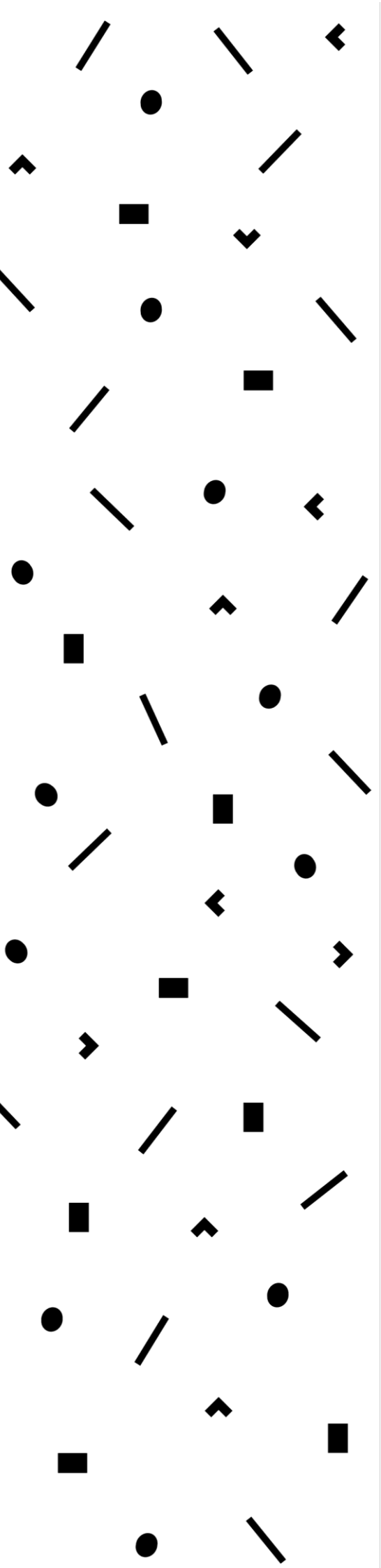


LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

ASIAKASYMPÄRISTÖ-VALVONNAN AUTOMATISOINTI

Asiakasohjelmisto ja tietokanta

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Syksy 2015
Tomi Saari

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

SAARI, TOMI:

Asiakasympäristövalvonnan automatisointi
Asiakasohjelmisto ja tietokanta

Ohjelmistotekniikan opinnäytetyö, 64 sivua

Syksy 2015

TIIVISTELMÄ

Tässä opinnäytetyössä tutkittiin päivittäisvalvontarutiinien automatisointia. Päivittäisvalvontarutiinit ovat asiakkaan ympäristöön sovitettuja toimenpiteitä, joiden avulla varmistetaan tietovaraston latauksiin liittyvien toimintojen onnistunutta suoritusta. Nämä toimenpiteet vievät aikaa päivittäisvalvontarutiineja suorittavilta henkilöiltä. Prosessin automatisointi vapauttaisi valvonnan suorittavat henkilöt muihin tehtäviin ja auttaa reagoimaan mahdollisiin ongelmatilanteisiin nopeammin.

Työ toteutettiin CGI Suomi Oy:n Lahden-toimipisteelle. CGI, eli Consultants to Government and Industry, on yksi maailman suurimmista IT-palvelualan yrityksistä, jolla on maailmanlaajuisesti 68 000 asiantuntijaa 40:ssä eri maassa erilaisissa työtehtävissä. Suomessa työntekijöitä on noin 3000. CGI Oy:n tarjoamia palveluita ovat esimerkiksi IT-infrastruktuuripalvelut, konsultointi, liiketoimintaprosessien hallinta, liiketoimintaratkaisut, sovellushallinta ja ulkoistuspalvelut.

Opinnäytetyössä suunniteltiin ja toteutettiin asiakkaan ympäristöön tuleva asiakassovellus, joka kerää päivittäisvalvontarutiinien suoritukseen tarvittavia tietoja. Asiakassovelluksen tarkoituksena on vähentää manuaalista työtä, ohjelmiston suorittamien rutiinien ansiosta. Kerätyt tiedot lähetetään eteenpäin tiedon analysointia ja historiointia varten. Työosuus toteutettiin käyttäen Microsoftin teknologioita.

Asiasanat: asiakasympäristövalvonta, SQL Server, Microsoft, C#, Windows Communication Foundation, Business Intelligence, .NET-viitekehys

Lahti University of Applied Sciences
Degree Programme in Information Technology

SAARI, TOMI:

Automation of Customer Environment Monitoring
Client software and database

Bachelor's Thesis in software engineering, 64 pages

Autumn 2015

ABSTRACT

The objective of this thesis was to study and develop automatic data warehouse monitoring software for a customer's environment. Data warehouse monitoring is a set of routines that are done manually on the customer's environment to ensure flawless data warehouse operation. These routines take time from the person who executes daily monitoring. When this process is automated, it will release the person from daily monitoring to other tasks and decrease reaction time in case of faulty operation.

The thesis was made for CGI Finland Ltd.'s Lahti office. CGI, Consultants to Government and Industry, is one of the world's largest IT service sector company which has 68 000 experts worldwide in 40 different countries in different work assignments. In Finland CGI has about 3000 workers. Services that CGI Ltd offers are for example IT infrastructure services, consultation, business process management, business solutions, software management and outsourcing services.

The thesis includes planning and coding of customer client software that collects information for the daily monitoring. The goal of client software was to reduce manual labor, due to routines executed by the software. Collected data will be sent for data analysis and history purposes. The work was carried out with Microsoft technologies.

Key words: customer environment monitoring, SQL Server, Microsoft, C#, Windows Communication Foundation, Business Intelligence, .NET Framework

SISÄLLYS

1 JOHDANTO

Nykypäivänä suuri osa yrityksistä perustaa päätöksensä liiketoiminnasta kerättyyn tietoon. Tieto voi olla esimerkiksi asiakaskäyttäytymistä. Nämä tiedot voivat olla toimipistekohtaista myyntitietoa, tilaustietoa, myyntitietoa tai lähes mitä tahansa tietoa, jota voidaan hyödyntää päätöksenteossa. Tätä kokonaisuutta kutsutaan Business Intelligenceksi. Business Intelligence tarkoittaa karkeasti tiedon hyödyntämistä päätöksenteossa. Tiedonlähteinä toimivat usein tekstitiedostot, Excel-tiedostot tai erillinen lähdetietokanta. Kun tietoa siirretään lähteestä kohdetietokantaan, on mahdollista, että tämän toimenpiteen aikana tapahtuu virhe, ja vaikka järjestelmä olisikin suunniteltu hyvin, kaikkiin tilanteisiin ei voida varauta ennakkoon.

Asiakkaina toimivien yritysten päätöksenteko perustuu Business Intelligence -raportteihin, minkä takia on tärkeää, että eräajot ovat menneet läpi. Tämän vuoksi asiakkaille tarjotaan jatkuvaan palveluun kuuluva aamuvalvonta. Aamuvalvonta on joukko asiakkaalle suoritettavia ympäristökohtaisia tarkistuksia, joiden tarkoituksena on saada asiakkaalle mahdollisimman nopeasti tieto siitä, että eräajot ovat menneet ongelmitta läpi. Näin asiakas saa varmuuden siitä, että data, joka hänellä on käytössä, on ajan tasalla, ja sitä voidaan käyttää päätöksenteossa ja raportoinnissa.

Työssä toteutetaan automaattinen aamuvalvontasovellus. Tällä säästetään useita henkilötyötunteja viikossa. Työ toteutetaan hyödyntämällä Microsoftin teknologioita, joihin perehdytään tarkemmin työn teoriaosuudessa. Ohjelmisto on kolmeosainen: CGI:n raportointinäkymä, tietokanta ja valvonnan suorittava aliohjelmisto. Tämä työ keskittyy tietokantarakenteeseen ja asiakasohjelmistoon.

2 BUSINESS INTELLIGENCE (BI)

Liiketoiminnassa syntyy tietoa, jota voidaan käyttää yritystä koskevia päätöksiä tehtäessä. Jos tätä tietoa osataan hyödyntää, sen perusteella voidaan tehdä yrityksen kannalta erittäin hyviä ratkaisuja. Tämä tieto voi sisältää esimerkiksi myynnit maittain, kunnittain tai kaupungeittain aina yksittäisen myyjän tasolle. Näillä tiedoilla voidaan esimerkiksi sulkea pitkään huonosti tuottanut toimipiste tai tehdä uudistuksia, joilla toimipisteen tulos saadaan parannettua.

Tämän opinnäytetyön kannalta Business Intelligence on tärkeässä osassa, koska asiakkaiden järjestelmät, jotka kuuluvat valvonnan piiriin, sisältävät tietoa, jota asiakas käyttää raportoinnissa. Kun tietoa viedään tietovarastoon, siirrossa voi tapahtua erilaisia virheitä, joiden huomaaminen varhaisessa vaiheessa on tärkeää. Virhe voi olla esimerkiksi saman rivin syöttäminen uudelleen tietokantaan. Jos eräajoissa on tapahtunut virhe eikä sitä huomata ajoissa, asiakas voi tehdä kriittisiä päätöksiä vanhojen tai jopa väärien tietojen perusteella.

2.1 Business Intelligencen ymmärtäminen

Gartnerin määritelmän mukaan Business Intelligencen (BI) määritelmä on seuraava: Business Intelligence on kaiken kattava termi, johon sisältyy sovellukset, infrastruktuuri, työkalut ja parhaat käytännöt, jotka mahdollistavat tiedon saannin ja analysoinnin ratkaisujen tehostamiseen, parantamisen ja optimointiin. Kuten määritelmästä huomataan, BI -järjestelmän päätavoitteena on auttaa päätöksentekijöitä tekemään oikeat päätökset BI -järjestelmän tuottaman data-analytiikan perusteella. (Rad 2014, 7 - 8.)

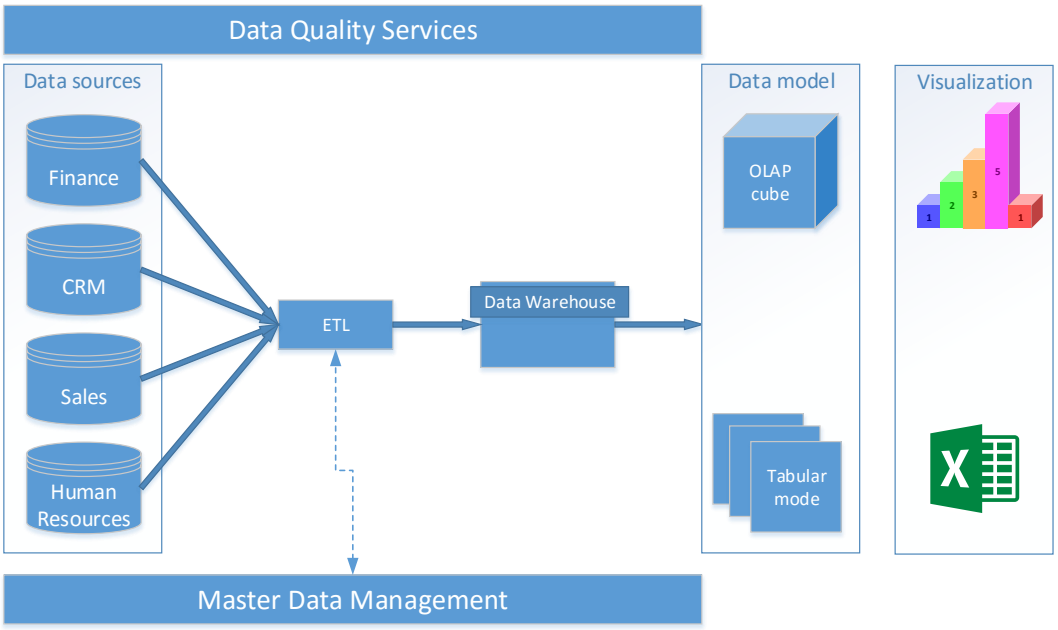
Yritykset käyttävät monenlaisia toiminnallisia järjestelmiä jokapäiväisten toimintojen yksinkertaistamiseen, standardoimiseen ja toimintojen automatisointiin. Jokaisella järjestelmällä voi olla oma tietokantansa, joista yksi voi toimia vaikka SQL Server- ja toinen Oraclen teknologialla. Vanhemmat järjestelmät voivat toimia vanhempien tietokantojen kanssa tai jopa tiedostojen varassa. Jotkin järjestelmät toimivat web-palveluiden kautta ja XML-

tiedostoilla. Toiminnalliset järjestelmät ovat hyödyllisiä jokapäiväisten liiketoimintaan liittyvien ratkaisujen helpottamiseksi, kuten henkilöstöosaston palkkauspäätöksissä, vähittäiskaupan myynnin seurannassa ja taloudellisten transaktioiden hallinnassa. (Rad 2014, 8.)

