, jonka avulla asiakkaat voivat rekisteröidä, julkaista ja etsiä web-palveluja. UDDI-rekisteri on alustariippumaton ympäristö, joka tarjoaa esittelyn jokaisesta siellä olevasta web-palvelusta, palvelun sisältämistä metodeista, tiedot siitä kuinka asiakassovellus voi kommunikoida web-palvelun kanssa ja kuinka sovelluskehittäjät saavat yhteyden UDDI-rekisteriin (Geetanjali, 2002).

UDDI on rakennettu toteuttamaan ensisijaisesti teollisuuden tarpeita ja sen pääasialliset tehtävät ovat web-palveluiden rekisteröinti ja sille toimitettujen tietojen tallettaminen sekä sen sisältämien rekisteritietojen päivittäminen. UDDI voisi tulevaisuudessa paitsi julkaista web-palveluja, niin myös rekisteröidä organisaatioita, rekisteröidä web-palvelujen käyttäjiä, tiedustella potentiaalisia web-palveluja ja paikallistaa yhteistyökumppaneita (Geetanjali, 2002).

6.1 DISCO

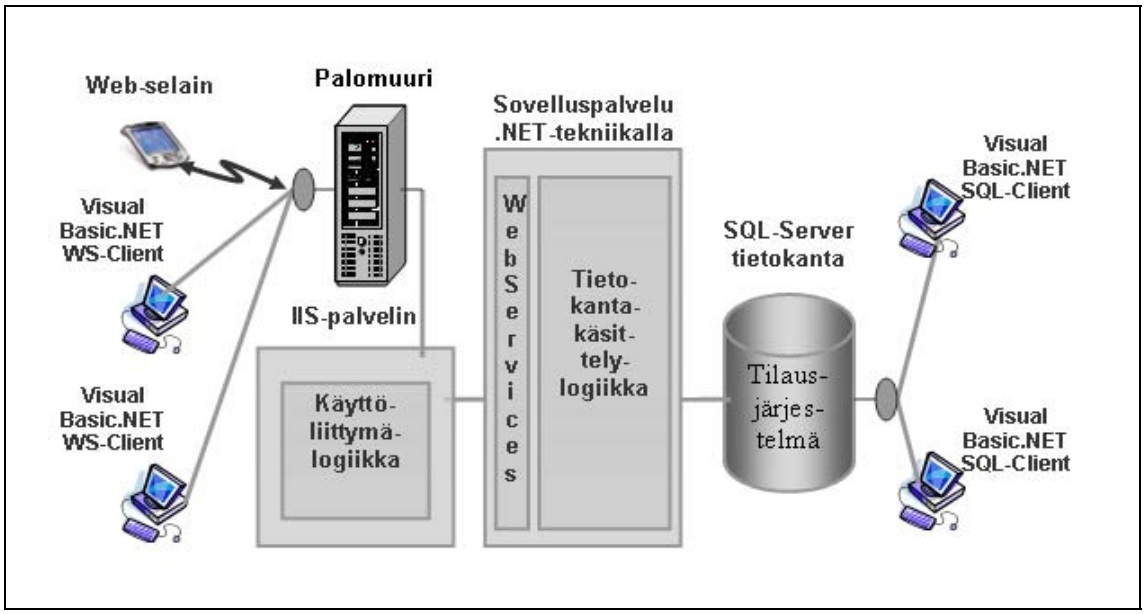
Kun sovelluskehittäjä on tunnistanut omia tarpeitaan vastaavan web-palvelun, niin UDDI:sta hän saa palvelun URL-osoitteen. Tämän jälkeen hän voi käyttää URL-osoitetta kyseisen palvelun WSDL-dokumentin hakemiseen. Jos web-palvelun tekijällä on ollut käytössään Microsoftin Visual Studio.NET niin web-palvelimelle on tällöin lisätty DISCO (Discovery of Web Services) -tiedosto.

DISCO on uusi teknologia, joka mahdollistaa WSDL-tiedostojen nopean etsimisen Intenetistä. Kun vastaavasti ohjelmistosuunnittelijalla, joka tekee web-palvelua käyttävää sovellusta, on käytössään Visual Studio.NET, niin hän voi myös käyttää WSDL-dokumentin hakemisessa apunaan samaa DISCO-teknologiaa. DISCO nopeuttaa kyseisen WSDL-dokumentin etsintää ja sen avulla löydetty WSDL-dokumentti voidaan lisätä Visual Studio.NET:in resursseihin. Tämän jälkeen ohjelmistosuunnittelija voi käyttää sitä kuin mitä tahansa komponenttia (Franklin, 2002).

7 WEB-PALVELUN LUOMINEN

Esimerkkisovelluksena tehty ohjelmisto on web-palvelukokonaisuus, joka sisältää web-palvelun, tietokantapalvelun ja asiakassovelluksen. Web-palvelu on nimeltään TilausjarjestelmaWS, ja se toimii Microsoftin IIS-palvelimella. IIS-palvelin on asennettu tietokoneelle nimeltään Pomi1 ja se sisältää myös IIS-palvelimen hallinta- ja valvontatyökalut. Samalla koneella sijaitsee tietokanta joka on nimeltään Tilausjarjestelma ja se toimii Microsoftin SQL Server palvelimella. Tietokanta ja sitä käyttävä .NET-asiakassovellus on kuvattu lähteessä Giren (2004). Asiakassovelluksen avulla tietokannan tietoihin päästään käsiksi myös suoralla tietokantayhteydellä, ilman web-palvelua. Suoran tietokantayhteyden avulla tyhjä tietokanta voidaan alustaa ja syöttää ensimmäiset tiedot tietokantaan.

Pomi1-koneelle on asennettu myös ohjelmallinen F-Securen palomuuri. Palomuurin avulla palvelua käyttävien asiakassovellusten yhteyspyyntöjä on mahdollista tarkkailla käytön aikana ja samalla se toimii testialustana, kun lähetetään pyyntöjä verkon yli. Koska käytössä on SOAP, niin pyynnöt eivät silloin pysähdy, vaikka käytössä on palomuuri (kuva 32).



Kuva 32: Tilausjärjestelmä-palvelun arkkitehtuuri.

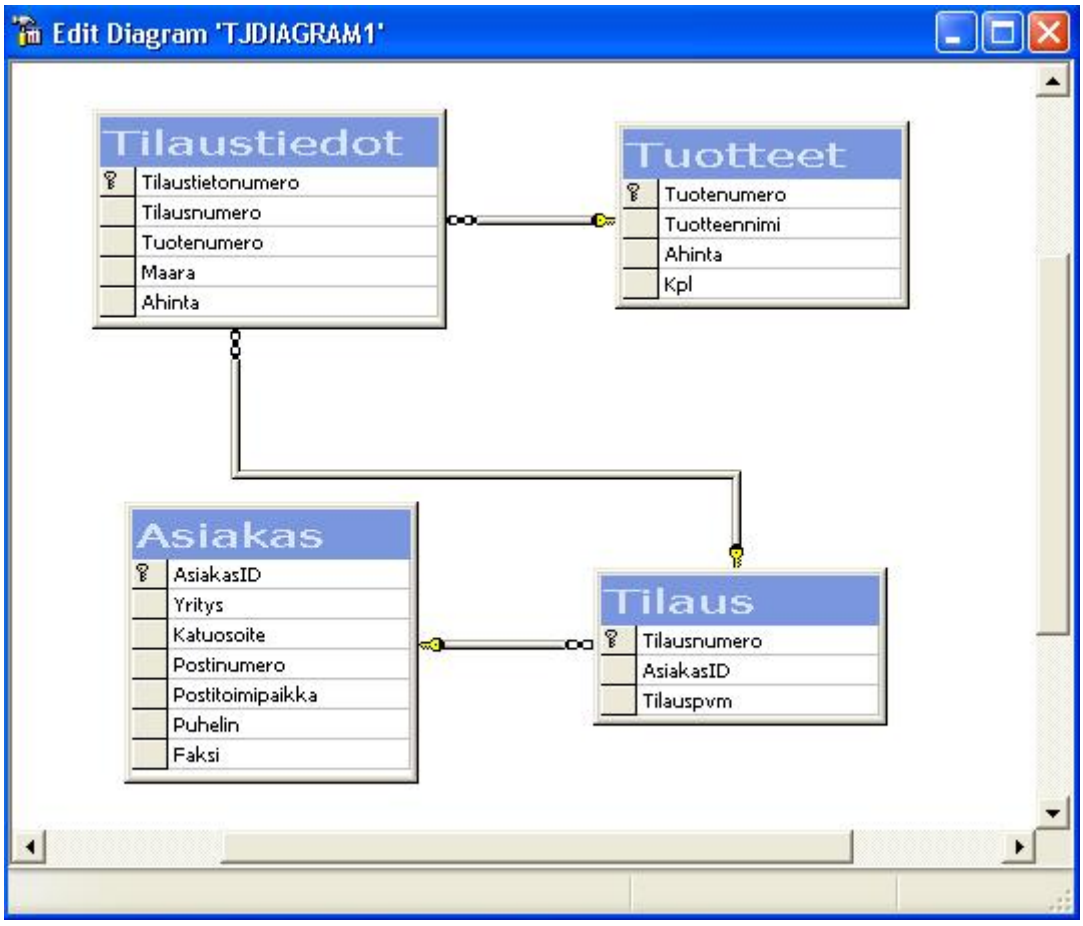
Esimerkkisovelluksen web-palvelu TilausjarjestelmaWS on luotu ensin Visual Studion versiolla 7, jonka jälkeen se on päivitetty Visual Studion versiolla 8. Päivitys ei muuttanut varsinaista palvelun toimintaa, vaan se pysyi ennallaan. Sen sijaan asiakassovellus muuttui hieman, koska uudessa versiossa on tehty pieniä muutoksia mm. lomakekomponenteissa.

Esimerkkisovelluksen toteutuksessa käytettiin seuraavia työkaluja:

- Toimintaympäristö: Microsoft .NET Framework, versio 2.0.50727.
- Ohjelmointiympäristöt: Microsoft Visual Studio 2003.
  Microsoft Visual Studio 2005, versio 8.0.50727.762.
- Ohjelmointikieli: Visual Basic .NET.
- Koodin analyysi: Project Analyzer, versio 8.1.02
- Tietokanta: Microsoft SQL-Server, versio 7.0.
- Tekstieditori: Microsoft Muistio, versio 5.1.
- Internet-palvelin: Microsoft IIS-server, versio 5.1.
- Internet-selain: Microsoft Internet Explorer 7, versio 7.0.5730.11.

7.1 Web-palvelun tietokanta

Web-palvelun tietokanta toimii Mikrosoftin SQL Server -palvelimella ja tietokannan nimi on Tilausjarjestelma. Se sisältää neljä taulua joiden nimet ovat: Tilaustiedot, Tuotteet, Asiakas ja Tilaus (kuva 33). Asiakastaulu sisältää asiakkaan perustiedot. Tuotetaulussa sijaitsee toimitettavat tuotteet. Kun asiakas tekee tilauksen, siitä tallennetaan tiedot Tilaustauluun, josta käy ilmi tilausnumero, kuka on tilannut ja milloin tilaus on tehty. Tietokantarakenne on suunniteltu siten, että Tilaustietotaulu yhdistää kaksi taulua Asiakas ja Tuotteet toisiinsa niin, että siitä saadaan selville asiakkaan tiedot ja hänen tilaamansa tuotteiden tiedot. Tilaustietotaulussa on talletettuna myös tilattujen tuotteiden määrät sekä hinnat.