Kun toiminnalliset järjestelmät lisääntyvät, tuo se myös mukanaan uuden vaatimuksen: eri järjestelmien sulauttaminen yhteen. Yritysten omistajat ja päätösten tekijät tarvitsevat yhdistetyn datan lisäksi myös yhdistetyn datan analysointia. Esimerkiksi on yleistä, että yrityksen päätöksentekijät vertaavat palkkausmääriä yrityksen palvelun tasoon ja käyttäjien tyytyväisyyttä palvelun tasoon nähden. Nämä vaatimukset tarvitsevat asiakkuudenhallinta- ja henkilöstöhallintajärjestelmien hyödyntämistä yhdessä. Vaatimuksena voi olla myös myynti- ja varastojärjestelmien tietoja, jos halutaan ottaa huomioon myynnit ja varastot. Isojen hypermarkettien omistajan tai päätöksentekijän on tärkeää ymmärtää, mitkä tuotteet menevät parhaiten kaupaksi yksittäisessä myymälässä. Tällainen tieto auttaa vastaamaan kysyntään ja voi avustaa päättämään, avataanko alueelle uusi toimipiste. Business Intelligencen päävaatimuksena on eri järjestelmien yhdistäminen, jotta pystytään rakentamaan koottuja raportteja päätöksentekijöiden päätösten tueksi. (Rad 2014, 8.)

2.2 BI järjestelmän arkkitehtuuri ja komponentit

BI-järjestelmä voidaan luoda manuaalisesti, mutta on myös työkaluja, joiden avulla voidaan implementoida järjestelmän komponentteja. Kuvio 1 sisältää BI-järjestelmän arkkitehtuurin ja pääkomponentit. (Rad 2014, 9.)



KUVIO 1. BI-järjestelmän arkkitehtuuri ja pääkomponentit (Rad 2014, 9)

BI:n komponentit ja arkkitehtuuri vaihtelevat muun muassa käytetyistä työkaluista ja ympäristöistä johtuen. Kuviossa 1 näkyvät komponentit ovat yleisiä suuressa osassa BI-järjestelmiä. (Rad 2014, 9.)

Kuviota 1 tarkasteltaessa nähdään useita eri tietolähteitä, joita voivat olla esimerkiksi myynti- ja HR-tietokanta. Näistä tietolähteistä ladataan tiedot tietovarastoon hyväksikäyttäen ETL-ketjua (Extract Transform Load). Dataan voidaan tehdä laskentaa esimerkiksi OLAP-kuution avulla. Nämä tiedot voidaan esittää käyttäjälle esimerkiksi graafeina tai Excel-tiedostona.

2.2.1 Tietovarasto (Data Warehouse)

Tietovarasto on BI-järjestelmän ydin. Tietovarasto on tietokanta, joka on rakennettu datan analysointia ja raportointia varten. Tämä tarkoitus muuttaa tämän tietokannan rakennetta. Toiminnalliset kannat rakennetaan normalisaatiostandardien mukaisesti, jotka ovat tehokkaita transaktionaalisiin järjestelmiin, esimerkiksi vähentämään tarpeetonta dataa. Normalisoinnilla tarkoitetaan sitä, että samaa tietoa ei toisteta useaan kertaan tietokannassa. Kolmannen normaalimuodon mukaan suunniteltu myyntijärjestelmä

sisältää monta taulua, jotka ovat relaatiossa toisiinsa. Esimerkiksi myyntiraportti voi käyttää yli kymmentä yhdistettyä ehtoa, mikä hidastaa kyselyn ja raportin vasteaikaa. Tietovarasto käyttää uutta suunnittelumallia, mikä vähentää reaktioaikaa ja nostaa analytiikka- ja raporttikyselyiden tehokkuutta. (Rad 2014, 9.)

2.2.2 Extract Transform Load

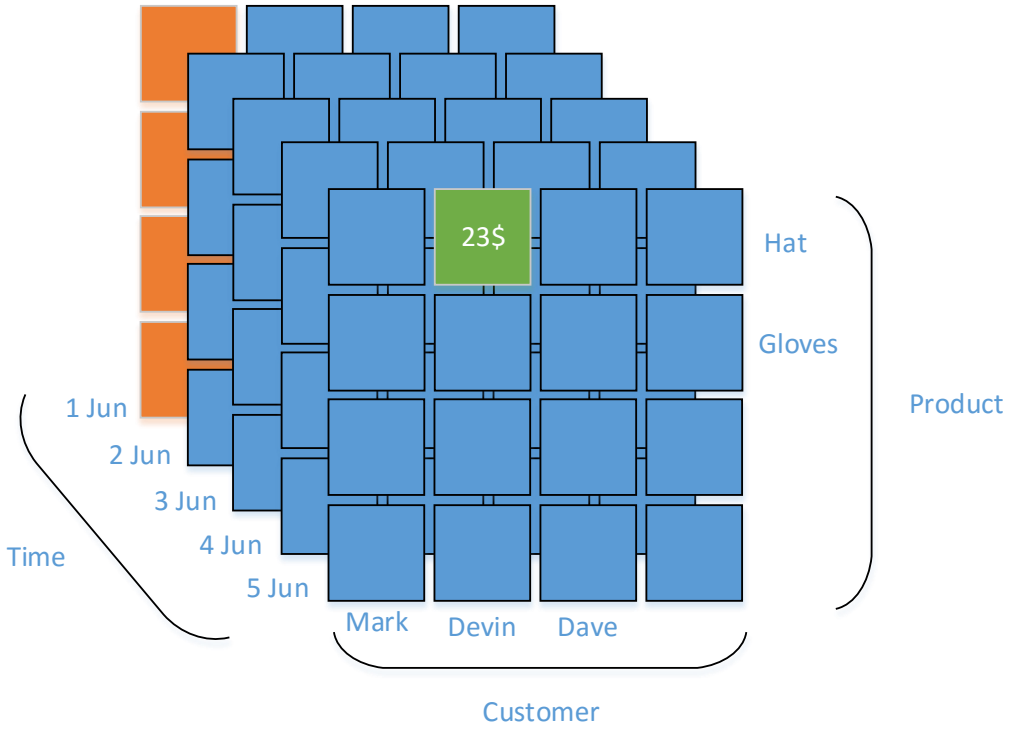
On erittäin todennäköistä, että BI-järjestelmän lähdejärjestelmänä toimii useita eri lähteitä. Tiedon yhdistämiselle on vaatimuksena tiedon lataus useasta eri lähteestä, datan muuntaminen tietovarastoon sopivaksi ja lopulta datan lataus tietovarastoon. Tätä koko ketjua kutsutaan Extract Transform Loadiksi (ETL). (Rad 2014, 10.)

ETL ei ole pelkästään datan yhdistämisvaihe. Esimerkkinä, tietokannassa voi olla useita tauluja, jotka sisältävät myyntitapahtumia. Tietovarastoa suunniteltaessa voidaan rakentaa yksi tai kaksi taulua säilömään myyntitapahtumat. ETL-prosessin pitää hakea data myyntitietokannasta ja muuttaa (yhdistää ja sovittaa yhteen) sitä, jotta se sopii tietovaraston taulujen malliin. (Rad 2014, 10.)

2.2.3 Tietomalli – Business Intelligence Scemantic Model (BISM)

Tietovarasto on suunniteltu toimimaan lähteenä analytiikalle ja raportoinnille, joten se tuottaa raportit paljon nopeammin kuin toiminnalliset järjestelmät. Tietovarasto ei kuitenkaan ole kovin nopea kattamaan kaikkia vaatimuksia, koska se on silti relaatiotietokanta ja tietokannoilla on monia riippuvuuksia, jotka hidastavat kyselyn vasteaikaa. Vaatimus nopealle suorituskyvylle ja vaatimus yhdistetylle tiedolle vaativat vielä yhden kerroksen luomisen. Niin kutsuttu tietomalli sisältää joko tiedosto- tai muistinvaraisen mallin tiedosta, jolla saavutetaan nopeat vasteet raporteille. (Rad 2014, 10.)

Microsoftin ratkaisu tietomallille on jakautunut kahteen teknologiaan: Online Analytical Prosessing, eli OLAP-kuutioon ja muistinvaraiseen tauluk-komalliin. OLAP-kuutio on tiedostoperäinen tiedonvarastointiteknologia, joka lataa datan tietovarastosta kuutiomalliin. Kuutio sisältää kuvaavaa tietoa, kuten dimensiot (esimerkiksi asiakas, tuote) ja solut (esimerkiksi faktat ja mittayksiköt, kuten myynti ja alennus). Kuvio 2 esittää OLAP-kuution. (Rad 2014, 10.)



KUVIO 2. OLAP-kuutio (Rad 2014, 11)

Kuviossa 2 esitetyssä kuutiossa on kolme dimensiotaulua: tuote, asiakas ja aika. Kuution jokainen solu edustaa näiden kolmen dimension liitoskohtaa. Esimerkiksi jos säilötään jokaisen myynnin summa yhteen soluun, niin vihreä solu näyttää että Devin maksoi 23 $ hatusta kesäkuun 5. päivä. Myös yhdistetyt tiedot on helppo hakea kuution rakenteesta. Esimerkiksi oranssi ryhmä näyttää, paljonko Mark maksoi kesäkuun 1. päivä kaikista tuotteista. Kuution rakenteen ansiosta on helpompaa ja nopeampaa päästä käsiksi tarvittavaan tietoon, kuin SQL-kyselyillä. (Rad 2014, 11.)

2.2.4 Datan visualisointi

Business Intelligence -järjestelmän käyttöliittymänä toimii datan visualisointi. Toisin sanoen datan visualisointi on se BI-järjestelmän osa, jonka käyttäjät näkevät. Tiedon visualisointiin on monia eri metodeja, kuten strategiset ja taktiset dashboardit, suorituskykymittarit (Key Performance Indicator, KPI) ja tarkat tai koostetut raportit. (Rad 2014, 11.)

Microsoft tarjoaa useita työkaluja BI:ssa tarvittavien dashboardien, suorituskykymittarien, tuloskorttien ja raporttien tekoon. Yksi näistä on Microsoft SQL Server Reporting Services (SSRS), joka on työkalu tarkkojen ja koottujen raporttien tekoon. (Rad 2014, 11.)

3 MICROSOFT-TEKNOLOGIAT

Tässä luvussa käydään läpi teknologioita, joiden tietäminen tätä opinnäytetyötä tehdessä oli hyödyllistä. Teknologioissa perehdytään niiden historiaan ja joissain poraudutaan hieman pintaa syvemmälle.