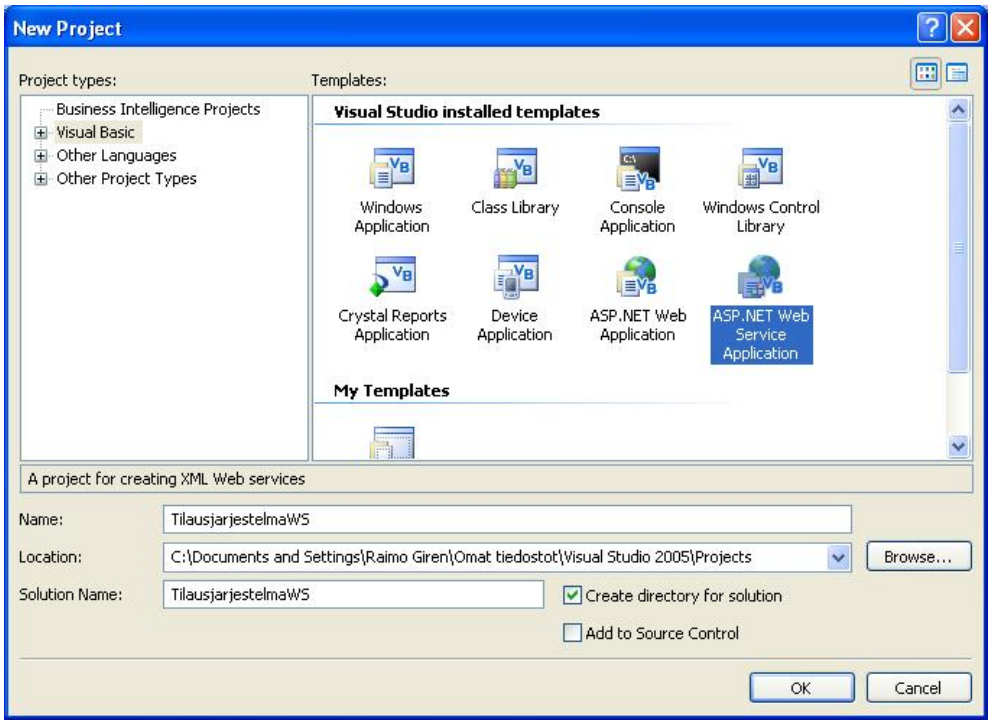


Kuva 33: Tilausjärjestelmä-palvelun tietokannan taulut (Giren 2004).

7.2 Web-palvelu TilausjarjestelmaWS

Web-palvelun luomiseen ei tarvitse käyttää mitään ohjelmistoympäristöä, vaan palvelun koodin voi kirjoittaa vaikka Muistiolla. Kun valmiin koodin tallettaa ASP.NET-muodossa kansioon ja tämän jälkeen IIS-hallintatyökalulla asettaa virtuaalihakemiston osoittamaan kyseiseen kansioon, niin palvelu on valmis (Platt, 2001). Esimerkkiohjelman web-palvelu on luotu käyttäen apuna Visual Studiota. Tämän jälkeen valmis web-palvelu on sijoitettu IIS-palvelimelle, samalla tavalla kuin sen olisi tehnyt Muistiolla, mutta nyt vain ohjelmointiympäristöä apuna käyttäen. Ohjelmointiympäristö tekee asioita koodaajalle helpommaksi, koska siinä useita toimintoja on automatisoitu.

Visual Studio.NET sisältää valmiin projektityypin web-palveluiden tekemistä varten. Kun web-palvelu luodaan, sille annetaan nimi ja samalla oma projektikansio, jonka nimeksi tulee sama kuin projektin nimi (kuva 34).



Kuva 34: TilausjarjestelmaWS-palvelu-projekti.

Visual Studio.NET luo tarvittavat hakemistot automaattisesti ja sijoittaa virtuaalikansion IIS-palvelimelle, kun se on käytössä. Samalla luodaan myös kaikki palvelun tarvitsemat tiedostot, kuten Web.Config-tiedosto, joka on XML-muotoinen tiedosto, jonka tehtävänä on kertoa palvelimelle, kuinka palvelua tulee käsitellä ajon aikana. Lisäksi muodostetaan myös TilausjarjestelmaWS.VSDISCO-tiedosto, jonka tehtävänä on kontrolloida asiakassovelluksen käyttäytymistä niiden käyttäessä web-palvelua, kuten esimerkiksi: mitä alikansioita saa sallia käytettäviksi, kun WSDL-dokumenttia käytetään. Varsinainen web-palvelun koodi sijoitetaan TilausjarjestelmaWS.ASMX -tiedostoon, joka tehtävänä on olla asiakasohjelmistojen pyyntöjen kohde (Platt, 2001).

Microsoftin .NET Services -laajennus on rakennettu siten, että varsinainen ASMX-tiedosto sisältää vain viittauksen siihen, missä varsinainen ASMX-koodi sijaitsee. Viittaus on esitelty Codebehind-ominaisuudessa, ja palvelupyyntöjä toteuttava koodi sijaitsee TilausjarjestelmaWS.ASMX.VB -tiedostossa. Ohjelmointikielenä on käytetty Visual Basic -kieltä ja lisäksi tiedostossa on esitelty toiminnot toteuttava luokka (kuva 35).

```
<%@ WebService Language="vb" Codebehind="TilausjarjestelmaWS.asmx.vb"
        Class="TilausjarjestelmaWSNET.TilausjarjestelmaWS" %>
```

Kuva 35: TilausjarjestelmaWS .ASMX.

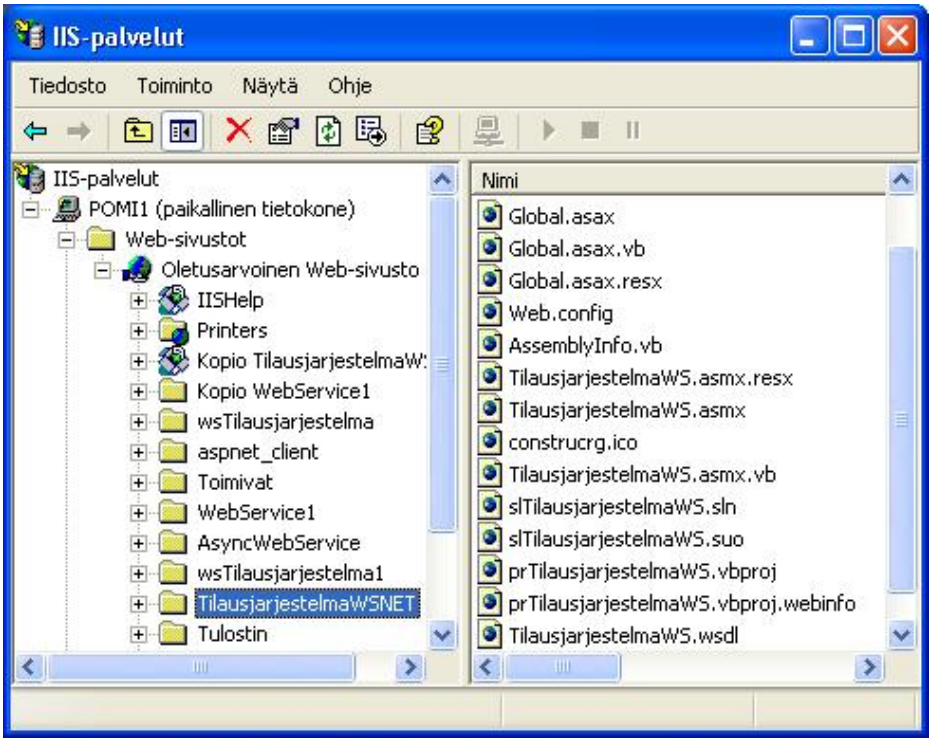
Kuvassa 36 on esitelty pieni pätkä esimerkkisovelluksena olevan web-palvelun ASMX.VB-koodista (ks. myös liite 1). Lyhenne VB tarkoittaa, että ASMX-tiedosto on koodattu käyttäen Visua Basic -kieltä. Koodin alussa Imports-komennolla tuodaan luokkakirjastot kyseisen sovelluksen koodin käyttöön, kuten Imports System.Web.Services -komento tekee Web Services -luokalle. Käytettävä nimiavaruus esitellään käyttäen Namespace-avainsanaa. Ohjelmointiympäristö tekee osan koodista valmiiksi, kuten esimerkiksi Public Sub New -avainkonstruktorin ja komponentit tuhoavan Sub Dispose -aliohjelman (Wei, 2001)

Vaikka Ohjelmointiympäristö tekeekin paljon, niin web-palvelun toteuttavat aliohjelmat ja funktiot on koodaajan kuitenkin itse kirjoitettava. Web-palvelun metodeja kutsutaan nimellä WebMethod. Jokainen kirjoitettava funktio tai aliohjelma aloitetaan muotoilulla <Web Method()> (kuva 36), joka muistuttaa hieman HTML-kieltä. Mitään muuta eroa ei Visual Basic -koodin kirjoittamisessa ole, vaan se noudattaa tässä tapauksessa samaa periaatetta kuin kirjoitettaisiin tavallista tietokantakyselyjä toteuttavaa koodia, joka sisältää SQL-kielen kyselyjä. Eron huomaa vasta kun määritellään yhteys tietokantaan. Kun kyseessä on IIS-palvelimella toimiva web-palvelukoodi, niin yhteyden polku on määriteltävä tarkasti. Muuten käytössä on samanlainen rakenne tietokannan käsittelyssä kuin tietokantasovelluksessa, jossa DataSet-olioon haetaan tarvittavat tiedot tietokannasta, niitä muokataan tarvittaessa ja muutetut tiedot palautetaan tietokantaan.

```
Imports System.Web.Services
Imports System.Data, System.Data.SqlClient
<System.Web.Services.WebService _
(Name-
space:="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS")> _
Public Class TilausjarjestelmaWS
   Inherits System.Web.Services.WebService
   <WebMethod()> _
   Public Function dstAsiakkaat() As DataSet
      Dim cnYhteys As SqlConnection = New SqlConnection
      Dim daAsiakkaat As SqlDataAdapter = New SqlDataAdapter
      Dim cmmAsiakkaat As SqlCommand = New SqlCommand _
      ("SELECT * FROM Asiakas", cnYhteys)
      dstAsiakkaat = New DataSet
      With cnYhteys
         .ConnectionString = "workstation id=Pomi1;packet size=4096;" _
         & "integrated security=Yes;data source=Pomi1;" _
         & "persist security info=False;initial catalog=sql_tilausjarjestelma"
         .Open()
      End With
      daAsiakkaat.SelectCommand = cmmAsiakkaat
      daAsiakkaat.Fill(dstAsiakkaat, "Asiakas")
      cnYhteys.Close()
   End Function
End Class
```

Kuva 36: TilausjarjestelmaWS .ASMX.VB.

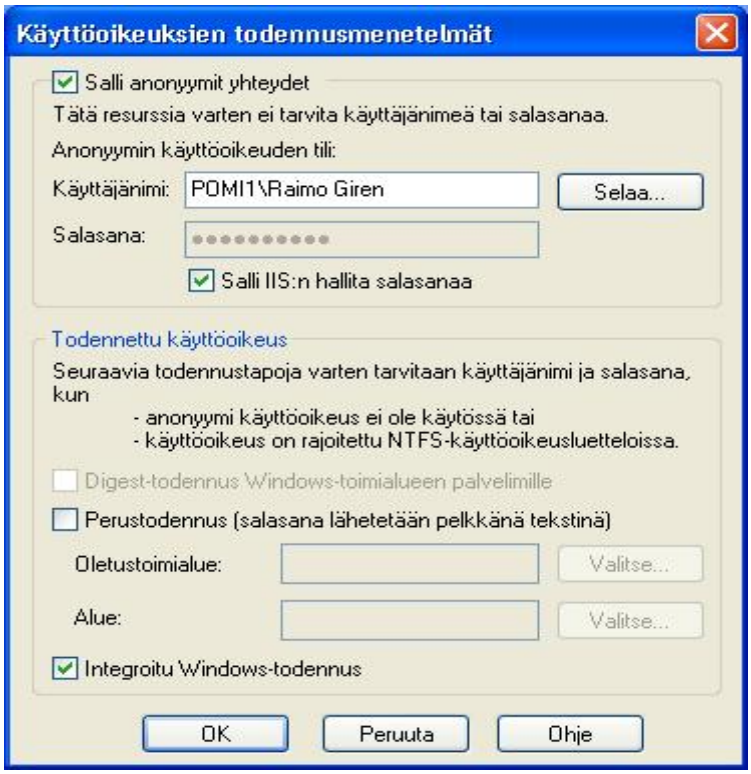
Kun aliohjelmat on kirjoitettu, niin Visual Studio.NET sijoittaa web-palvelun palvelimelle. Visual Studio.NET tunnistaa palvelimen automaattisesti ja käyttää löytämäänsä palvelinta oletuspalvelimena. Kun IIS-palvelin on asennettuna koneelle, jossa projekti käännetään, niin palvelun sijoittaminen tapahtuu IIS:lle ja samalla luodaan virtuaalihakemisto web-palvelulle (kuva 37). Tämä kaikki tapahtuu kun sovellus käännetään ensimmäisen kerran. Jos käännös onnistuu, niin web-palvelu on valmis toimimaan ja se avautuu Internet-selaimeen testausta varten automaattisesti.



Kuva 37: TilausjarjestelmaWS-palvelu IIS-palvelimella.

IIS-palvelimelle asennettu web-palvelu toimii heti ja avautuu Internet-selaimeen, jos koodissa ei ole virheitä. Palvelun käyttäminen Windows-sovelluksen kautta vaatii kuitenkin IIS-palvelimella käyttöoikeuksien muokkaamista. Esimerkkiohjelmistolle ei ole luotu mitään varsinaista turvajärjestelmää, vaan sen on tarkoitus toimia ilman niitä.

Esimerkkiohjelmiston käyttöoikeudet määritellään IIS-palvelimen hallintatyökalulla (kuva 38) siten, että virtuaalikansiolle annetaan lukuoikeudet sekä komentosarjojen suoritusoikeudet. Lisäksi käyttöoikeuksissa määritellään anonyymit yhteydet sallittaviksi, jolloin resurssia varten ei tarvita käyttäjänimeä eikä salasanaa. Anonyymille käyttöoikeudelle määritellään kuitenkin tili, jolla on käyttäjänimi, ja IIS:n sallitaan hallita salasanaa. Vaikka TilausjarjestelmaWS-palvelu on verkon yli toimiva ja kaikilta toiminnoitaan täysimittainen web-palvelu, niin se ei kuitenkaan ole Internet-palvelu. Tämän vuoksi sisäverkossa voidaan käyttää integroitua Windows-todennusta, joka tekee asiakassovelluksen käytönaikaisen tunnistamisen helpoksi.



Kuva 38: TilausjarjestelmaWS-palvelun käyttöoikeudet.

7.3 TilausjarjestelmaWS-palvelun Proxy-luokka

Web-palveluja tehtäessä apuna voidaan käyttää myös .NET SDK:n lisätyökaluja. WSDL.EXE nimisen työkalun avulla voidaan luoda ns. Proxy-luokka helpottamaan palvelua käyttävän ohjelmiston ohjelmointia. Proxy-luokan kantaluokka, eli luokka, josta se periytyy, on System.Web.Services.Protocols.SoapHttpClientProtocol. TilausjarjestelmaWS-palvelun WSDL-tiedostosta tehdyn Proxy-luokan koodista alkuosa on esitelty kuvassa 39 (ks. myös liite 3).

Proxy-luokka luodaan siten, että WSDL.EXE-apuohjelma lukee palvelun kuvauksen WSDL-tiedostosta ja tulkitsee sen toiminnot. Tämän jälkeen se generoi dokumentista automaattisesti Proxy-luokan, joka puolestaan sisältää kaikki palvelun käytössä tarvittavat funktiot ja metodit sillä ohjelmointikielellä, jolla se on generoitu.

Asiakasohjelmiston kutsuessa jotakin Proxy-luokan funktioista Proxy-luokka tekee HTTP-pyynnön ja lähettää sen palvelimelle. Vastaavasti Proxy-luokka saadessaan vastauksen palvelimelta HTTP-muodossa, muodostaa asiakasohjelmalle vastauksen, joka antaa paluuarvon kutsutulta funktiolta. Visual Basic -kielen Proxy-luokassa voidaan määritellä sekä synkroninen että asynkroninen toimintaversio (Platt, 2001).

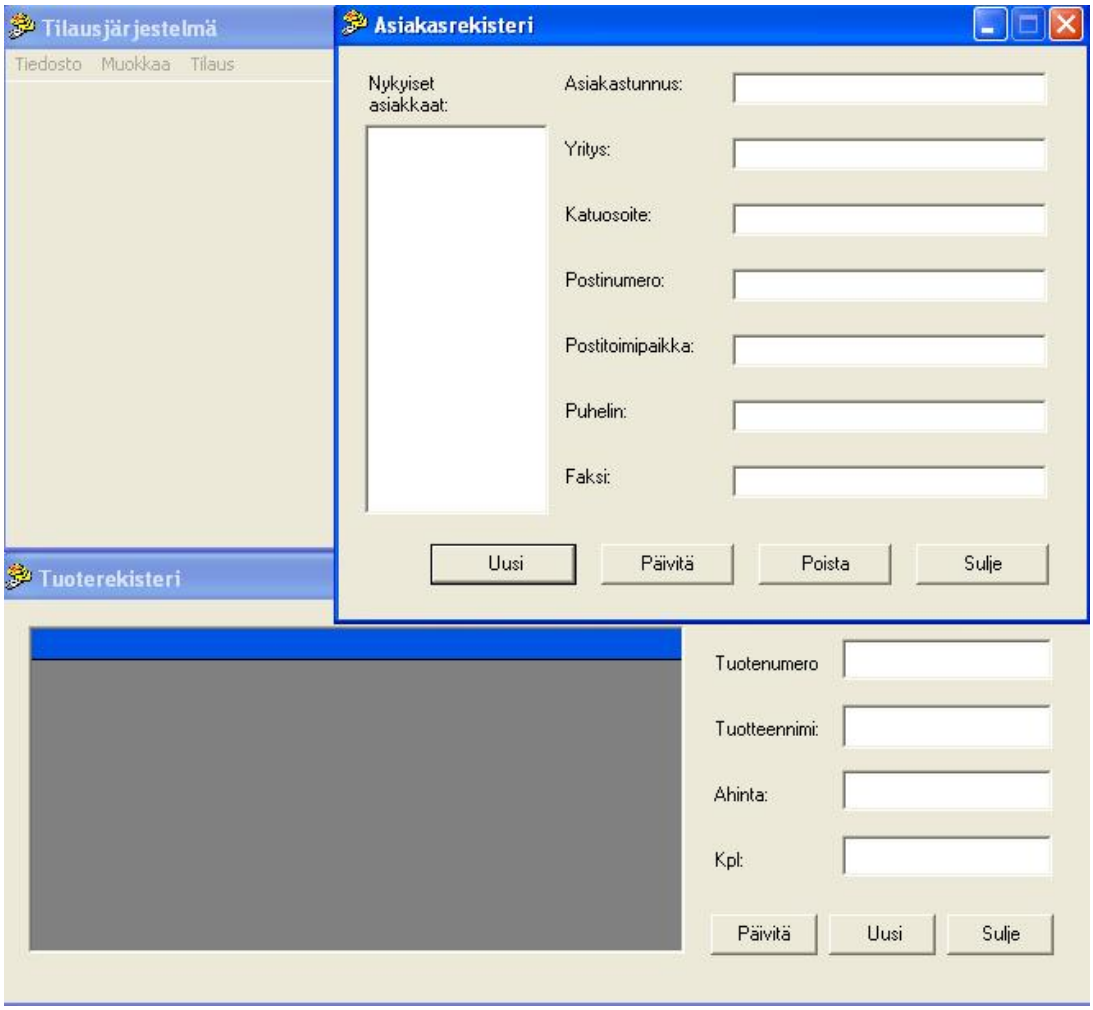
Kuvan 39 Proxy-luokka sisältää URL-nimisen ominaisuuden, jonka avulla määritellään sen palvelimen URL, jonne kutsu ohjataan: Me.Url = http://pomi1/TilausjarjestelmaWSNET/TilausjarjestelmaWS.asmx. Kyseisessä Proxy-luokassa (ks. liite 3) on esitelty myös neljä tietojoukkoa (DataSet), joiden avulla palvelua käytetään.

```
'------------------------------------------------------------
' <auto-generated>
'     This code was generated by a tool.
' </auto-generated>
'------------------------------------------------------------
Namespace localhost
<System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"2.0.50727.42"),  _
    System.Diagnostics.DebuggerStepThroughAttribute(),  _
    System.ComponentModel.DesignerCategoryAttribute("code"),  _
System.Web.Services.WebServiceBindingAttribute(Name:="TilausjarjestelmaWS
Soap",
[Namespace]:="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS
")>  _
   Partial Public Class TilausjarjestelmaWS
      Inherits System.Web.Services.Protocols.SoapHttpClientProtocol
      Private dstAsiakkaatOperationCompleted As
System.Threading.SendOrPostCallback
      Private dstAsiakkaatTallennaOperationCompleted As
System.Threading.SendOrPostCallback
      Private dstTuotteetOperationCompleted As
System.Threading.SendOrPostCallback
      Private dstTuotteetTallennaOperationCompleted As
System.Threading.SendOrPostCallback
      Private useDefaultCredentialsSetExplicitly As Boolean
      '"<remarks/>
      Public Sub New()
         MyBase.New
         Me.Url =
"http://pomi1/TilausjarjestelmaWSNET/TilausjarjestelmaWS.asmx"
         If (Me.IsLocalFileSystemWebService(Me.Url) = true) Then
            Me.UseDefaultCredentials = true
            Me.useDefaultCredentialsSetExplicitly = false
         Else
            Me.useDefaultCredentialsSetExplicitly = true
         End If
      End Sub
```

Kuva 39: TilausjarjestelmaWS:n Proxy-luokka.