3.1 Microsoft SQL Server

Microsoft SQL Server on Microsoftin kehittämä tietokantamoottori. Microsoft SQL Server toimii asiakasohjelmiston keräämän tiedon lopullisena sijoituspaikkana ja se toimii monessa tilanteessa tiedonkeräyksen lähteenä.

3.1.1 Historia

Microsoft lähti yritystietokantamaailmaan vuonna 1997, kun se solmi kumppanuuden Sybasen kanssa, jotta Sybasen DataServer-tuote saataisiin myös Microsoft /IBM OS/2 -alustalle. Tämän yhteistyön ansiosta julkaistiin ensimmäinen SQL Server 1.0, joka oli vain porttaus Sybase DataServerin UNIX-versiosta OS/2:lle. (Boettger, Cierkowski, Leiter & Wood 2009, 1.)

Vuosia myöhemmin Microsoftin sovelluskehittäjät saivat enemmän ja enemmän oikeuksia Sybasen lähdekoodeihin, jotta testaus ja virheenkorjaus olisi helpompaa. SQL Server -ohjelmiston ydin oli vielä Sybasen omistuksessa SQL Server 4.2 julkaisuun asti, jolloin se julkaistiin Windows NT:lle maaliskuussa 1992. (Boettger ym. 2009, 1.)

SQL Server 4.2 oli ensimmäinen todellinen yhteinen tuote, jota kehitti sekä Sybase että Microsoft. Tietokantamoottori oli edelleen Sybasen, mutta työkalut ja tietokantakirjastot olivat Microsoftin toteutusta. Vaikka SQL Server oli kehitetty pääasiassa OS/2-alustalle, niin Windows NT julkaisu aloitti uuden aikakauden. Microsoftin kehittäjät käytännössä hylkäsivät kaiken OS/2 kehityksen ja keskittyivät tuomaan SQL Serverin Windows NT:lle. (Boettger ym. 2009, 1.)

3.1.2 Ensimmäinen tietokantajärjestelmä

Sybase menestyi hyvin UNIX-markkinoilla, ja Microsoft kasvatti suosiotaan Windows-markkinoilla, mikä johti siihen, että yritykset kilpailivat samoista markkinoista tuotteella, joka oli alun perin Sybasen kehittämä. Tämän takia yritykset päättivät yhteistyönsä vuonna 1994 ja Sybase antoi Microsoftille rajoitetun lisenssin käyttää ja muokata Sybase-teknologiaa yksinoikeudella järjestelmille, jotka käyttivät Windows-käyttöjärjestelmää. (Boettger ym. 2009, 2.)

Vuotta myöhemmin, kesäkuussa 1995, Microsoft julkaisi ensimmäisen täysin Microsoftin kehittäjien kehittämän SQL Serverin, SQL Server 6.0:n, mutta teknologia perustui silti suurelta osin Sybasen koodiin. Alle vuosi eteenpäin, huhtikuussa 1996, Microsoft julkaisi muunnellun SQL Server 6.5:n. (Boettger ym. 2009, 2.)

Kun SQL Server 6.5 oli saatu valmiiksi, SQL Serverin kehittäjätiimi aloitti työskentelemään uuden Sphinx-koodinimellisen tietokantajärjestelmän parissa. Sybasen lähdekoodi kirjoitettiin lähes kokonaisuudessaan uusiksi. Jäljelle jäi vain vähän koodia, joka viittasi SQL Serverin OS/2-juuriin. (Boettger ym. 2009, 2.)

Sphinx julkaistiin virallisesti joulukuussa 1998 SQL Server 7.0 nimellä. Muutokset edelliseen SQL Server 6.5-versioon olivat selkeitä heti ensimmäisten käyttösekuntien aikana, kun tietokantojen ylläpitäjä avasi uuden Enterprise Managerin. Vihdoin markkinoilla oli luotettava tietokantajärjestelmä, jota oli helppo hallita, joka oli helppo oppia ja joka oli silti riittävän tehokas monille yrityksille. (Boettger ym. 2009, 2.)

Kun SQL Server 7.0 julkaistiin, oli seuraava versio jo kehityksessä. Uuden version koodinimi oli Shiloh. Shilohista tuli lopulta SQL Server 2000, ja se julkaistiin vuoden 2000 elokuussa. Muutokset itse tietokantamoottoriin olivat pieniä, mutta SQL Serverin skaalautuvuuteen tuli uusia ominaisuuksia: indeksoidut näkymät, hajautetut tietokantapalvelimet ja taulujen väliset viite-eheydet. Microsoftin yritystietokantapalvelimesta tuli vihdoin todellinen haastaja markkinoille. (Boettger ym. 2009, 2.)

Seuraavien vuosien aikana SQL-tiimi kehitti uutta julkaisua Yukon-koodinimellä, josta tunnetaan nyt nimellä SQL Server 2005. Yli viisi vuotta myöhemmin julkaistiin tuote, jota ihmiset kutsuivat nimellä ”Yukon, suuri (Oraclen) tappaja”. (Boettger ym. 2009, 2.)

3.1.3 Business Intelligence

Vaikka SQL Server 2005:n kutsuminen “Oraclen tappajaksi” onkin aika optimistisesti sanottu, niin sitä ei voi kieltää, etteivätkö SQL Server 2005:n muutokset olisivat olleet suuri harppaus Microsoftille. Julkaisun jälkeen SQL Server on toiminut taustateknologiana monelle Microsoftin tuotteelle, kuten SharePoint-, PerformancePoint- ja System Center -tuoteperheen tuotteille. Monet kolmannen osapuolen toimittajat ovat myös hyödyntäneet SQL:ää ERP (Enterprise Resource Planning) järjestelmiin ja muihin ohjelmistotuotteisiin. (Boettger ym. 2009, 2.)

Se, missä SQL Server 2005 todella erottui kilpailijoistaan, oli sen Business Intelligence tarjonta. Tähän kuului työkalut siirtää ja muuntaa tietoa (SQL Server Integration Services), tiedon analysointityökalu (SQL Server Analysis Services) ja tiedon raportointityökalut (SQL Server Reporting Services). Nämä kolme komponenttia Notification Servicen ja Service Brokerin lisäksi olivat osa Microsoftin strategiaa saada SQL Server 2005 erottumaan muutenkin, kuin vain tietokantamoottorina. Näiden teknologioiden sisällyttäminen teki SQL Server 2005:stä houkuttelevan yrityksille, jotka olivat vasta alkaneet tutustumaan ja hyödyntämään Business Intelligenceä. (Boettger ym. 2009, 2.)

3.2 SQL Server Integration Services

Vaikka SQL Server Integration Services (SSIS) kuuluukin osaksi SQL Serveriä, niin aiheen laajuuden vuoksi asia on hyvä käydä omana lukunaan. SSIS-paketit ovat asiakkaiden järjestelmissä yleisiä. SSIS-ajojen lokeissa on tarkat selitykset tapahtumista, niiden avulla on helppo nähdä, ovatko ajot onnistuneet vai onko niissä ollut jotain ongelmia.

3.2.1 SSIS:n käyttötarkoitus

ETL (Extract, transform ja load) on pakollista tietoa kaikille kehittäjille. Yritysten tarvitsee yhdistää dataa, johtui se sitten tarpeesta lisätä sisältöä tuotekatalogiin, lisätä tietoa kauppasopimustietoihin kumppanien välillä, teknologiamigraatioihin tai uusien tietovarastojen luontiin, joka on sen yksi suosituimmista käyttökohteista. Tämän takia ETL-taidot joko Microsoftin tuotteilla, Oraclen työkaluilla tai muiden valmistajien tuotteilla ovat välttämättömiä tietojärjestelmämarkkinoilla. On tärkeää ymmärtää, kuinka nopeasti tietoteknologia kehittyy ja kuinka tärkeäksi integraatio on tullut; se on nyt osa prosessia, jossa tieto välitetään yrityksen johdolle. (Machado 2014, 12.)

SQL Server Integration Services on Microsoftin Business Intelligence paketti- ja ETL-työkalu. Jos omistaa jo SQL Serverin maksullisen lisenssin, ei kannata olla käyttämättä näitä työkaluja. Se ei ole pelkästään työkalu, jolla siirretään dataa tietokannasta toiseen. Se tarjoaa työkalut datan siivoamiseen ja datan muuntamiseen ”raahaa ja pudota” -kehitysympäristössä, joka on organisoitu tiedonsiirtotasojen mukaan. SSIS tarjoaa helpon ja käyttäjäystävällisen ympäristön projektien ylläpitämiseen. (Machado 2014, 12.)

SSIS:n päätavoitteena on luoda integroitua, puhdasta ja yhtäläistä dataa tietovarastoille, kuten OLAP-pohjaisille järjestelmille, esimerkiksi SQL Server Analysis Services -ratkaisulla tehtyihin analyysikuutioihin. Kehittäjät ovat huomanneet, että se on enemmän kuin pelkkä ETL-työkalu, sitä käytetään myös tiedon siirtämiseen tietokannasta toiseen, datan muuntamiseen kohteeseen sopivaksi ja datan siirtämiseen lähdetiedostosta toiseen. (Machado 2014, 12.)

3.2.2 SSIS:n arkitehtuuri

Microsoftin Business Intelligence -paketti on monipuolinen järjestelmä erilaisia työkaluja ja alustoja datan hallitsemiseen, muuntamiseen, tietovarastoon siirtämiseen ja datan visualisointiin monista eri perspektiiveistä, jotta

päätöksentekijöillä olisi parempi näkymä tarvittavaan tietoon. Jokainen alusta on monimutkainen ja vaatii syvän tietämyksen siitä, miten ne toimivat. (Machado 2014, 13.)

Runtime Engine

Runtime Enginen tehtävä on suorittaa paketteja, jotka sisältävät datan prosessointiin liittyviä komponentteja. Runtime Engine on SSIS:n sydän, koska se ei vain pelkästään suorita paketteja, vaan se tallentaa pakettien rakenteen ja tarjoaa tapahtumahistorioinnin, pysäytyspisteet, konfiguraatiot, yhteydet ja transaktiot. Tätä moottoria kutsutaan aina, kun halutaan ajaa SSIS-paketti, joka voidaan tehdä komentokehotteen kautta, tallennetusta proseduurista, PowerShell-skriptillä, SSIS Designerillä, SQL Agent -työkalulla tai itse tehdyllä ohjelmalla. (Machado 2014, 14.)

Integration Services Service

Integration Services Service on Windowsin palvelu Integration Services -paketeille, ja se on käytettävissä ainoastaan Sql Server Management Studiolla. Se mahdollistaa kehittäjiä tuomaan ja viemään paketteja, katsomaan ajossa olevia paketteja ja tarkastelemaan tallennettuja paketteja. Sen avulla pystyy katsomaan ajossa olevia Integration Services -paketteja SQL Server Management Studion kautta ja hallitsemaan tallennettuja paketteja. (Machado 2014, 14.)

SSIS Designer

SSIS Designer on työkalu, jolla paketit kehitetään. SSIS Designerillä voi luoda ja ylläpitää Integration Services -paketteja. Niitä voidaan käyttää seuraaviin asioihin:

- paketin ohjelmistovuon kehittämiseen
- paketin tietovirran kehittämiseen
- tapahtumankäsittelijöiden lisäämiseen paketteihin ja objekteihin
- paketin sisällön katsomiseen
- ajonaikaisen ajoketjun seuraamiseen. (Machado 2014, 14.)

Log Provider

SSIS-enginen ohjaaman log providerin toimintoihin kuuluu kaikkien lokitietojen teko, joita tarvitaan paketin suorituksen monitorointiin. Kun paketti ajetaan, log provider kirjoittaa tietoa paketin suoritukseen liittyen, kuten keston, virheet, varoitukset ja muut siihen liittyvät tiedot. (Machado 2014, 14.)

Connection Manager

SSIS-engine ohjaa connection managerin toimintoja, joihin kuuluu projektitason tai pakettitason yhteyksien ylläpito. Tämä työkalu löytyy SSIS Designeristä, jotta sitä kautta pystyy luomaan tarvittavat yhteydet datan lukemiseen ja kirjoittamiseen, esimerkiksi tiedostosta tietokantaan. Sillä voi hallita FTP (File Transfer Protocol) -palvelinyhteyksiä, tietokantayhteyksiä, tiedostoyhteyksiä ja Web-palveluyhteyksiä. SSIS connection managerin avulla SSIS -paketit on helppo yhdistää monentyyppisiin tietokantajärjestelmiin, joko Microsoft- tai muihin alustoihin. (Machado 2014, 15.)

Paketit

Paketit ovat SSIS enginen pääkomponentteja, mutta ne ovat myös alijärjestelmiä, koska ne koostuvat useasta komponentista. Paketit yhdistävät lähde- ja kohdedatan connection managerien avulla. Ne sisältävät tehtävät, joita tarvitaan kokonaisen ETL-ketjun suorittamiseen. Paketteja voidaan ajaa yksittäin, järjestyksessä, yhdistämällä muihin paketteihin tai rinnakkain suorittamalla. Paketit ovat Integration Services -projektin tuotoksia, joiden tiedostopäätteenä on .dtsx. (Machado 2014, 15.)

Tehtävät (Tasks)

Tehtävät ovat ohjausvuoelementtejä, jotka määrittelevät paketin sisäisten tapahtumien ohjausvuon. SSIS-paketit koostuvat yhdestä tai useammasta tehtävästä. Jos paketissa on useampi kuin yksi tehtävä, ne yhdistetään ja sekvensoidaan ohjausvuossa määrittämällä riippuvuudet tehtävien välille. SSIS tarjoaa useita tehtäviä moniin yleisiin ongelmiin, mutta siinä voi myös kehittää ja käyttää omia taskeja. (Machado 2014, 15.)

Tapahtumankäsittelijät (Event handlers)

SSIS Runtime Enginen hallitseman tapahtumankäsittelijän tehtävänä on suorittaa tehtävät, kun jokin tietty tapahtuma tapahtuu, esimerkiksi jos paketissa tapahtuu OnError -virhe. Eri tapahtumia varten voi luoda omia tapahtumankäsittelijöitä pakettien toiminnollisuuden parantamiseksi, ja se helpottaa pakettien ajonaikaista hallintaa. (Machado 2014, 15.)

Säiliöt (Containers)

Säiliöt ovat SSIS:n objekteja, jotka tarjoavat rakenteen paketeille ja palvelut tehtäville. Ne tukevat uudelleentoistettavia paketinsisäisiä ohjainrakenteita ja ne ryhmittävät tehtävät, ja säiliöt eri työtehtäviin. Tehtävien lisäksi säiliöt voivat sisältää toisia säiliöitä. Säiliöitä on neljää erityyppistä:

- tehtäväisäntä -säiliö (sisältää yhden tehtävän)
- for-silmukka -säiliö
- for-each-silmukka -säiliö
- sarjasäiliö. (Machado 2014, 15 - 16.)

Ohjausvuo (Control Flow)

Ohjausvuo organisoi komponenttien suoritusjärjestyksen. Sen sisällä komponentteja hallitaan järjestysehdoilla. (Machado 2014, 16.)

Projektin ja paketin käyttöönottomallit

SSIS 2012 mahdollistaa kahdentyyppisen paketin käyttöönoton. Ensimmäinen, legacy deployment model, käsittelee paketteja omina julkaisuyksikköinä. Toinen, project deployment model, luo käyttöönottopaketin, joka sisältää paketit ja parametrit, joita tarvitaan käyttöönotossa, yhteen tiedostoon. Parametrien käyttö on rajoitettu pelkästään project deployment modelille. (Machado 2014, 16.)

3.3 SQL Server Analysis Services

SQL Server Analysis Services on usealla asiakkaalla käytössä, minkä takia tämä on tärkeä osa tätä projektia. SQL Server Analysis Services tarjoaa työkalut Business Intelligence -raporttien tekemiseen. Näiden raporttien avulla asiakkaat voivat tarkastella esimerkiksi myyntiä kuukausitasolla ja tehdä päätöksiä näiden perusteella.

Analysis Services on online analytical processing (OLAP) -tietokanta, joka on optimoitu Business Intelligence (BI) -ympäristöissä käytettyihin yleisimpiin kyselyihin ja laskutoimituksiin. Se tekee monia samoja asioita kuin relaatiotietokannat, mutta eroaa usealla osa-alueella. Joissain tapauksissa on helpompaa toteuttaa BI-ratkaisu käyttämällä Analysis Servicen ja relaatiotietokannan sekoitusta, kuin käyttämällä pelkästään relaatiotietokantaa. Analysis Services ei korvaa relaatiotietokantaa eikä oikein suunniteltua tietovarastoa (data warehouse). (Ferrari, Russo & Webb 2012, 1.)

Analysis Serviceä voisi pitää ylimääräisenä metatietokerroksena tai semanttisena mallina, joka toimii relaatiotietokannan tietovaraston päällä. Tämä ylimääräinen kerros pitää sisällään tiedon siitä, kuinka fakta- ja dimensiotaulut kuuluu liittää toisiinsa, kuinka mittayksiköt (measures) pitää liittää yhteen, kuinka käyttäjien pitäisi päästä selailemaan tietoa hierarkioiden avulla, yleisimpien laskutoimitusten vastaukset ja niin edelleen. Tämä kerros sisältää yhden tai useamman mallin, joka sisältää tietovaraston liiketoimintalogiikan. Käyttäjät kyselevät (query) näitä malleja, sen sijaan, että he kyselisivät suoraan relaatiotietokantaa. Kun kaikki tämä tieto on varastoitu yhteen, jokaiselle käyttäjälle näkyvään paikkaan, käyttäjien kirjoittamien kyselyiden ei tarvitse olla niin monimutkaisia. Usein riittää, että käyttäjä kertoo, mitkä sarakkeet ja rivit tarvitaan, malli osaa pitää huolen liiketoimintalogiikan avulla siitä, että palautetut arvot ovat järkeviä. Joskus käyttäjän tekemä virheellinen kysely ei voi palauttaa väärää dataa, mikä johtuu esimerkiksi kahden väärän taulun yhdistämisestä, vaan käyttäjä saa tästä virheilmoituksen. Tämä puolestaan tarkoittaa sitä, että loppukäyttäjän käyttämien raportointi- ja analysointityökalujen ei tarvitse tehdä niin paljon töitä ja ne voivat tarjota siistimmän yleisilmeen loppukäyttäjälle. Tämän

avulla voidaan käyttää useita eri työkaluja samassa mallissa ja silti loppukäyttäjä saa saman lopputuloksen. (Ferrari ym. 2012, 1 - 2.)

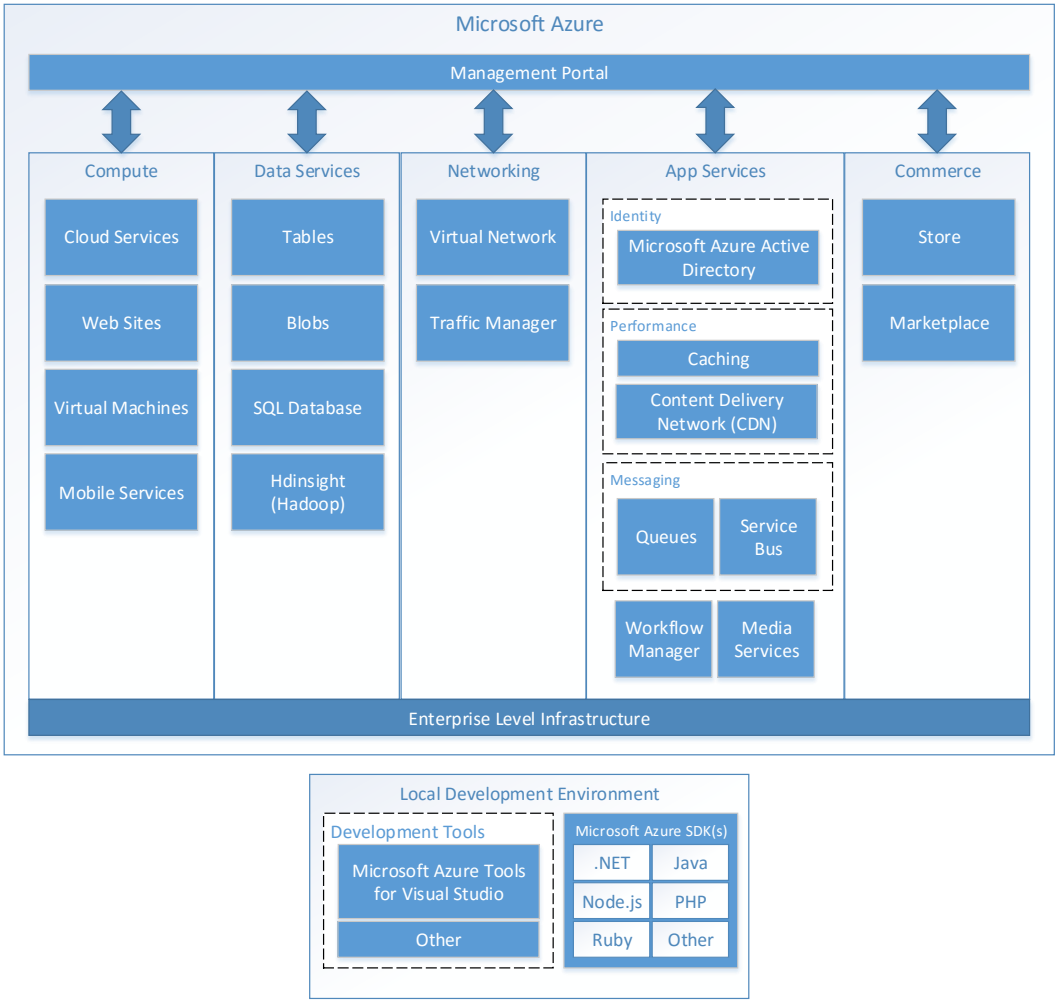
Analysis Services toimii myös varastona, jota voidaan käyttää raportoinnin nopeuttamiseen. Kun Analysis Service on käytössä, on yleistä kopioida siihen tietovarastosta löytyvä tieto. Tämän jälkeen raportointi ja analyyttiset kyselyt tehdään Analysis Serviceen eikä relaatiotietokantaan. Vaikka nykyaikaiset relaatiotietokannat ovat erittäin optimoituja ja sisältävät paljon ominaisuuksia, jotka on suunnattu BI-raportointiin, niin Analysis Services on suunniteltu erityisesti juuri tähän tarkoitukseen ja voi useimmissa tapauksissa saavuttaa paljon paremman kyselysuorituskyvyn. Loppukäyttäjän näkökulmasta tämä on tärkeää, koska sen avulla he voivat selata raporttien dataa odottamatta pitkiä aikoja. (Ferrari ym. 2012, 2.)