7.4 Web-palvelua käyttävä Windows-sovellus

TilausjarjestelmaWS-palvelua käyttävä asiakassovellus on tavallinen Windows-ympäristössä toimiva ohjelmisto (kuva 40), joka käyttää tietokantaa verkon yli web-palvelun avulla. Ohjelmiston käyttöliittymä koostuu kolmesta lomakkeesta joita ovat: päälomake, asiakasrekisteri ja tuoterekisteri. Lomakkeet on luotu käyttäen Windowsin peruskomponentteja. Päälomakkeelta voi avata sovelluksen muut lomakkeet. Asiakasrekisterilomakkeella sijaitsevat asiakkaantiedot ovat: Asiakastunnus, Yritys, Katuosoite, Postinumero, Puhelin ja Faksi. Tuoterekisterilomake sisältää tuotteiden tiedot kuten: Tuotenumero, Tuotteennimi, Kappalehinta (Ahinta) ja Lukumäärä (Kpl).



Kuva 40: Asiakassovelluksen lomakkeet.

Koska päälomakkeen tehtävänä on toimia muiden lomakkeiden avaajana, niin sen koodi on tavallista Windows-sovelluksen koodia. Asiakasrekisterilomakkeelle sen sijaan haetaan tietokannasta asiakkaiden tiedot ja niitä voi myös muokata samalla lomakkeella.

Web-palvelua käyttävä Windows-sovellus tarvitsee tiedon siitä, mistä kyseinen palvelu löytyy. Vaikka kyseessä ei olekaan Internet-palvelu, voidaan nyt käyttää ominaisuutta, jonka Microsoftin Visual Studio.NET sisältää, eli UDDI-rekisterin kautta tapahtuvaa palveluiden hakua. Kun UDDI-työkalun avaa, niin se etsii kaikki mahdolliset käytettävissä olevat web-palvelut, vaikka ne sijaitsevatkin nyt Intranet-verkossa. Kun TilausjarjestelmaWS-palvelu on löytynyt, niin se lisätään Visual Studio.NET -projektin Web Referensiksi.

TilausjarjestelmaWS-palvelu on rakennettu käyttämään Proxy-luokkaa. Se helpottaa koodin kirjoittamista siten, että jokaiseen käytettävään web-palvelun metodiin voi viitata samalla tavalla kuin tavallisessa Visual Basic.NET -koodissa. Proxy-luokka on luotu käyttäen WSDL.EXE-apuohjelmaa lukemaan palvelun kuvauksen WSDL-tiedostosta. WSDL.EXE muodostaa kuvauksen tiedoista Proxy-luokan. Proxy-luokan avulla Visual Studio.NET saa käyttöönsä kaikki palvelun funktiot ja metodit Visual Basic.NET-kielellä kirjoitettuna.

Kun web-palvelu sijaitsee paikallisverkossa ja se on lisätty projektin Web Referenssiksi, niin palvelu löytyy, kun kirjoitetaan Localhost ja käytetään pistenotaatiota. Tietojoukko wsdstAsiakkaat esitellään Dim-avainsanalla ja New-avainsanalla luodaan uusi olio Localhost.TilausjarjestelmaWS-palvelusta (kuva 41). Esimerkkisovelluksen Taytaasiakaslista-aliohjelmassa koko lomakkeen käytössä olevaan dstAsiakkaat-tietojoukkoon sijoitetaan tiedot wsdstAsiakkaat-tietojoukosta. Tämän jälkeen Asiakas-taulun asiakkaat järjestetään Label-komponenttiin Yritys-kentän mukaan.

```
Private Sub Taytaasiakaslista() ' ok
    Dim wsdstAsiakkaat As New localhost.TilausjarjestelmaWS
    dstAsiakkaat = wsdstAsiakkaat.dstAsiakkaat
    lbAsiakkaat.DataSource = dstAsiakkaat.Tables("Asiakas")
    lbAsiakkaat.DisplayMember = "Yritys"
End Sub
```

Kuva 41: Aliohjelma Taytaasiakaslista.

7.4.1 Asynkroninen TilausjarjestelmaWS-palvelu.

Koska TilausjarjestelmaWS-palvelu on verkon yli toimiva tietokantasovellus, niin silloin on huomioitava aikariippuvuus. Tietokannan käsittely vie aina oman aikansa ja tietokannan tiedoista muodostuu helposti paljon siirrettävää dataa. Tämän vuoksi esimerkkisovellus on toteutettu käyttäen Proxy-luokan asynkronista versiota funktiokutsuista. Sovellusta voi käyttää silloinkin, kun se ei vielä ole saanut tarvitsemiaan tietoja tietokannasta.

Esimerkkisovelluksen asiakaspuolen näkökulma asiakastietojen päivitykseen on esitetty kuvassa 42 (ks. myös liite 4). Aliohjelmassa cmdPaivita_Click esitellään ensin AsyncResult-objekti, joka on tyyppiä IAsyncResult. Sen avulla palvelimen vastaus palautetaan asiakasohjelmalle asynronisesti. Sama objekti esiintyy myös automaattisesti generoidussa Proxy-luokassa (kuva 43). Seuraavalla rivillä esitellään palvelinobjekti wsdstAsiakkaatTallenna, joka on tyyppiä Newlocalhost.TilausjarjestelmaWS, jonka tehtävänä on päivityksen tekeminen tietokantaan.

Kuvan 42 koodissa on käytössä tyypittämätön DataSet-objekti dstAsiakkaat, josta seuraa, että sen sisältämiä tietoja käsitellään indeksin mukaan eikä suoralla viittauksella kyseisen taulun riville tai kenttään. Lomakkeen tiedot sijoitetaan tähän dstAsiakkaat-objektiin, jonka jäkeen AsyncResult-objektille annetaan tehtäväksi käynnistää asynkroninen palvelu kohteessa wsdstAsiakkaatTallenna komennolla BegindstAsiakkaatTallenna ja sijoittaa sekä palauttaa totuusarvo pyydettäessä tehtävän

onnistumisesta. BegindstAsiakkaatTallenna saa parametrinaan mukaansa mm. dstAsiakkaat-objektin, jonka lisäksi on aina määriteltävä Nothing-parametreinä myös ne arvot, joita ei käytetä.

```
Private Sub cmdPaivita_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles cmdPaivita.Click ' ok
 Dim AsyncResult As IAsyncResult
 Dim wsdstAsiakkaatTallenna As Newlocalhost.TilausjarjestelmaWS
 Dim i As Integer
 Dim s As String = ""
 Tarkastakentat(s)
 If s = "!" Then Exit Sub
  i = lbAsiakkaat.SelectedIndex ' Listasta valitun indeksi
  dstAsiakkaat.Tables("Asiakas").Rows(i).Item("AsiakasID")
   =txtAsiakastunnus.Text
  dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Yritys")
   =txtYritys.Text
  dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Katuosoite")
   =txtKatuosoite.Text
  dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Postinumero")
   =txtPostinumero.Text
  stAsiakkaat.Tables("Asiakas").Rows(i).Item("Postitoimipaikka")
   =txtPostitoimipaikka.Text
  dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Puhelin")
   =txtPuhelin.Text
  dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Faksi")
   =txtFaksi.Text
  AsyncResult=wsdstAsiakkaatTallenna.
   BegindstAsiakkaatTallenna(dstAsiakkaat, Nothing, Nothing)
  wsdstAsiakkaatTallenna.EnddstAsiakkaatTallenna(AsyncResult)
   If AsyncResult.IsCompleted = True Then
    MsgBox("Tiedot talletettu.", MsgBoxStyle.Information,
    Me.Text)
    Tyhjennakentat()
    Taytaasiakaslista()
   Else
    MsgBox("Talletetus epäonnistui.", MsgBoxStyle.
    Critical, Me.Text)
   End If
End Sub
```

Kuva 42: Asiakastietojen päivitys cmdPaivita_Click-aliohjelmassa.

Kun palvelupyyntö on asiakassovellukselta lähetetty, niin paluuarvoa kysytään wsdstAsiakkaatTallenna-objektin metodilla EnddstAsiakkaatTallenna, jonka parametriksi on nyt annettava AsyncResult-objekti. Jos palvelin sai palvelupyynnön toteutetuksi, niin AsyncResult-objektin IsCompleted-paluuarvona on True ja asiakassovelluksen lomake tyhjennetään sekä lomakkelle palautetaan tietokannasta uudet muuttuneet tiedot. Jos taas IsCompleted-paluuarvona on False, niin asiakassovellus antaa virheilmoituksen käyttäjälle. Virheilmoitus voidaan muotoilla uudelleen tai jättää asynkronisessa palvelupyyntöjenkäsittelyssä kokonaan pois, koska paluuviestin kysyminen palvelimelta voidaan automatisoida. Käyttäjän kannalta oleellista on vain se, että hän saa tarvitsemansa tiedot mahdollisimman nopeasti. Tarpeelliset viestit toiminnasta kannattaa antaa vasta silloin, kun jotakin poikkeavaa esiintyy. Esimerkkiohjelman tehtävänä on toimia eri toimintojen havainnollistajana, jolloin virheilmoituksen kautta käyttäjä voi saada tarvitsemansa tiedot myös siitä, kuinka nopeasti palvelu toimii.

Kuvassa 43 (ks. myös liite 3) on esitetty esimerkkisovelluksen Proxy-luokasta osa, jossa käsitellään asiakastietojen päivittämistä tietokantaan. Proxy-luokka sisältää funktion, jonka esittely on muotoa: ”Public Function BegindstAsiakkaatTallenna”. Sen parametrina on DataSet-olio. Kun funktiota kutsutaan, paluuarvona on AsyncResult-tyyppinen objekti, jonka avulla palvelimen vastaus palautetaan asiakasohjelmalle asynronisesti. AsyncResult-tyyppistä objektia voidaan kutsua myöhemmin ja välillä asiakassovelluksella voidaan tehdä jotakin muuta. BegindstAsiakkaatTallenna-funktio aloittaa palvelinpyynnön käsittelyn silloin, kun sitä kutsutaan.

```
<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempur
i.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS/dstAsiakkaatTallenn"& _
        "a",
RequestNamespace:="http://tempuri.org/TilausjarjestelmaWSNET/Tilausjarjeste
lmaWS",
ResponseNamespace:="http://tempuri.org/TilausjarjestelmaWSNET/Tilausjarjes
telmaWS", Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)
> _
    Public Function
dstAsiakkaatTallenna(<System.Xml.Serialization.XmlElementAttribute("dstAsi
akkaat")> ByVal dstAsiakkaat1 As System.Data.DataSet) As Object
        Dim results() As Object = Me.Invoke("dstAsiakkaatTallenna", New
Object() {dstAsiakkaat1})
        Return CType(results(0),Object)
    End Function
    '"<remarks/>
    Public Function BegindstAsiakkaatTallenna(ByVal dstAsiakkaat1 As
System.Data.DataSet, ByVal callback As System.AsyncCallback, ByVal
asyncState As Object) As System.IAsyncResult
        Return Me.BeginInvoke("dstAsiakkaatTallenna", New Object()
{dstAsiakkaat1}, callback, asyncState)
    End Function
    '"<remarks/>
    Public Function EnddstAsiakkaatTallenna(ByVal asyncResult As
System.IAsyncResult) As Object
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),Object)
    End Function
    '"<remarks/>
    Public Overloads Sub dstAsiakkaatTallennaAsync(ByVal dstAsiakkaat1 As
System.Data.DataSet)
        Me.dstAsiakkaatTallennaAsync(dstAsiakkaat1, Nothing)
    End Sub
    '"<remarks/>
    Public Overloads Sub dstAsiakkaatTallennaAsync(ByVal dstAsiakkaat1 As
System.Data.DataSet, ByVal userState As Object)
        If (Me.dstAsiakkaatTallennaOperationCompleted Is Nothing) Then
            Me.dstAsiakkaatTallennaOperationCompleted = AddressOf
Me.OndstAsiakkaatTallennaOperationCompleted
        End If
        Me.InvokeAsync("dstAsiakkaatTallenna", New Object()
{dstAsiakkaat1}, Me.dstAsiakkaatTallennaOperationCompleted, userState)
    End Sub
```

Kuva 43: Asiakastietojen tallennus Proxy-luokan avulla.