IT-osaston näkökulmasta eräs hyöty tulee siitä, että he voivat siirtää raporttien laatimisen loppukäyttäjille. Suurin ongelma Business Intelligence -projektien parissa, jotka eivät käytä OLAP:a on se, että IT-osaston on tehtävä tietovarasto ja sen lisäksi myös raportit sitä varten. Tämä lisää ajantarvetta ja työmäärää ja voi aiheuttaa turhautumista yritykselle, kun he huomaavat, ettei IT-osastolla ole tarvittavaa ymmärrystä raportointitarpeille eivätkä he saa niitä valmiiksi riittävän nopeasti. Kun käytössä on OLAP-tietokanta, kuten Analysis Services, IT-osasto voi antaa loppukäyttäjille oikeudet niihin malleihin, jota he tarvitsevat raporttien luontiin, jotta he voivat tehdä raportit haluamillaan työkaluilla. Suosituin asiakasohjelma on Microsoft Excel. Office 2000:n jälkeen, Excelin Pivot -taulut ovat voineet ottaa suoraan yhteyttä Analysis Services -kuutioihin ja Excel 2010 sisältää tehokkaita ominaisuuksia toimia asiakasohjelmana Analysis Servicelle. (Ferrari ym. 2012, 2.)

3.4 Microsoft Azure

Tietokanta, jota tämän opinnäytetyön ohjelmat käyttävät, sijaitsee Microsoft Azure -virtuaalipalvelimella. Microsoft Azure -pilvipalvelu on tähän hyvä valinta, koska siellä olevien palvelinten saatavuus on korkealla tasolla ja palvelun hinta määräytyy käytön mukaan. Näin säästytään siltä, ettei tarvitse huolehtia fyysisen palvelimen toiminnasta ja säilytyksestä.

Azure ei ole tuote, vaan kokonainen pilvialusta, kuten kuvio 3 osoittaa. Alimpana kerroksena ovat datakeskukset, joissa Azurea ylläpidetään. Nämä modernit datakeskukset ovat Microsoftin omistuksessa, ja niiden toiminnasta huolehtii Global Foundation Services (GFS) -tiimi, joka ylläpitää myös muita Microsoftin palveluita. Datakeskuksia sijaitsee kolmessa maanosassa: Amerikassa, Aasiassa ja Euroopassa. GFS-tiimi huolehtii myös fyysisestä ja virtuaalisesta turvallisuudesta, kuten myös yhdessä ylläpitotiimin kanssa Azuren yleisestä turvallisuudesta. (Chauhan, Fontama, Hart, Tok & Woody 2014, 21 - 22.)



KUVIO 3. Microsoft Azure -alustan yleisnäkymä (Chauhan ym. 2014, 23)

4 .NET FRAMEWORK

Puhuttaessa Microsoftin teknologioista on vaikea olla mainitsematta .NET Frameworkia. Sen avulla on mahdollista luoda helposti ohjelmia Microsoft -ympäristöihin, koska se tarjoaa hyvän tuen ja helposti käytettävät kirjastot.

4.1 .NET yleiskatsaus

Microsoft julkaisi .NET-viitekehyksen heinäkuussa, vuonna 2000. .NET-alusta on ohjelmistokomponenttikirjasto ohjelmointirajapinnalla Windowsin palveluihin ja API:n (Application Programming Interface), yhdistäen useat Microsoftin teknologiat 1990 -luvun loppupuolelta. .NET-viitekehykseen on yhdistetty COM+ (Component Services) -komponenttipalvelut: ASP (Active Server Pages) web-kehitys -viitekehys; omistautuminen XML- ja olio-orientoituneeseen suunnitteluun; tuki uusille web-palvelu -protokollille, kuten SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) ja UDDI (Universal Description Discovery and Integration), ja sen suuntaus on internetissä. Taulukossa 1 on kerrottu .NET-alustan tuoteryhmät. (Thai & Lam 2002, 5.)

TAULUKKO 1. .NET-alustan tuoteryhmät (Thai & Lam 2002, 5)

Kehitystyökalut	Ohjelmistokielien ryhmä, johon kuuluu C# ja VB.NET. Kehitystyökalut, johon kuuluu Visual Studio .NET. Kattava luokkakirjasto web-palveluiden ja web ja Windows applikaatioiden luomiseen. Kehitystyökaluihin sisältyy myös CLR (Common Language Runtime) .NET olioiden suorittamiseen.
Erikoispalvelimet	Ryhmä .NET yrityspalvelimia (Enterpise Servers), aiemmin tunnettu nimillä SQL Server 2000, Exchange 2000, BizTalk 2000, ja niin edelleen, jotka mahdollistavat erikoistuneet toiminnot relationaaliseen tiedon tallennukseen, sähköpostiin ja yritysmarkkinointiin.
Web palvelut	Maksullisten web palveluiden tarjoama, tarkemmin .NET My Services aloite, aiemmin tunnettu nimellä HailStorm. Nämä ovat maksullisia palveluita, joita ohjelmoijat voivat käyttää rakentaessaan sovelluksia, jotka vaativat tietoa henkilöiden identiteetistä.
Laitteet	Uudet .NET yhteensopivat ei-PC laitteet, kuten puhelimet ja pelikonsolit.

4.2 Microsoft .NET

Vaikka .NETin päästrategiana oli mahdollistaa ohjelmisto palveluna, on .NET paljon muutakin. Vaikka palvelu on suuntautunut vahvasti web -kehitykseen, Microsoft .NET noudattaa ja toteuttaa taulukon 2 mukaisia kehityssuuntia. (Thai & Lam 2002, 6.)

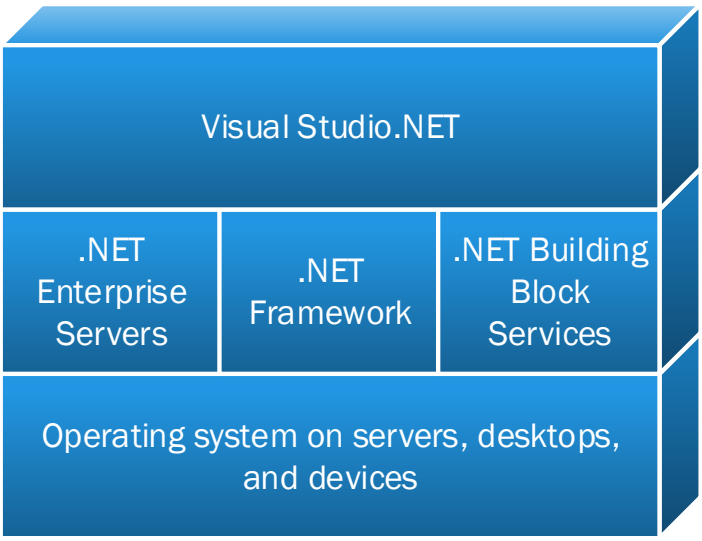
TAULUKKO 2. .NET Framework kehityssuuntaukset (Thai & Lam 2002, 6)

Hajautettu laskenta	Asiakas/palvelin ohjelmien kehityksen yksinkertaistaminen. Aiemmat hajautetut teknologiat vaativat korkeaa toimittaja-sitoutumista ja niistä puuttui yhteen toimivuus internetin kanssa. Microsoft .NET tarjoaa hajautetun arkkitehtuurin, joka hyödyntää avoimia internet standardeja, kuten HTTP (Hyper Text Transfer Protocol), XML (Extensible Markup Language) ja SOAP (Simple Object Access Protocol).
Komponentointi	Eri kehittäjien kehittämien ohjelmistokomponenttien integroimisen helpottaminen. COM (Component Object Model) plug and play ominaisuuden ohjelmistoihin, mutta COM komponenttikehitys ja käyttöönotto on liian monimutkaista. Microsoft .NET tarjoaa yksinkertaisemman tavan rakentaa ja ottaa komponentteja käyttöön.
Yrityspalvelut	.NET mahdollistaa skaalautuvien yritysohjelmistojen kehityksen, ilman että tarvitsee kirjoittaa koodia joka hallitsee transaktiot, turvallisuuden tai resurssien jakamisen. Yrityspalveluiden kannalta Microsoft .NET tarjoaa huomattavasti lyhyemmät kehitysajat ja pienemmän työmäärän, joka kuluu suurien ohjelmistokehitysprojektien kehitykseen, kuin esimerkiksi C++-ohjelmointikielellä kehitetyt ohjelmat.
Web paradigma muutos	Edustaa web-teknologian muutosta yksinkertaisemman kehitysprosessin suuntaan web-applikaatioissa. Vuosien saatossa web-applikaatioiden kehitys on siirtynyt yhdistämisestä (TCP/IP) esittämiseen (HTML) ja siitä ohjelmoitavuuteen (XML/SOAP). Microsoft .NETin päätavoitteena on mahdollistaa ohjelmistojen myynnin ja levittämisen palveluna.
Maturiteettitekijät	Tällä tarkoitetaan oppeja, joita ohjelmistoala on oppinut suurien yritys- ja web-applikaatioiden kehittämisestä. Kaupallisten web-applikaatioiden täytyy tukea yhteentoimivuutta, skaalautuvuutta ja hallittavuutta. Microsoftin .NET sisältää kaikki nämä tavoitteet.

Vaikka nämä ovatkin Microsoft .NET-viitekehyksen pääkonseptit (taulukko 2), niin vielä huomattavampaa on se, että Microsoft .NET käyttää avoimia internetstandardeja (HTTP, XML ja SOAP) sen ytimenä olioiden siirtoon koneelta toiselle internetin välityksellä. XML:n ja .NET-olioita yhdistää kaksisuuntainen avainarvopari. Esimerkiksi, luokka voidaan esittää XML-skeema -määritelmänä (XML Schema Definition, XSD). Olio voidaan muuntaa ja lukea XML -puskurista. Metodi voidaan spesifioida käyttäen WDSL (Web Services Description Language) -formaattia ja metodikutsu voidaan esittää XML -formaatissa SOAP:n avulla. (Thai & Lam 2002, 6.)

4.3 .NET-alusta

Microsoftin .NET-alusta koostuu viidestä pääkomponentista, jotka näkyvät kuviossa 4. Alinpana kerroksena on käyttöjärjestelmä (Operating System), joka voi olla yksi Windows -käyttöjärjestelmistä, kuten Windows 10, Windows 8.1, Windows 7, Windows CE tai jokin muu Windows-käyttöjärjestelmä. Osana .NET -strategiaa Microsoft on luvannut toimittaa lisää .NET-laiteohjelmistoja uuden sukupolven älylaitteille. (Thai & Lam 2002, 6.)