Proxy-luokka sisältää myös funktion: ”Public Function EnddstAsiakkaatTallenna”. Asiakasohjelma kutsuu tätä funktiota, kun se haluaa saada paluuarvon, jonka se jätti palvelimen haettavaksi. Kun paluuarvoa haetaan, välitetään funktiolle AsyncResult-objekti, joka oli paluuarvona BegindstAsiakkaatTallenna-funktiolla. Koska palvelimella voi olla useita palvelupyyntöjä jonossa, niin AsyncResult-objektin avulla se tietää, minkä palvelun paluuarvoa se on palauttamassa. Koska palvelimen on tiedettävä myös, onko kysytty palvelupyynnön vastaus valmis, niin AsyncResult-objekti sisältää totuusarvon palauttavan metodin dstAsiakkaatTallennaOperationCompleted. Jos palvelupyyntö on saatu suoritetuksi ja vastaus voidaan palauttaa, niin IsCompleted-arvo on True. Jos arvo on False niin vastausta joudutaan odottamaan, mutta palvelua voidaan kutsua automaattisesti ja säännöllisesti, mutta silti jatkaa sovelluksen toimintaa keskeyttämättä.

8 YHTEENVETO

Internet on nykyisin ja tulevaisuudessa se verkosto, jonka avulla informaatio välitetään palvelujen tarjoajilta asiakkaille. Informaation määrä tulee edelleen kasvamaan ja sen tarjonta monipuolistumaan. Nykyiset sovellukset tarjoavat monenlaisia toimintoja, joilla näitä palveluja voidaan käyttää. Ongelmana on ollut, ja on edelleen, etteivät erilaiset ohjelmistotuotteiden tuottajat täysin hyväksy yhteisiä formaatteja, vaan toimivat omien intressiensä mukaan. Markkinavoimat ja kova kilpailu ajavat ohi tervejärkisen yhteistyön. Merkkejä paremmasta tulevaisuudesta on kuitenkin ollut luvassa. Suuret ohjelmistoalan toimijat ovat yhteistyössä kehitelleet menetelmiä, joilla päästäisiin parempaan lopputulokseen kuin yksin toimien ja omaa formaattia puolustaen. Yhteisesti kehitettyjä menetelmiä on saatu aikaiseksi ja niitä on myös alettu hyväksyä standardeiksi.

Merkkauskielet ovat nousseet merkittävään asemaan, paitsi rakenteisten dokumenttien käsittelyssä, kun informaatiota järjestellään ja talletetaan, niin myös olio-ohjelmointiin perustuvien ohjelmointiympäristöjen apuvälineenä. Merkkauskielistä uusin, aiemmista kielistä kehitetty, XML on noussut suosituksi, koska sen rakenne on selkeä ja ominaisuudet riittävät moneen tarkoitukseen. Kun nyt viimein XML on yleisesti hyväksytty ja saanut standardiaseman, niin sillä on mahdollisuus yhtenäistää siirrettävää informaatiota. XML sopii hyvin myös olio-ohjelmointiin, koska sen avulla voidaan merkata olioita ja palauttaa niitä verkon yli siirrettäessä.

Microsoftin .NET-arkkitehtuuri toi yhden suuren toimijan mukaan olio-ohjelmointiin, ja samalla se avasi uuden mahdollisuuden liittää erilaisia tekniikoita yhteen. .NET-arkkitehtuuri on täysin olio-ohjelmointiin perustuva ympäristö, joka toimii Windows-alustalla. XML on tiukasti integroitu lähes jokaiselle .NET-arkkitehtuurin osa-alueelle, mutta erityisesti se on kytketty mukaan tietokanta- ja internet-sovellusten sovellusympäristöihin. Tietokannan tietoja voidaan merkata XML-muotoon, kun niitä käsitellään ja siirretään. Samoin voidaan tehdä informaatiolle, jota siirretään tietoverkkojen yli internetissä.

Web-palvelu on uusi yritys lähestyä ongelmaa, joka verkostoituneessa maailmassa usein esiintyy. Informaatiota on paljon, mutta kaikki toimivat omilla säännöillään, eikä mitään yhteistä synny. Koska enää kenenkään voimavarat eivät riitä laajojen kokonaisuuksien hallintaan, niin tarvitaan jokin sellainen formaatti, joka ei ole ajasta, paikasta eikä alustasta riippuvainen. Web-palvelu on hyvä yritys tähän suuntaan. Se käyttää monia uusia tekniikoita apunaan. Näistä tärkein on XML, koska se mahdollistaa alustariippumattoman tiedon siirron verkkojen yli niin, että informaatio ei katoa sitä liikuteltaessa. XML-kielen avulla on mahdollista liitää erilaisia alustoja käytävät tietokoneet toisiinsa. Samoin voidaan liittää erilaiset ohjelmointi- ja sovellusympäristöt toisiinsa. Koska XML on tiukasti määritelty, toisin kuin esimerkiksi HTML, niin sen tulkinnassa ei synny samanlaisia ongelmia kuin määrittelemättömille kielille on ominaista. Määrittelystä on apua aina, kun tiedetään kielen versio, siitä selviää kuinka sitä pitää tulkita.

Web-palvelussa käytetään yleisesti SOAP-protokollaa tiedon paketoinnissa. SOAP antaa mahdollisuuden tehdä paketointi niin, ettei verkossa liikuteltava informaatio pysähdy palomuureihin, koska sen käyttämänä tiedonsiirtoprotokollana voi olla HTTP, SMTP tai FTP. Kun palvelin käyttää SOAP-kuuntelija, joka kuuntelee saapuvia SOAP-kutsuja, voidaan viestit välittää suoraan asianomaisille komponenteille.

Visual Studio.NET on Microsoftin uusin integroitu sovellusympäristöjen ohjelmointityökalu, jossa on yhdistetty usean eri ohjelmointikielen käyttäminen samassa ympäristössä. Sen suunnittelussa on pitkälle huomioitu sovelluskehittäjien kasvava tarve luoda internet-sovelluksia. Siksi myös web-palvelun luomisessa Visual Studio.NET on hyvä ympäristö tehdä sovelluksia. Web-palvelun koodaamista on helpotettu monella eri tavalla, kuten erilaisilla apuvälineillä, jotka suorittavat rutiinitoimintoja melkein automaattisesti. Kun ohjelmistosuunnittelija luo Web Service -projektin niin ohjelmointiympäristö luo siihen kaikki perustoiminnot, vain varsinaiset palvelumetodit on koodattava itse.

IIS-palvelimelle sijoitettu web-palvelu on heti valmis testattavaksi, ja sitä käyttäviä asiakassovelluksia voi luoda vaikka tavallisista Windows-sovelluksista, kun lisää UDDI-työkalun avulla Web Referenssiksi kyseisen web-palvelun. WSDL-do-

kumentointi helpottaa huomattavasti palveluiden käyttämistä, koska dokumentin avulla voi generoida Proxy-luokan. Proxy-luokka antaa ohjelmistosuunnittelijalle helpon mahdollisuuden käyttää samaa ohjelmointikieltä ja samoja tuttuja metodeita joita hän on oppinut koodatessaan hyödyntämään. Valmis web-palvelua hyödyntävä asiakas-palvelinsovellus syntyy melko vaivattomasti, vaikka käytössä olisi myös perinteinen tietokantaa käyttävä sovellus. Käyttäjä tuskin edes huomaa eroa tavallisen tietokantasovelluksen tai web-palvelua käyttävän tietokantasovelluksen välillä. Siitä kuinka tiedot verkossa kulkevat, ei asiakas olekaan kiinnostunut, kunhan tiedot ovat oikeita ja ne on saatavilla juuri silloin, kun niitä tarvitaan.

Vaikka web-palvelu on toimiva malli tiedonsiirtoon ja mahdollistaa monta uutta asiaa, niin se tuskin on vielä lopullista muotoaan löytänyt, vaan koko ajan sitä kehittävät ohjelmistoalan yritykset keksivät jotkin uutta. Tiedonsiirtotekniikka tullee edelleen kehittymään ja käytettävät protokollat todennäköisesti paranevat myös.

VIITELUETTELO

Franklin, K. (2002) *VB.NET for Developers*. Sams, New York.

Geetanjali, A. (2002) *Building Web Services with XML*. Premier Press, Cincinnati.

Giren, R. (2004) *Tietokantaohjelmointi ja Visual Basic.NET*. Kandidaatintutkielma. Joensuun yliopisto, tietojenkäsittelytiede.

Hochgurtel, B. (2003) *Cross-Platform Web Services Using C# and Java*. Charles River Media, Massachusetts.

Jorgensen, D. (2002) *Developing .NET Web Services with XML*. Syngress Publishing, Rockland.

Koskimies, K. (1998) *Pieni Oliokirja*. Suomen Atk-kustannus Oy, Jyväskylä.

Microsoft Press. (1999) *Visual Basic 6.0 Programmers Guide*. Microsoft Press, Washington.

Microsoft. (2001) *Visual Studio .NET Combined Collection*. Microsoft Press, Washington.

Platt, D, S. (2001) *Introducing Microsoft.NET*. Microsoft Press, Washington.

Sceppa, D. (2002) *Microsoft ADO.NET*. Microsoft Press, Washington.

Thomsen, C. (2001) *Database Programming with Vb.NET*. Springer-Verlag, New York.

Wei Meng, L. (2001) *ASP.NET Developer's Guide*. Syngress Publishing, Rockland.

W3C (2008) The World Wide Web Consortium. *Extensible Markup Language (XML).* WWW-sivusto, http://www.w3.org/XML/ (7.6.2008).