KUVIO 4. Microsoft .NET-alusta (Thai & Lam 2002, 7)

Käyttöjärjestelmäkerroksen päällä on joukko .NET Enterprise Server -tuotteita, jotka nopeuttavat suurien yritysjärjestelmien kehittämistä. Näitä

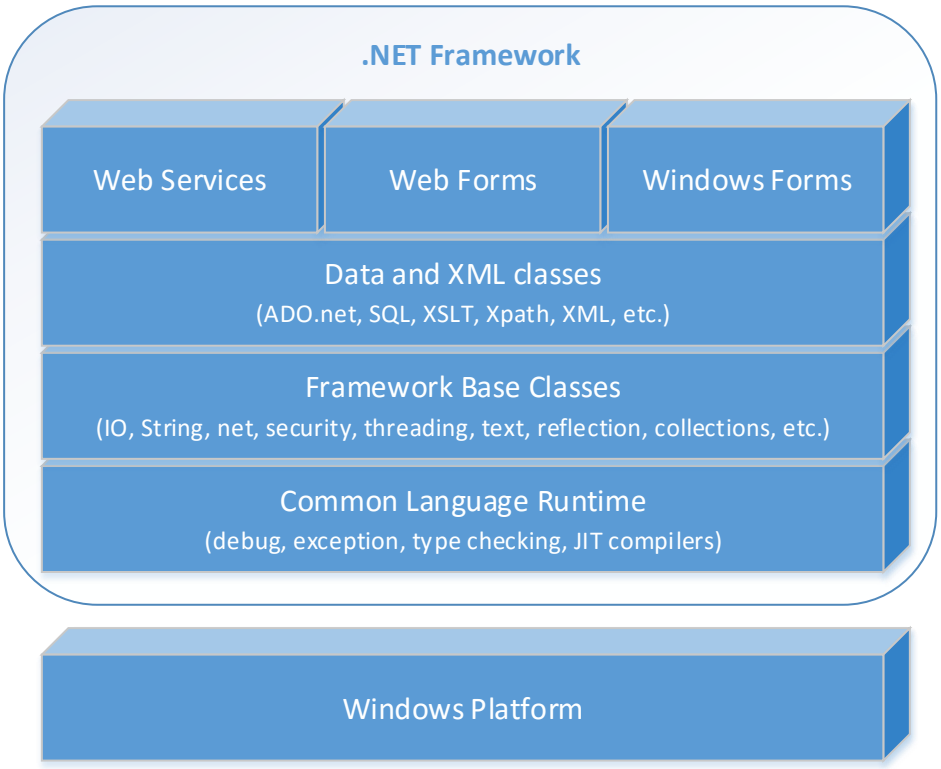
palvelintuotteita ovat esimerkiksi Application Center (ei enää tukea), BizTalk Server, Commerce Server, Exchange Server, Host Integration Server, Internet Security and Acceleration Server ja SQL Server. (Thai & Lam 2002, 7.)

Päällimmäinen kerros .NET-arkitehtuurissa on Microsoftin kehitystyökalu Visual Studio.NET (VS.NET), jonka avulla on mahdollista kehittää nopeasti web-palveluita ja muita ohjelmia. Visual Studio.NET korvasi Microsoft Visual Studio 6.0:n. Visual Studio.NET on integroitu ohjelmointiympäristö (Integrated Development Environment, IDE), joka tukee useita eri ohjelmointikieliä ja ominaisuuksia, kuten ohjelmointikielien välistä virheenkorjausta ja XML-skeema editoria. (Thai & Lam 2002, 7.)

Keskellä .NET-alustaa on Microsoft .NET-viitekehys. .NET-viitekehys on ohjelmointi- ja ajoinfrastruktuuri, joka muutti yrityssovellusten kehittämisen Windows -alustoilla. Se sisältää Common Language Runtime- (CLR) ja yleisiä viitekehysluokkia, joita voidaan käyttää kaikissa .NET-ohjelmointikielissä. (Thai & Lam 2002, 7.)

4.4 .NET-viitekehys

Kuten kuviosta 5 nähdään, .NET-viitekehys on käyttöjärjestelmän päällä, joka voi olla jokin Windows -versio (nykypäivänä toimii myös muissakin käyttöjärjestelmissä) ja koostuu useista komponenteista. Käytännössä .NET-viitekehys on vain sovellus, joka toimii Windowsissa ja muissa käyttöjärjestelmissä. (Thai & Lam 2002, 11.)



KUVIO 5. .NET-viitekehys (Thai & Lam 2002, 12)

Viitekehyksen tärkein komponentti on CLR. Java-ohjelmoijan on helppo ajatella, että CLR on .NETin vastaava Java Virtual Machinelle (JVM). Muutoin voi miettiä, että CLR on .NET-arkitehtuurin sydän. Korkealla tasolla CLR aktivoi olioita, suorittaa turvallisuustarkistuksia niille, sijoittaa ne muistiin, suorittaa ne ja suorittaa niille roskankeruun. (Thai & Lam 2002, 12.)

Käsitteellisesti CLR ja JVM ovat samanlaisia siinä suhteessa, että ne molemmat ovat suoritusinfrastruktuureja, jotka abstraktoivat taustalla olevat alustaerot. Kuitenkin siinä missä JVM tukee tällä hetkellä vain Java -ohjelmointikieltä, CLR tulee kaikkia kieliä, jotka kuuluvat Common Intermediate Language (CIL) -ohjelmointikieliin. JVM suorittaa tavukoodia, joten teknisesti sekin tukee useita eri ohjelmointikieliä. Ero näiden infrastruktuurien välillä on se, että Java -koodia on voitu suorittaa alusta lähtien eri alustalla, kun taas .NET-koodia ei aluksi voitu suorittaa kuin Windows CLR -alustoilla. (Thai & Lam 2002, 12.)

Kuten kuviosta 5 huomataan, CLR -kerroksen yläpuolella on viitekehyksen perusluokat. Nämä perusluokat ovat vastaavia kuin Standard Template

Libraryn (STL), Microsoft Foundation Classesin (MFC), Active Template Libraryn (ATL) tai Javan perusluokat. Nämä luokat esimerkiksi tukevat alkeellisia luku- ja kirjoitustoimintoja, tekstinmuokkausta, turvallisuuden hallintaa, verkkotoimintoja, säikeenhallintaa, tekstinhallintaa ja keräystoiminnallisuutta kuten myös muitakin funktioita. (Thai & Lam 2002, 12.)

Viitekehyksen perusluokkien päällä on joukko luokkia, jotka tukevat datan hallintaa ja XML -muokkausta. Dataluokat tukevat persistenttiä datanhallintaa tiedolle, joka on tallessa backend -tietokannoissa. Nämä luokat sisältävät Structured Query Language (SQL) -luokkia, jotka mahdollistavat datan muokkausta persistenteissä tietovarastoissa standardin SQL rajapinnan kautta. Vastaavasti kuin SQL-luokat, luokkakokoelma nimeltään ADO.NET mahdollistaa persistentin datan muokkaamisen. Dataluokkien lisäksi .NET-viitekehys tukee useita luokkia, jotka mahdollistavat XML-datan muokkauksen, XML -hakujen tekemisen ja XML -transformaation. (Thai & Lam 2002, 12.)

Luokat kolmessa eri teknologiassa, Web Services, Web Forms ja Windows Forms (kehitys lopetettu vuonna 2005), laajentavat viitekehyksen perusluokkia, dataluokkia ja XML-luokkia. Web Services sisältää useita luokkia, jotka tukevat kevyiden hajautettujen komponenttien kehitystä, jotka toimivat palomuuri- ja Network Address Translation (NAT) -ohjelmistojen kanssa. Nämä komponentit tukevat plug-and-play -ominaisuutta internetin välityksellä, koska Web Services käyttää standardeja HTTP- ja SOAP-protokollia. (Thai & Lam 2002, 13.)

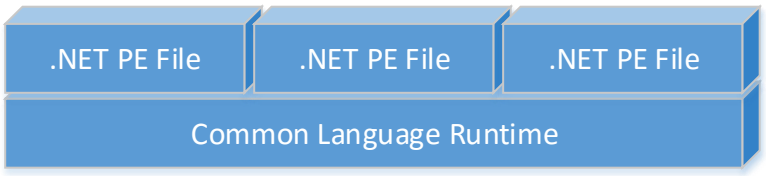
Web Forms sisältää useita luokkia, jotka mahdollistavat nopean, graafisen käyttöliittymän (Graphical User Interface, GUI) sisältävän web -ohjelman luomisen. Web Forms mahdollistaa Web GUI:n luomisen raahaa-pudota tyylisesti, tavallisten Windows Forms -ohjelmien GUI:n tapaan. (Thai & Lam 2002, 13.)

Windows Forms tukee useita luokkia, jotka mahdollistavat natiivi Windows GUI -ohjelmien luomisen. Nämä luokat ovat oikeastaan parempi versio MFC:stä, koska ne tukevat helpompaa GUI -kehitystä ja tarjoavat yleisen,

konsistentin käyttöliittymän, jota voidaan käyttää kaikilla ohjelmointikielillä. (Thai & Lam 2002, 13.)

4.5 Common Language Runtime ympäristö (Environment)

.NET-infrastruktuuri sisältää Common Language Runtime (CLR) -ympäristön. Toisin kuin ohjelmistokirjastot, kuten MFC tai ATL, CLR on kehitetty puhtaalta pöydältä. CLR hallitsee .NET-viitekehyksen suoritettavasta koodista. Kuvio 6 näyttää .NET-viitekehyksen kaksi osa-aluetta, joista alempi on CLR ja ylempi osio on CLR:n suoritettavat tai siirrettävät (Portable Executable, PE) tiedostot, jotka ovat .NETin osia tai käyttöönottoyksiköitä. CLR on suoritusmoottori, joka lataa tarvittavat luokat, suorittaa ohjelman tarvitsemien metodien ajonaikaisen kääntämisen, pakottaa turvallisuustarkastuksia ja suorittaa paljon muita ajonaikaistoimintoja. CLR:n suoritettavat tiedostot, jotka näkyvät kuviossa 6, ovat joko EXE- tai DLL-tiedostoja, jotka koostuvat suurimmaksi osaksi metadatasta tai koodista. (Thai & Lam 2002, 14.)



KUVIO 6. Common Language Runtime -ympäristö (Thai & Lam 2002, 14)

Microsoftin .NET suoritettavat tiedostot eroavat tavallisista Windowsissa ajettavista ohjelmista. Tavalliset ajettavat ohjelmat sisältävät ohjelman koodin ja datan, kun taas .NET ajettavat ohjelmat sisältävät koodin lisäksi myös metadatan. (Thai & Lam 2002, 14.)

5 EXTENSIBLE MARKUP LANGUAGE

Extensible Markup Languagea käytetään muun muassa SQL-yhteyksien säilöntään ja tiedonvälitykseen aliohjelman ja pääohjelman välillä. Näistä ratkaisuista on kerrottu enemmän työnosuudessa.

ADO.NET (ActiveX Data Objects) on siirtynyt pois COM-pohjaisista tietuejoukoista ja siirtynyt käyttämään XML:a tiedonsiirtoon. Koska XML on alustariippumaton, voidaan sen tietoja lukea millä tahansa ohjelmalla, joka pystyy lukemaan tai kirjoittamaan XML-muotoisia tiedostoja. Tämä on suuri etu ADO:iin nähden, koska sen COM-pohjaiset tietuejoukot eivät ole alustariippumattomia. (Thai & Lam 2002, 115.)

5.1 XML-jäsennin

Vaikka XML on pelkkää tekstiä, on XLM-tiedoston lukeminen ja tiedostoon kirjoittaminen helpompaa ohjelmallisesti. Tämän mahdollistaa XML-jäsennin. XML-jäsentimiä on kahdentyppisiä: parseri- ja tapahtumapohjaisia. (Thai & Lam 2002, 115.) Näistä kahdesta keskitytään vain tapahtumapohjaiseen, josta käytetään termiä puupohjainen, tämän puumallisen rakenteen takia.

Puupohjainen XML-jäsennin lukee tiedoston tai bittivirran kokonaisuudessaan ja luo niistä XML-solmupuun. XML-solmua voidaan pitää XML-tagina, josta esimerkki kuviossa 7. (Thai & Lam 2002, 115.)

```
<elokuva>
  <nimi>Taru sormusten herrasta: Sormuksen ritarit</nimi>
  <lajityyppi>Fantasia</lajityyppi>
  <vuosi>2001</vuosi>
  <ohjaaja>Peter Jackson</ohjaaja>
</elokuva>
```

KUVIO 7. Esimerkki XML-tiedostosta

Puumuotoon jäsenneltynä, XML-tiedostosta löytyy yksi juurisolmu, elokuva, kuten kuviosta 7 selviää. Elokuvan alapuolelta löytyy neljä solmua: nimi, lajityyppi, vuosi, ohjaaja. Jos tietomäärä on suuri, ei ole suositeltavaa

käyttää puumallia, koska silloin puurakenteesta tulisi liian iso ja se veisi paljon keskusmuistia. (Thai & Lam 2002, 115.)