LIITE 1: TILAUSJARJESTELMAWS.ASMX)

```
Imports System.Web.Services
Imports System.Data, System.Data.SqlClient


<System.Web.Services.WebService _
(Namespace:="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS")> _
Public Class TilausjarjestelmaWS
    Inherits System.Web.Services.WebService


#Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent() call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services Designer
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()

    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

#End Region
```

```
    <WebMethod()> _
    Public Function dstAsiakkaat() As DataSet

        Dim cnYhteys As SqlConnection = New SqlConnection
        Dim daAsiakkaat As SqlDataAdapter = New SqlDataAdapter
        Dim cmmAsiakkaat As SqlCommand = New SqlCommand _
        ("SELECT * FROM Asiakas", cnYhteys)
        dstAsiakkaat = New DataSet
        With cnYhteys
            .ConnectionString = "workstation id=Pomi1;packet size=4096;" _
            & "integrated security=Yes;data source=Pomi1;" _
            & "persist security info=False;initial catalog=sql_tilausjarjestelma"
            .Open()
        End With
        daAsiakkaat.SelectCommand = cmmAsiakkaat
        daAsiakkaat.Fill(dstAsiakkaat, "Asiakas")
        cnYhteys.Close()

    End Function


    <WebMethod()> _
    Public Function dstAsiakkaatTallenna(ByVal dstAsiakkaat As DataSet)

        Dim cnYhteys As SqlConnection = New SqlConnection
        Dim daAsiakkaat As SqlDataAdapter = New SqlDataAdapter
        Dim cbAsiakas As SqlCommandBuilder = New SqlCommand-
Builder(daAsiakkaat)
        Dim cmmAsiakkaat As SqlCommand = New SqlCommand _
        ("SELECT * FROM Asiakas", cnYhteys)
        With cnYhteys
            .ConnectionString = "workstation id=Pomi1;packet size=4096;" _
            & "integrated security=Yes;data source=Pomi1;" _
            & "persist security info=False;initial catalog=sql_tilausjarjestelma"
            .Open()
        End With
        daAsiakkaat.SelectCommand = cmmAsiakkaat
        daAsiakkaat.Update(dstAsiakkaat, "Asiakas")
        cnYhteys.Close()

    End Function
```

```
    <WebMethod()> _
    Public Function dstTuotteet() As DataSet

        Dim cnYhteys As SqlConnection = New SqlConnection
        Dim daTuotteet As SqlDataAdapter = New SqlDataAdapter
        Dim cmmTuotteet As SqlCommand = New SqlCommand _
        ("SELECT * FROM Tuotteet", cnYhteys)
        dstTuotteet = New DataSet
        With cnYhteys
            .ConnectionString = "workstation id=Pomi1;packet size=4096;" _
            & "integrated security=Yes;data source=Pomi1;" _
            & "persist security info=False;initial catalog=sql_tilausjarjestelma"
            .Open()
        End With
        daTuotteet.SelectCommand = cmmTuotteet
        daTuotteet.Fill(dstTuotteet, "Tuotteet")
        cnYhteys.Close()

    End Function


    <WebMethod()> _
    Public Function dstTuotteetTallenna(ByVal dstTuotteet As DataSet)

        Dim cnYhteys As SqlConnection = New SqlConnection
        Dim daTuotteet As SqlDataAdapter = New SqlDataAdapter
        Dim cbTuotteet As SqlCommandBuilder = New SqlCommandBuilder(daTuotteet)
        Dim cmmTuotteet As SqlCommand = New SqlCommand _
        ("SELECT * FROM Tuotteet", cnYhteys)
        With cnYhteys
            .ConnectionString = "workstation id=Pomi1;packet size=4096;" _
            & "integrated security=Yes;data source=Pomi1;" _
            & "persist security info=False;initial catalog=sql_tilausjarjestelma"
            .Open()
        End With
        daTuotteet.SelectCommand = cmmTuotteet
        daTuotteet.Update(dstTuotteet, "Tuotteet")
        cnYhteys.Close()

    End Function

End Class
```

LIITE 2: TILAUSJARJESTELMAWS.WSDL

```
<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" targetName-
space="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
- <wsdl:types>
- <s:schema elementFormDefault="qualified" targetName-
space="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS">
<s:import namespace="http://www.w3.org/2001/XMLSchema" />
- <s:element name="dstAsiakkaat">
<s:complexType />
</s:element>
- <s:element name="dstAsiakkaatResponse">
- <s:complexType>
- <s:sequence>
- <s:element minOccurs="0" maxOccurs="1" name="dstAsiakkaatResult">
- <s:complexType>
- <s:sequence>
<s:element ref="s:schema" />
<s:any />
</s:sequence>
</s:complexType>
</s:element>
</s:sequence>
</s:complexType>
</s:element>
- <s:element name="dstAsiakkaatTallenna">
- <s:complexType>
- <s:sequence>
- <s:element minOccurs="0" maxOccurs="1" name="dstAsiakkaat">
- <s:complexType>
- <s:sequence>
<s:element ref="s:schema" />
<s:any />
</s:sequence>
</s:complexType>
</s:element>
</s:sequence>
</s:complexType>
</s:element>
```

<s:element name="dstAsiakkaatTallennaResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dstAsiakkaatTallennaResult" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="dstTuotteet">
<s:complexType />
</s:element>
<s:element name="dstTuotteetResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dstTuotteetResult">
<s:complexType>
<s:sequence>
<s:element ref="s:schema" />
<s:any />
</s:sequence>
</s:complexType>
</s:element>
</s:sequence>
</s:complexType>
</s:element>
<s:element name="dstTuotteetTallenna">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dstTuotteet">
<s:complexType>
<s:sequence>
<s:element ref="s:schema" />
<s:any />
</s:sequence>
</s:complexType>
</s:element>
</s:sequence>
</s:complexType>
</s:element>
<s:element name="dstTuotteetTallennaResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dstTuotteetTallennaResult" />
</s:sequence>
</s:complexType>
</s:element>
</s:schema>
</wsdl:types>
<wsdl:message name="dstAsiakkaatSoapIn">

<wsdl:part name="parameters" element="tns:dstAsiakkaat" />
</wsdl:message>
- <wsdl:message name="dstAsiakkaatSoapOut">
<wsdl:part name="parameters" element="tns:dstAsiakkaatResponse" />
</wsdl:message>
- <wsdl:message name="dstAsiakkaatTallennaSoapIn">
<wsdl:part name="parameters" element="tns:dstAsiakkaatTallenna" />
</wsdl:message>
- <wsdl:message name="dstAsiakkaatTallennaSoapOut">
<wsdl:part name="parameters" element="tns:dstAsiakkaatTallennaResponse" />
</wsdl:message>
- <wsdl:message name="dstTuotteetSoapIn">
<wsdl:part name="parameters" element="tns:dstTuotteet" />
</wsdl:message>
- <wsdl:message name="dstTuotteetSoapOut">
<wsdl:part name="parameters" element="tns:dstTuotteetResponse" />
</wsdl:message>
- <wsdl:message name="dstTuotteetTallennaSoapIn">
<wsdl:part name="parameters" element="tns:dstTuotteetTallenna" />
</wsdl:message>
- <wsdl:message name="dstTuotteetTallennaSoapOut">
<wsdl:part name="parameters" element="tns:dstTuotteetTallennaResponse" />
</wsdl:message>
- <wsdl:portType name="TilausjarjestelmaWSSoap">
- <wsdl:operation name="dstAsiakkaat">
<wsdl:input message="tns:dstAsiakkaatSoapIn" />
<wsdl:output message="tns:dstAsiakkaatSoapOut" />
</wsdl:operation>
- <wsdl:operation name="dstAsiakkaatTallenna">
<wsdl:input message="tns:dstAsiakkaatTallennaSoapIn" />
<wsdl:output message="tns:dstAsiakkaatTallennaSoapOut" />
</wsdl:operation>
- <wsdl:operation name="dstTuotteet">
<wsdl:input message="tns:dstTuotteetSoapIn" />
<wsdl:output message="tns:dstTuotteetSoapOut" />
</wsdl:operation>
- <wsdl:operation name="dstTuotteetTallenna">
<wsdl:input message="tns:dstTuotteetTallennaSoapIn" />
<wsdl:output message="tns:dstTuotteetTallennaSoapOut" />
</wsdl:operation>
</wsdl:portType>
- <wsdl:binding name="TilausjarjestelmaWSSoap" ty-
pe="tns:TilausjarjestelmaWSSoap">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="dstAsiakkaat">
<soap:operation soapAc-
tion="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS/dstAsiakkaat"
style="document" />

<wsdl:input>
<soap:body use="literal" />
</wsdl:input>
<wsdl:output>
<soap:body use="literal" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="dstAsiakkaatTallenna">
<soap:operation soapAction="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS/dstAsiakkaatTallenna" style="document" />
<wsdl:input>
<soap:body use="literal" />
</wsdl:input>
<wsdl:output>
<soap:body use="literal" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="dstTuotteet">
<soap:operation soapAction="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS/dstTuotteet" style="document" />
<wsdl:input>
<soap:body use="literal" />
</wsdl:input>
<wsdl:output>
<soap:body use="literal" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="dstTuotteetTallenna">
<soap:operation soapAction="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS/dstTuotteetTallenna" style="document" />
<wsdl:input>
<soap:body use="literal" />
</wsdl:input>
<wsdl:output>
<soap:body use="literal" />
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="TilausjarjestelmaWS">
<documentation xmlns="http://schemas.xmlsoap.org/wsdl/" />
<wsdl:port name="TilausjarjestelmaWSSoap" binding="tns:TilausjarjestelmaWSSoap">
<soap:address location="http://pomi1/TilausjarjestelmaWSNET/TilausjarjestelmaWS.asmx" />
</wsdl:port>

```
</wsdl:service>
</wsdl:definitions>
```