5.2 XML .NET-Frameworkissa

XML:n ydinluokat voidaan kategorisoida niiden funktioiden mukaan, dokumenttien kirjoitukseen ja lukemiseen, dokumenttien validoimiseen, solmujen navigointiin ja valintaan, skeematiedon hallintaan ja dokumenttimuutoksiin. Koko XML .NET Framework löytyy system.xml.dll-kirjastosta. (Esposito 2003, 6.)

.NET Framework tarjoaa myös XML-objektien serialisoinnin. Nämä toiminnot löytyvät System.Xml.Serialization nimiavaruudesta. Xml-serialisointi kirjoittaa objekteja XML-dokumenteiksi ja lukee XML-dokumenteiksi käännettyjä objekteja. Tällainen muunnos on erityisen hyödyllistä yhdistämällä SOAP:n ja .NET Framework XML Web-palvelun. (Esposito 2003, 6.)

XML on tiukasti integroituna .NET Frameworkiin tiedonsiirtoteknologiana. Taulukko 3 esittää, missä alueissa XML on käytössä .NET Frameworkissa. Jokainen osa-alue sisältää useita luokkia, ja ne tuottavat paljon ohjelmistotason funktioita. (Esposito 2003, 6.)

TAULUKKO 3. XML:n näkyvyys .NET Frameworkissa (Esposito 2003, 6)

Kategoria	Kuvaus
ADO.NET	Datasäiliöt, kuten DataSet objektit siirretään ja etäkäsitellään XML:n avulla. .NET Framework tarjoaa myös kaksisuuntaisen synkronoidun sidoksen tabulaarisen dataformaatin ja XML-formaatin välillä.
Asetukset	Sovellusasetukset säilötään XML-tiedostoihin hyödyntäen ennalta määritettyjä ja käyttäjän määrittämiä osa-alue lukijoita.
Etäkäyttö	.NET Frameworkin etäobjekteihin pääsee käsiksi valmistelemalla ja suorittamalla kutsut SOAP-paketeilla.
Web palvelut	SOAP on kevyt XML-protokolla, jota web-palvelut käyttävät tiedonvälitykseen hajautetuissa ympäristöissä. SOAP:aa voidaan käyttää kutsumaan web-palveluiden metodeja alustariippumattomasti.
XML Parsinta	Ydinluokat tarjoavat XML-parsinnan ja manipuloinnin virtaus API:n (Application programming interface) ja XMLDOM (XML Document Object Model) kautta.
XML Serialisointi	Mahdollistaa olioinstanssien tallentamisen ja lukemisen XML-dokumenttimuodosta.

5.3 XML-serialisointi

Serialisointi on ajonaikainen prosessi, joka muuntaa olion tai ryhmän olioita bittivirraksi. Tätä lopputuotosta voidaan käyttää olion säilömiseen tai verkon yli siirtämiseen. Microsoft .NET Framework sisältää kolme eri serialisointimahdollisuutta: Binääri, Simple Object Access Protocol (SOAP) ja XML. (Esposito 2003, 388.)

Ajonaikainen olion serialisointi, kuten esimerkiksi binääri serialisointi ja XML-serialisointi, on erityyppisiä teknologioita eri implementoinneilla ja eri tavoitteilla. Kuitenkin molemmat serialisoinnit tekevät vain yhden asian, ne muuntavat olion tilan muistiin, josta ne siirretään toiseen tallennustilaan, kuten kiintolevylle. Ajonaikaista serialisointia hallitaan .NET Frameworkin formatter -olioilla. XML-serialisointi kuuluu XMLSerializer-luokkaan. (Esposito 2003, 388.)

XML-serialisointiprosessi muuntaa olion julkisen rajapinnan tiettyyn XML-skeemaan. Samaa mekanismia käytetään laajalti .NET Frameworkissa olion tilan tallennukseen virtaus- tai muistipuskuriin. Web-palvelut käyttävät XMLSerializer-luokkaa muuntamaan metodien palauttamien olioiden instanssit. (Esposito 2003, 388.)

5.4 Olion serialisointiprosessi

.NET Frameworkissa olion serialisointi tapahtuu System.Runtime.Serialization-nimiavaruuden luokkien avulla. Deserialisointi on serialisointiprosessi käännettynä. Deserialisointi käyttää serialisoitua oliota ja luo olion siitä informaatiosta. (Esposito 2003, 388.)

Olion serialisointi .NET Frameworkissa mahdollistaa julkisten, suojattujen ja yksityisten kenttien tallennuksen, ja se hoitaa automaattisesti kehäviittaukset. Kehäviittaus tapahtuu, kun lapsiolio viittaa pääolioon ja pääolio viittaa myös lapsiolioon. Serialisointiluokat .NET Frameworkissa tunnistavat nämä kehäviittaukset ja korjaavat nämä. Serialisointi voi generoida ulostulodatan moneen eri muotoon käyttämällä erilaisia räätälöityjä jäsenninmoduuleja. Kaksi järjestelmän tarjoamaa jäsennintä ovat BinaryFormatter- ja SoapFormatter-luokat, jotka kirjoittavat olion tilan binäärimuotoon ja SOAP-muotoon. (Esposito 2003, 388.)

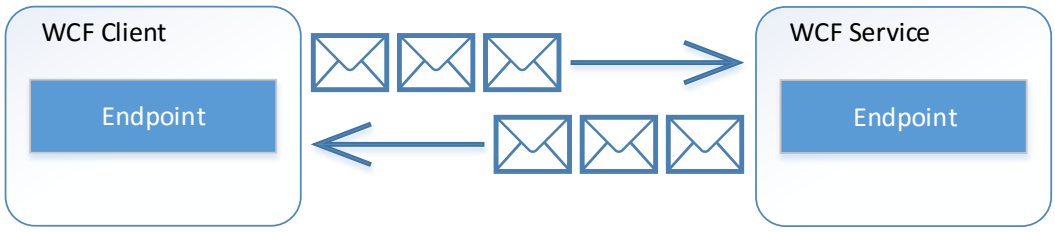
6 WINDOWS COMMUNICATION FOUNDATION (WCF)

Windows Communication Foundation, eli WCF on teknologia, jonka avulla opinnäytetyön työnosuudessa tehty ohjelmisto keskustelee Microsoft Azuren kanssa. Tämä teknologia helpottaa kommunikaation rakentamista kahden pisteen välillä.

Palvelut ovat keskeisessä asemassa globaalissa hajautetussa verkossa ja WCF on helppo tapa tuottaa ja hyödyntää palveluita Microsoft-alustoilla. Hyödyntämällä WCF:a, kehittäjät voivat keskittyä itse ohjelmiston tekoon, eikä tarvitse käyttää aikaa kommunikointiprotokollien tekemiseen. (Bowen, Crane & Resnick 2008, 1.)

6.1 WCF-palvelu

Palvelu on joukko päätepisteitä, jotka tuottavat hyödyllisiä toimintoja asiakasohjelmistolle. Päätepiste on verkkoresurssi, jonne voidaan lähettää viestejä. Asiakasohjelmistot hyödyntävät näitä toimintoja lähettämällä viestin päätepisteelle formaatissa, joka on sovittu asiakasohjelmiston ja päätepisteen kesken. Palvelut kuuntelevat viestejä tietyssä osoitteessa ja odottavat viestien saapuvan oikeassa formaatissa. Kuviossa 8 on esitetty palvelun ja asiakasohjelmiston välinen suhde perustasolla. (Bowen ym. 2008, 3.)

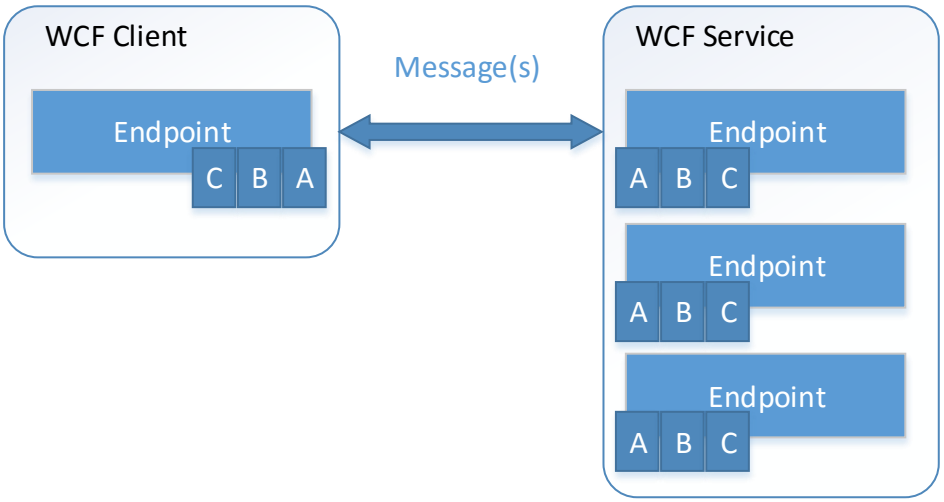


KUVIO 8. Palvelun ja asiakasohjelman välinen kommunikointi (Bowen ym. 2008, 4)

Jotta asiakasohjelmisto voisi kommunikoida palvelun kanssa, sen tulee tietää ABC, joka tarkoittaa seuraavaa:

- A – osoite, minne. Se määrittää, minne verkossa viesti pitää lähettää, jotta päätepiste voi vastaanottaa sen. Tämä on kohde, jonne asiakasohjelmiston pitää lähettää viestit. Hyper Text Transfer Protocol, eli http-osoite näyttäisi seuraavalta http://minunpalvelin/minunpalvelu/, kun taas TCP-osoite olisi net.tcp//minunpalvelin:8080/minunpalvelu.
- B – sidos, miten. Sidos määrittää kommunikointikanavan päätepisteen kanssa. Kanavat ovat putkia, joita pitkin kaikki viestit menevät WCF-ohjelmien sisällä. Kanava koostuu useasta sidoselementistä. Alimman tason sidoselementti on siirto, joka välittää viestit verkon yli. Sisäänrakennetut siirtotavat ovat HTTP, TCP, Named Pipes, PerChannel ja Message Queuing (MSMQ). Näiden yläpuolella on sidoselementit, jotka määrittävät turvallisuuden ja transaktiot.
- C – sopimus, mitä. Sopimus määrittelee operaatiot, jotka päätepiste tarjoaa, sekä viestiformaatin, jonka operaatiot vaativat. Sopimusoperaatiot ovat sidonnaisia luokka-metodeihin, jotka päätepiste implementoi, kuten parametrien allekirjoitukset, jotka syötetään ja palautetaan metodeista. (Bowen ym. 2008, 4.)

Kuten kuviosta 9 huomataan, monta päätepistettä luo WCF-palvelun, jossa jokainen päätepiste on määritelty osoitteella, sidoksella ja sopimuksella. Koska viestit kulkevat yleensä kaksisuuntaisesti, päätepistellä on myös epäsuorasti päätepiste. (Bowen ym. 2008, 5.)