LIITE 3: PROXY-LUOKKA

```
'------------------------------------------------------------------------------
' <auto-generated>
'     This code was generated by a tool.
'     Runtime Version:2.0.50727.42
'
'     Changes to this file may cause incorrect behavior and will be lost if
'     the code is regenerated.
' </auto-generated>
'------------------------------------------------------------------------------

Option Strict Off
Option Explicit On

Imports System
Imports System.ComponentModel
Imports System.Data
Imports System.Diagnostics
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Xml.Serialization

'
'This source code was auto-generated by Microsoft.VSDesigner, Version 2.0.50727.42.
'
Namespace localhost

    '"<remarks/>
    <System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"2.0.50727.42"),  _
     System.Diagnostics.DebuggerStepThroughAttribute(),  _
     System.ComponentModel.DesignerCategoryAttribute("code"),  _

System.Web.Services.WebServiceBindingAttribute(Name:="TilausjarjestelmaWSSoap
", [Namespace]:="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaWS")>
_
    Partial Public Class TilausjarjestelmaWS
        Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

        Private dstAsiakkaatOperationCompleted As
System.Threading.SendOrPostCallback

        Private dstAsiakkaatTallennaOperationCompleted As
System.Threading.SendOrPostCallback
```

```
    Private dstTuotteetOperationCompleted As
System.Threading.SendOrPostCallback

    Private dstTuotteetTallennaOperationCompleted As
System.Threading.SendOrPostCallback

    Private useDefaultCredentialsSetExplicitly As Boolean

    '''<remarks/>
    Public Sub New()
        MyBase.New
        Me.Url = "http://pomi1/TilausjarjestelmaWSNET/TilausjarjestelmaWS.asmx"
        If (Me.IsLocalFileSystemWebService(Me.Url) = true) Then
            Me.UseDefaultCredentials = true
            Me.useDefaultCredentialsSetExplicitly = false
        Else
            Me.useDefaultCredentialsSetExplicitly = true
        End If
    End Sub

    Public Shadows Property Url() As String
        Get
            Return MyBase.Url
        End Get
        Set
            If (((Me.IsLocalFileSystemWebService(MyBase.Url) = true)  _
                        AndAlso (Me.useDefaultCredentialsSetExplicitly = false))  _
                        AndAlso (Me.IsLocalFileSystemWebService(value) = false)) Then
                MyBase.UseDefaultCredentials = false
            End If
            MyBase.Url = value
        End Set
    End Property

    Public Shadows Property UseDefaultCredentials() As Boolean
        Get
            Return MyBase.UseDefaultCredentials
        End Get
        Set
            MyBase.UseDefaultCredentials = value
            Me.useDefaultCredentialsSetExplicitly = true
        End Set
    End Property

    '''<remarks/>
    Public Event dstAsiakkaatCompleted As dstAsiakkaatCompletedEventHandler

    '''<remarks/>
```

```
        Public Event dstAsiakkaatTallennaCompleted As
dstAsiakkaatTallennaCompletedEventHandler

        '''<remarks/>
        Public Event dstTuotteetCompleted As dstTuotteetCompletedEventHandler

        '''<remarks/>
        Public Event dstTuotteetTallennaCompleted As
dstTuotteetTallennaCompletedEventHandler

        '''<remarks/>

<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/T
ilausjarjestelmaWSNET/TilausjarjestelmaWS/dstAsiakkaat",
RequestNamespace:="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaW
S",
ResponseNamespace:="http://tempuri.org/TilausjarjestelmaWSNET/Tilausjarjestelma
WS", Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>  _
        Public Function dstAsiakkaat() As System.Data.DataSet
            Dim results() As Object = Me.Invoke("dstAsiakkaat", New Object(-1) {})
            Return CType(results(0),System.Data.DataSet)
        End Function

        '''<remarks/>
        Public Function BegindstAsiakkaat(ByVal callback As System.AsyncCallback,
ByVal asyncState As Object) As System.IAsyncResult
            Return Me.BeginInvoke("dstAsiakkaat", New Object(-1) {}, callback,
asyncState)
        End Function

        '''<remarks/>
        Public Function EnddstAsiakkaat(ByVal asyncResult As System.IAsyncResult) As
System.Data.DataSet
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),System.Data.DataSet)
        End Function

        '''<remarks/>
        Public Overloads Sub dstAsiakkaatAsync()
            Me.dstAsiakkaatAsync(Nothing)
        End Sub

        '''<remarks/>
        Public Overloads Sub dstAsiakkaatAsync(ByVal userState As Object)
            If (Me.dstAsiakkaatOperationCompleted Is Nothing) Then
                Me.dstAsiakkaatOperationCompleted = AddressOf
Me.OndstAsiakkaatOperationCompleted
```

```
        End If
        Me.InvokeAsync("dstAsiakkaat", New Object(-1) {},
Me.dstAsiakkaatOperationCompleted, userState)
    End Sub

    Private Sub OndstAsiakkaatOperationCompleted(ByVal arg As Object)
        If (Not (Me.dstAsiakkaatCompletedEvent) Is Nothing) Then
            Dim invokeArgs As
System.Web.Services.Protocols.InvokeCompletedEventArgs =
CType(arg,System.Web.Services.Protocols.InvokeCompletedEventArgs)
            RaiseEvent dstAsiakkaatCompleted(Me, New
dstAsiakkaatCompletedEventArgs(invokeArgs.Results, invokeArgs.Error,
invokeArgs.Cancelled, invokeArgs.UserState))
        End If
    End Sub

    '''<remarks/>

<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/T
ilausjarjestelmaWSNET/TilausjarjestelmaWS/dstAsiakkaatTallenn"& _
        "a",
RequestNamespace:="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaW
S",
ResponseNamespace:="http://tempuri.org/TilausjarjestelmaWSNET/Tilausjarjestelma
WS", Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>  _
    Public Function
dstAsiakkaatTallenna(<System.Xml.Serialization.XmlElementAttribute("dstAsiakkaat"
)> ByVal dstAsiakkaat1 As System.Data.DataSet) As Object
        Dim results() As Object = Me.Invoke("dstAsiakkaatTallenna", New Object()
{dstAsiakkaat1})
        Return CType(results(0),Object)
    End Function

    '''<remarks/>
    Public Function BegindstAsiakkaatTallenna(ByVal dstAsiakkaat1 As
System.Data.DataSet, ByVal callback As System.AsyncCallback, ByVal asyncState As
Object) As System.IAsyncResult
        Return Me.BeginInvoke("dstAsiakkaatTallenna", New Object()
{dstAsiakkaat1}, callback, asyncState)
    End Function

    '''<remarks/>
    Public Function EnddstAsiakkaatTallenna(ByVal asyncResult As
System.IAsyncResult) As Object
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),Object)
    End Function
```

```
        '''<remarks/>
        Public Overloads Sub dstAsiakkaatTallennaAsync(ByVal dstAsiakkaat1 As
System.Data.DataSet)
            Me.dstAsiakkaatTallennaAsync(dstAsiakkaat1, Nothing)
        End Sub

        '''<remarks/>
        Public Overloads Sub dstAsiakkaatTallennaAsync(ByVal dstAsiakkaat1 As
System.Data.DataSet, ByVal userState As Object)
            If (Me.dstAsiakkaatTallennaOperationCompleted Is Nothing) Then
                Me.dstAsiakkaatTallennaOperationCompleted = AddressOf
Me.OndstAsiakkaatTallennaOperationCompleted
            End If
            Me.InvokeAsync("dstAsiakkaatTallenna", New Object() {dstAsiakkaat1},
Me.dstAsiakkaatTallennaOperationCompleted, userState)
        End Sub

        Private Sub OndstAsiakkaatTallennaOperationCompleted(ByVal arg As Object)
            If (Not (Me.dstAsiakkaatTallennaCompletedEvent) Is Nothing) Then
                Dim invokeArgs As
System.Web.Services.Protocols.InvokeCompletedEventArgs =
CType(arg,System.Web.Services.Protocols.InvokeCompletedEventArgs)
                RaiseEvent dstAsiakkaatTallennaCompleted(Me, New
dstAsiakkaatTallennaCompletedEventArgs(invokeArgs.Results, invokeArgs.Error,
invokeArgs.Cancelled, invokeArgs.UserState))
            End If
        End Sub

        '''<remarks/>

<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/T
ilausjarjestelmaWSNET/TilausjarjestelmaWS/dstTuotteet",
RequestNamespace:="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaW
S",
ResponseNamespace:="http://tempuri.org/TilausjarjestelmaWSNET/Tilausjarjestelma
WS", Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>  _
        Public Function dstTuotteet() As System.Data.DataSet
            Dim results() As Object = Me.Invoke("dstTuotteet", New Object(-1) {})
            Return CType(results(0),System.Data.DataSet)
        End Function

        '''<remarks/>
        Public Function BegindstTuotteet(ByVal callback As System.AsyncCallback,
ByVal asyncState As Object) As System.IAsyncResult
            Return Me.BeginInvoke("dstTuotteet", New Object(-1) {}, callback, asyncState)
        End Function
```

```
        '''<remarks/>
        Public Function EnddstTuotteet(ByVal asyncResult As System.IAsyncResult) As
System.Data.DataSet
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),System.Data.DataSet)
        End Function

        '''<remarks/>
        Public Overloads Sub dstTuotteetAsync()
            Me.dstTuotteetAsync(Nothing)
        End Sub

        '''<remarks/>
        Public Overloads Sub dstTuotteetAsync(ByVal userState As Object)
            If (Me.dstTuotteetOperationCompleted Is Nothing) Then
                Me.dstTuotteetOperationCompleted = AddressOf
Me.OndstTuotteetOperationCompleted
            End If
            Me.InvokeAsync("dstTuotteet", New Object(-1) {},
Me.dstTuotteetOperationCompleted, userState)
        End Sub

        Private Sub OndstTuotteetOperationCompleted(ByVal arg As Object)
            If (Not (Me.dstTuotteetCompletedEvent) Is Nothing) Then
                Dim invokeArgs As
System.Web.Services.Protocols.InvokeCompletedEventArgs =
CType(arg,System.Web.Services.Protocols.InvokeCompletedEventArgs)
                RaiseEvent dstTuotteetCompleted(Me, New
dstTuotteetCompletedEventArgs(invokeArgs.Results, invokeArgs.Error,
invokeArgs.Cancelled, invokeArgs.UserState))
            End If
        End Sub

        '''<remarks/>

<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/T
ilausjarjestelmaWSNET/TilausjarjestelmaWS/dstTuotteetTallenna"& _
            "",
RequestNamespace:="http://tempuri.org/TilausjarjestelmaWSNET/TilausjarjestelmaW
S",
ResponseNamespace:="http://tempuri.org/TilausjarjestelmaWSNET/Tilausjarjestelma
WS", Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>  _
        Public Function
dstTuotteetTallenna(<System.Xml.Serialization.XmlElementAttribute("dstTuotteet")>
ByVal dstTuotteet1 As System.Data.DataSet) As Object
```

```
        Dim results() As Object = Me.Invoke("dstTuotteetTallenna", New Object()
{dstTuotteet1})
        Return CType(results(0),Object)
    End Function

    '"<remarks/>
    Public Function BegindstTuotteetTallenna(ByVal dstTuotteet1 As
System.Data.DataSet, ByVal callback As System.AsyncCallback, ByVal asyncState As
Object) As System.IAsyncResult
        Return Me.BeginInvoke("dstTuotteetTallenna", New Object() {dstTuotteet1},
callback, asyncState)
    End Function

    '"<remarks/>
    Public Function EnddstTuotteetTallenna(ByVal asyncResult As
System.IAsyncResult) As Object
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),Object)
    End Function

    '"<remarks/>
    Public Overloads Sub dstTuotteetTallennaAsync(ByVal dstTuotteet1 As
System.Data.DataSet)
        Me.dstTuotteetTallennaAsync(dstTuotteet1, Nothing)
    End Sub

    '"<remarks/>
    Public Overloads Sub dstTuotteetTallennaAsync(ByVal dstTuotteet1 As
System.Data.DataSet, ByVal userState As Object)
        If (Me.dstTuotteetTallennaOperationCompleted Is Nothing) Then
            Me.dstTuotteetTallennaOperationCompleted = AddressOf
Me.OndstTuotteetTallennaOperationCompleted
        End If
        Me.InvokeAsync("dstTuotteetTallenna", New Object() {dstTuotteet1},
Me.dstTuotteetTallennaOperationCompleted, userState)
    End Sub

    Private Sub OndstTuotteetTallennaOperationCompleted(ByVal arg As Object)
        If (Not (Me.dstTuotteetTallennaCompletedEvent) Is Nothing) Then
            Dim invokeArgs As
System.Web.Services.Protocols.InvokeCompletedEventArgs =
CType(arg,System.Web.Services.Protocols.InvokeCompletedEventArgs)
            RaiseEvent dstTuotteetTallennaCompleted(Me, New
dstTuotteetTallennaCompletedEventArgs(invokeArgs.Results, invokeArgs.Error,
invokeArgs.Cancelled, invokeArgs.UserState))
        End If
    End Sub
```

```
        '''<remarks/>
        Public Shadows Sub CancelAsync(ByVal userState As Object)
            MyBase.CancelAsync(userState)
        End Sub

        Private Function IsLocalFileSystemWebService(ByVal url As String) As Boolean
            If ((url Is Nothing)  _
                        OrElse (url Is String.Empty)) Then
                Return false
            End If
            Dim wsUri As System.Uri = New System.Uri(url)
            If ((wsUri.Port >= 1024)  _
                        AndAlso (String.Compare(wsUri.Host, "localHost",
System.StringComparison.OrdinalIgnoreCase) = 0)) Then
                Return true
            End If
            Return false
        End Function
    End Class

    '''<remarks/>
    <System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"2.0.50727.42")>  _
    Public Delegate Sub dstAsiakkaatCompletedEventHandler(ByVal sender As Object,
ByVal e As dstAsiakkaatCompletedEventArgs)

    '''<remarks/>
    <System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"2.0.50727.42"),  _
     System.Diagnostics.DebuggerStepThroughAttribute(),  _
     System.ComponentModel.DesignerCategoryAttribute("code")>  _
    Partial Public Class dstAsiakkaatCompletedEventArgs
        Inherits System.ComponentModel.AsyncCompletedEventArgs

        Private results() As Object

        Friend Sub New(ByVal results() As Object, ByVal exception As
System.Exception, ByVal cancelled As Boolean, ByVal userState As Object)
            MyBase.New(exception, cancelled, userState)
            Me.results = results
        End Sub

        '''<remarks/>
        Public ReadOnly Property Result() As System.Data.DataSet
            Get
                Me.RaiseExceptionIfNecessary
                Return CType(Me.results(0),System.Data.DataSet)
            End Get
```

```
        End Property
    End Class

    '''<remarks/>
    <System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"2.0.50727.42")>  _
    Public Delegate Sub dstAsiakkaatTallennaCompletedEventHandler(ByVal sender As
Object, ByVal e As dstAsiakkaatTallennaCompletedEventArgs)

    '''<remarks/>
    <System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"2.0.50727.42"),  _
     System.Diagnostics.DebuggerStepThroughAttribute(),  _
     System.ComponentModel.DesignerCategoryAttribute("code")>  _
    Partial Public Class dstAsiakkaatTallennaCompletedEventArgs
        Inherits System.ComponentModel.AsyncCompletedEventArgs

        Private results() As Object

        Friend Sub New(ByVal results() As Object, ByVal exception As
System.Exception, ByVal cancelled As Boolean, ByVal userState As Object)
            MyBase.New(exception, cancelled, userState)
            Me.results = results
        End Sub

        '''<remarks/>
        Public ReadOnly Property Result() As Object
            Get
                Me.RaiseExceptionIfNecessary
                Return CType(Me.results(0),Object)
            End Get
        End Property
    End Class

    '''<remarks/>
    <System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"2.0.50727.42")>  _
    Public Delegate Sub dstTuotteetCompletedEventHandler(ByVal sender As Object,
ByVal e As dstTuotteetCompletedEventArgs)

    '''<remarks/>
    <System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"2.0.50727.42"),  _
     System.Diagnostics.DebuggerStepThroughAttribute(),  _
     System.ComponentModel.DesignerCategoryAttribute("code")>  _
    Partial Public Class dstTuotteetCompletedEventArgs
        Inherits System.ComponentModel.AsyncCompletedEventArgs
```

```
        Private results() As Object

        Friend Sub New(ByVal results() As Object, ByVal exception As
System.Exception, ByVal cancelled As Boolean, ByVal userState As Object)
            MyBase.New(exception, cancelled, userState)
            Me.results = results
        End Sub

        '''<remarks/>
        Public ReadOnly Property Result() As System.Data.DataSet
            Get
                Me.RaiseExceptionIfNecessary
                Return CType(Me.results(0),System.Data.DataSet)
            End Get
        End Property
    End Class

    '''<remarks/>
    <System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"2.0.50727.42")>  _
    Public Delegate Sub dstTuotteetTallennaCompletedEventHandler(ByVal sender As
Object, ByVal e As dstTuotteetTallennaCompletedEventArgs)

    '''<remarks/>
    <System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"2.0.50727.42"),  _
     System.Diagnostics.DebuggerStepThroughAttribute(),  _
     System.ComponentModel.DesignerCategoryAttribute("code")>  _
    Partial Public Class dstTuotteetTallennaCompletedEventArgs
        Inherits System.ComponentModel.AsyncCompletedEventArgs

        Private results() As Object

        Friend Sub New(ByVal results() As Object, ByVal exception As
System.Exception, ByVal cancelled As Boolean, ByVal userState As Object)
            MyBase.New(exception, cancelled, userState)
            Me.results = results
        End Sub

        '''<remarks/>
        Public ReadOnly Property Result() As Object
            Get
                Me.RaiseExceptionIfNecessary
                Return CType(Me.results(0),Object)
            End Get
        End Property
    End Class
End Namespace
```

LIITE 4: FRMASIAKASREKISTERI.VB

```
Option Strict Off
Option Explicit On

' Kun tuodaan nimiavaruudet, nimet lyhenee
Imports System.Data, System.Data.SqlClient

Friend Class frmAsiakasrekisteri

    Inherits System.Windows.Forms.Form
    Dim dstAsiakkaat As New DataSet ' Tyypittämätön DataSet

    Private Sub frmAsiakasrekisteri_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load ' ok
        Taytaasiakaslista()
    End Sub

    Private Sub Taytaasiakaslista() ' ok
        Dim wsdstAsiakkaat As New localhost.TilausjarjestelmaWS
        dstAsiakkaat = wsdstAsiakkaat.dstAsiakkaat
        lbAsiakkaat.DataSource = dstAsiakkaat.Tables("Asiakas")
        lbAsiakkaat.DisplayMember = "Yritys"
    End Sub

    Private Sub lbAsiakkaat_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles lbAsiakkaat.Click ' ok
        Dim i As Integer
        i = lbAsiakkaat.SelectedIndex ' listasta valitun indeksi, trimmataan merkkijonot
        txtAsiakastunnus.Text =
RTrim(CType(dstAsiakkaat.Tables("Asiakas").Rows(i).Item("AsiakasID"), String))
        txtYritys.Text =
RTrim(CType(dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Yritys"), String))
        txtKatuosoite.Text =
RTrim(CType(dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Katuosoite"), String))
        txtPostinumero.Text =
RTrim(CType(dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Postinumero"), String))
        txtPostitoimipaikka.Text =
RTrim(CType(dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Postitoimipaikka"),
String))
        txtPuhelin.Text =
RTrim(CType(dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Puhelin"), String))
        txtFaksi.Text =
RTrim(CType(dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Faksi"), String))
    End Sub
```

```
    Private Sub Tyhjennakentat() ' ok
        ' Tyhjennetään kentät
        txtAsiakastunnus.Text = ""
        txtYritys.Text = ""
        txtKatuosoite.Text = ""
        txtPostinumero.Text = ""
        txtPostitoimipaikka.Text = ""
        txtPuhelin.Text = ""
        txtFaksi.Text = ""
    End Sub

    Private Sub Tarkastakentat(ByRef s As String) ' ok
        ' Tarkastetaan että kentissä on tietoja
        If txtYritys.Text = "" Then s = "!"
        If txtKatuosoite.Text = "" Then s = "!"
        If txtPostinumero.Text = "" Then s = "!"
        If txtPostitoimipaikka.Text = "" Then s = "!"
        If txtPuhelin.Text = "" Then s = "!"
        If txtFaksi.Text = "" Then s = "!"
        If s = "!" Then MsgBox("Tarkasta tiedot!")
    End Sub

    Private Sub cmdPaivita_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cmdPaivita.Click ' ok
        Dim AsyncResult As IAsyncResult
        Dim wsdstAsiakkaatTallenna As New localhost.TilausjarjestelmaWS
        Dim i As Integer
        Dim s As String = ""
        Tarkastakentat(s)
        If s = "!" Then Exit Sub
        i = lbAsiakkaat.SelectedIndex ' Listasta valitun indeksi
        dstAsiakkaat.Tables("Asiakas").Rows(i).Item("AsiakasID") =
txtAsiakastunnus.Text
        dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Yritys") = txtYritys.Text
        dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Katuosoite") = txtKatuosoite.Text
        dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Postinumero") =
txtPostinumero.Text
        dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Postitoimipaikka") =
txtPostitoimipaikka.Text
        dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Puhelin") = txtPuhelin.Text
        dstAsiakkaat.Tables("Asiakas").Rows(i).Item("Faksi") = txtFaksi.Text
        AsyncResult = wsdstAsiakkaatTallenna.BegindstAsiakkaatTallenna(dstAsiakkaat,
Nothing, Nothing)
        wsdstAsiakkaatTallenna.EnddstAsiakkaatTallenna(AsyncResult)
```

```
        If AsyncResult.IsCompleted = True Then
            MsgBox("Tiedot talletettu.", MsgBoxStyle.Information, Me.Text)
            Tyhjennakentat()
            Taytaasiakaslista()
        Else
            MsgBox("Talletetus epäonnistui.", MsgBoxStyle.Critical, Me.Text)
        End If
    End Sub

    Private Sub cmdUusi_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cmdUusi.Click ' ok
        If cmdUusi.Text = "Uusi" Then
            cmdUusi.Text = "Tallenna"
            Tyhjennakentat()
            MsgBox("Nyt voit kirjoittaa uuden asiakkaan tiedot. " _
            & "Paina lopuksi Tallenna-painiketta.", MsgBoxStyle.OKOnly, _
            "Uuden asiakkaan kirjaaminen")
            txtYritys.Focus()
            lbAsiakkaat.Enabled = False
            cmdPaivita.Enabled = False
            cmdPoista.Enabled = False
            Exit Sub
        End If

        Dim AsyncResult As IAsyncResult
        Dim wsdstAsiakkaatTallenna As New localhost.TilausjarjestelmaWS
        Dim rowAsiakas As DataRow = dstAsiakkaat.Tables("Asiakas").NewRow
        Dim s As String
        Tarkastakentat(s)
        If s = "!" Then Exit Sub
        rowAsiakas("AsiakasID") = txtAsiakastunnus.Text
        rowAsiakas("Yritys") = txtYritys.Text
        rowAsiakas("Katuosoite") = txtKatuosoite.Text
        rowAsiakas("Postinumero") = txtPostinumero.Text
        rowAsiakas("Postitoimipaikka") = txtPostitoimipaikka.Text
        rowAsiakas("Puhelin") = txtPuhelin.Text
        rowAsiakas("Faksi") = txtFaksi.Text
        dstAsiakkaat.Tables("Asiakas").Rows.Add(rowAsiakas)
        cmdUusi.Text = "Uusi"
        lbAsiakkaat.Enabled = True
        cmdPaivita.Enabled = True
        cmdPoista.Enabled = True
        AsyncResult = wsdstAsiakkaatTallenna.BegindstAsiakkaatTallenna(dstAsiakkaat,
Nothing, Nothing)
        wsdstAsiakkaatTallenna.EnddstAsiakkaatTallenna(AsyncResult)
```

```
        If AsyncResult.IsCompleted = True Then
            MsgBox("Tiedot talletettu.", MsgBoxStyle.Information, Me.Text)
            Tyhjennakentat()
            Taytaasiakaslista()
        Else
            MsgBox("Talletetus epäonnistui.", MsgBoxStyle.Critical, Me.Text)
        End If
    End Sub

    Private Sub cmdPoista_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cmdPoista.Click ' ok
        Dim AsyncResult As IAsyncResult
        Dim wsdstAsiakkaatTallenna As New localhost.TilausjarjestelmaWS
        If lbAsiakkaat.SelectedIndex >= 0 Then
            dstAsiakkaat.Tables("Asiakas").Rows(lbAsiakkaat.SelectedIndex).Delete()
        Else
            MsgBox("Valitse listasta.", MsgBoxStyle.Information, Me.Text)
        End If
        AsyncResult = wsdstAsiakkaatTallenna.BegindstAsiakkaatTallenna(dstAsiakkaat,
Nothing, Nothing)
        wsdstAsiakkaatTallenna.EnddstAsiakkaatTallenna(AsyncResult)
        If AsyncResult.IsCompleted = True Then
            MsgBox("Tiedot poistettu.", MsgBoxStyle.Information, Me.Text)
            Tyhjennakentat()
            Taytaasiakaslista()
        Else
            MsgBox("Poistaminen epäonnistui.", MsgBoxStyle.Critical, Me.Text)
        End If
    End Sub

    Private Sub cmdSulje_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cmdSulje.Click ' ok
        Me.Close()
    End Sub

End Class
```