KUVIO 9. Asiakasohjelman ja palvelupäätepisteen välinen kommunikointi (Bowen ym. 2008, 5)

Päätepiste ei voi vastata viestiin, ennen kuin palvelu isännöidään käyttöjärjestelmän prosessissa. Isäntä voi olla mikä tahansa prosessi, kuten valvomaton palvelinprosessi, web-palvelin tai jopa koko näytössä oleva asiakasohjelmisto tai asiakasohjelma pienennettynä. Palveluilla on toimintaa ohjaavia käyttäytymisiä, jotka kontrolloivat toimivuutta, suorituskyvyn rajoittamista, transaktioita, turvallisuutta ja muita järjestelmä-semantiikkoja. Käyttäytyminen voidaan implementoida käyttäen .NET attribuutteja, manipuloimalla ajonaikaista WCF suoritusta tai määritysten avulla. Käyttäytyminen ja joustava ylläpitomalli yhdessä vähentävät kompleksisuutta monisäikeistä koodia kirjoittaessa. (Bowen ym. 2008, 5.)

6.2 Sopimukset

Tavallisessa maailmassa sopimus on kahden tai useamman osapuolen välinen sitoomus, joka määrittelee tuotteiden tai palveluiden tuottamisen tiettyyn hintaan. Tietokonemaailmassa sopimus on samantyylinen: se on kahden tai useamman välinen sitoomus, jotka määrittelee viestit jota voidaan välittää, ja ehdot, joiden mukaan toimitaan. (Bowen, Crane, Resnick 2008, 33.)

Sopimus on kuvaus viesteistä, joita välitetään päätepisteelle ja päätepisteeltä toisaalle. Jokainen päätepiste on määritelty verkko-osoitteella, jonne

viestit lähetetään, sitomuksella, joka kuvaa, kuinka viestit lähetetään, ja sopimuksella, joka kuvailee viestien formaatin. (Bowen ym. 2008, 33.)

Palvelu on kokoelma päätepisteitä, ja päätepisteet implementoivat tietyt algoritmit koodissaan. Ne voi implementoida korkean tason liiketoimintafunktioita, kuten tilausten syöttämisen tilausjärjestelmään tai ne voivat hakea esimerkiksi asiakkaan osoitteen. Korkean luokan funktiot vaativat usein monimutkaisia tietorakenteita, kun taas kohdistetut funktiot toimivat usein yksinkertaisemmilla tietotyypeillä. Kummassakin tapauksessa päätepisteen pitää määritellä toiminnot, jotka se sisältää, ja oikean dataformaatin, jossa se olettaa tiedon olevan. Yhdessä nämä määritykset luovat sopimuksen. (Bowen ym. 2008, 33.)

WCF:ssa on kolmen tyyppisiä sopimuksia:

- Palvelusopimus: Palvelusopimukset kuvaavat toiminnalliset operaatiot, jotka on implementoitu palveluun. Palvelusopimus muuntaa .NET-luokkametodit Web Service Description Language (WSDL) -palveluiksi, porttityypeiksi ja operaatioiksi. Operaatiosopimukset palvelusopimuksien sisällä kuvaavat palveluoperaatioita, jotka ovat metodeja, jotka on toteutettu palvelun funktioissa.
- Datasopimus: Datasopimus kuvaa datastruktuurin, jota palvelu käyttää asiakasohjelmien kanssa kommunikointiin. Datasopimus muuntaa CLR-tyypit XML Shema Definitions (XSD) -muotoon ja määrittää, kuinka ne serialisoidaan ja deserialisoidaan. Datasopimukset kuvaavat kaikkea dataa, joka vastaanotetaan palveluoperaatioille tai lähetetään palveluoperaatioilta.
- Viestisopimukset: Viestisopimukset muuntavat CLR-tyypit Simple Object Access Protocol (SOAP) -viesteiksi ja vaikuttavat niiden viestien WSDL- ja XSD-määrityksiin. Viestisopimukset tuottavat tarkan kontrollin SOAP-otsikkoihin ja runkoihin. (Bowen ym. 2008, 34.)

Toimivuus useassa järjestelmässä on saavutettu siten, että palvelut on esitetty Web Service Description Language (WSDL) -kielellä. World Wide Web Consortiumin, joka tunnetaan paremmin lyhenteenä W3C, mukaan

suuret toimittajat, kuten esimerkiksi Microsoft ja IBM, määrittivät WSDL -spesifikaation seuraavasti: WSDL on XML-formaatti verkkopalveluiden kuvaamiseen päätepisteinä, jotka toimivat viestien välillä, jotka sisältävät joko dokumenttiorientoitunutta tai proseduuriorientoitunutta informaatiota. Operaatiot ja viestit on kuvattu abstraktisti, minkä jälkeen ne sidotaan konkreettiseen verkkoprotokollaan ja viestiformaattiin, joka määrittää päätepisteen. Läheiset konkreettiset päätepisteet yhdistetään abstarkteiksi päätepisteiksi, eli palveluiksi. WSDL on laajennettavissa, jotta se sallii päätepisteiden kuvauksen ja niiden viestit, riippumatta missä viestiformaatissa viesti on tai mitä verkkoprotokollaa käytetään. (Bowen ym. 2008, 34.)

W3C tuottaa WSDL:n täyden spesifikaation, jotta toimittajat, kuten Microsoft, voivat hyödyntää tehdessään työkaluja WSLD:n hyödyntämiseen. WSDL:n pääelementit löytyvät taulukosta 4. (Bowen ym. 2008, 35.)

TAULUKKO 4. WSDL-elementit (Bowen ym. 2008, 34)

WSDL Elementti	Kuvaus
Tyyppi	Datatyyppikuvaus, jota käytetään välitettävien viestien kuvaukseen. Nämä ovat yleensä esitetty XML Schema määrittelyssä.
Viesti	Kuvaa abstractia määritystä välitettävälle datalle. Viesti koostuu loogisista osista, joista jokainen on yhdistetty jonkin tyyppijärjestelmän kuvaukseen. Viesti on samantyyppinen kuin funktiokutsun parametri tai metodin parametri rajapinnassa ja sitä käytetään operaatioiden allekirjoitukseen.
Operaatio	Palvelun tukeman toiminnon nimi ja kuvaus. Operaatiot kuvaavat palvelupäätepisteen toiminnot ja ominaisuudet.
Porttityyppi	Nimetty ryhmä abstrakteja operaatioita ja abstrakteja viestejä. Palvelupäätepiste implementoi porttityyppi, mihin ryhmään toiminnot kuuluvat.
Sidos	Määrittää viestiformaatin ja protokollatiedot operaatioille ja viesteille, tietylle porttityypille.
Portti	Määrittää yksilölliset päätepisteet, määrittämällä yhden osoitteen sidokselle.
Palvelu	Määrittää yhteenkuuluvien porttien ryhmät.

7 ASIAKASSOVELLUKSEN TOTEUTUS

Asiakassovellus toteutettiin CGI Suomi Oy:n Lahden-toimipisteelle. Työn tarkoituksena oli automatisoida asiakasyrityksille tehtävät jatkuvaan palveluun kuuluvat aamuvalvonnat. Asiakkaiden ympäristöissä on pääasiassa Microsoftin teknologioita, joten työ toteutettiin tukemaan näitä. Tämän vuoksi sovellus on suunniteltu toimimaan Microsoft Windows Server -käyttöjärjestelmien ja Microsoft SQL Serverin kanssa.

Valvonnat suoritetaan manuaalisesti, mikä tarkoittaa etäyhteyden muodostamista asiakkaan ympäristöön, jossa suoritetaan vaaditut toimenpiteet. Kaikki valvonnat sisältävät joitain yhteneväisyyksiä, mutta asiakaskohtaisia eroja löytyy. Syy, miksi tällaista automatisoitua valvontajärjestelmää tarvitaan, on se, että näin vapautetaan valvonnan suorittavat henkilöt nopeammin muihin työtehtäviin. Sen lisäksi automaattinen valvonta auttaa reagoimaan nopeammin mahdollisiin virhetilanteisiin. Kun tiedot on historioitu yhteen paikkaan, voidaan asiakaskohtaisesti tehdä analysointia ja hallita tulevia häiriöitä jo ennen niiden tapahtumista. Esimerkiksi, jos vapaan levytilan määrä laskee tasaista tahtia, voidaan nähdä etukäteen, milloin levytilaa on kriittisen vähän. Historioitu näkymä auttaa myös näkemään, jos jokin tietty komponentti aiheuttaa usein virheitä. Tämän tiedon avulla latausten varmuutta pystytään parantamaan, kun pystytään reagoimaan komponenttitasolla ongelmaan.

Asiakassovelluksen tarkoituksena on poistaa kokonaan se vaihe, että valvonnan suorittavan henkilön tarvitsisi ottaa etäyhteys asiakkaan ympäristöön valvontoja tehtäessä. Asiakassovellus suorittaa samat toimenpiteet kuin valvonnan suorittava henkilö ja lähettää valvonnan tuloksen Azuressa sijaitsevaan tietokantaan. Tietenkin ongelmatilanteissa virhe täytyy selvittää menemällä asiakkaan ympäristöön.

7.1 Kokonaisuuden suunnittelu

Ennen tietokannan tai ohjelmiston suunnittelua oli luotava yleinen käsitys siitä, miten ohjelmisto tulisi kokonaisuudessaan toimimaan. Ilman kunnollista suunnittelua valvontasovelluksen ja asiakasohjelmiston yhteensovittaminen olisi ollut hankalaa, ellei jopa mahdotonta. Esimerkeissä käytetyt asiakkaat ovat keksittyjä eivätkä ne liity CGI:n asiakaskuntaan millään tasolla.

7.1.1 Liikennevalonäkymä

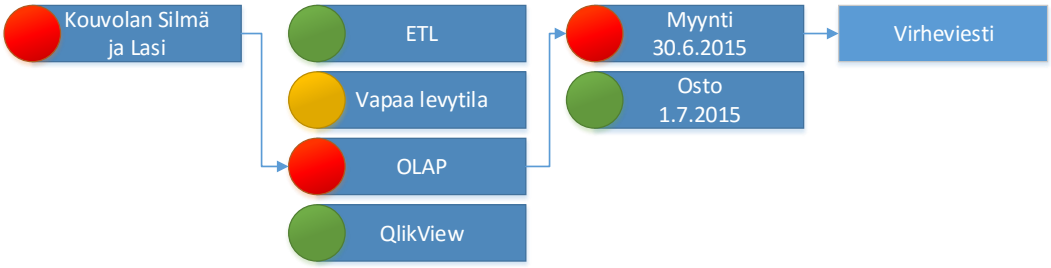
Valvontojen automatisointiin oli olemassa dokumentti, joka toimi suunnittelun pohjana. Dokumentissa oli niin sanottu liikennevalonäkymä, jonka avulla pystyisi yhdellä silmäyksellä näkemään, onko asiakkaan ympäristössä ollut yön aikana ongelmia. Vaikka tämä liikennevalonäkymän toteutus ei kuulunut tämän opinnäytetyön piiriin, on sen ymmärtäminen tärkeää. Liikennevalonäkymän idea löytyy kuviosta 10.

KUVIO 10. Liikennevalonäkymä

Kuvion 10 esittämä liikennevalonäkymä olisi se, minkä valvonnan suorittava henkilö näkisi heti valvontaa aloittaessa. Vihreä tarkoittaa sitä, että asiakkaan ympäristössä ei ole tapahtunut virheitä. Keltainen tarkoittaa sitä, että virheitä on ollut sellaisessa paikassa, mikä ei vaikuta raportointiin, eikä vaadi reagointia välttämättä heti. Tällainen tapaus voi olla vaikka se, että vapaan levytilan määrä lähestyy kriittistä rajaa. Punainen tarkoittaa

sitä, että raportointi on joko virhettä edeltävällä tasolla tai kokonaan saavuttamattomissa.

Pelkkä tällainen liikennevalonäkymä ei vielä anna riittävän tarkkaa tietoa siitä, mikä tilanne on ja mitä asiakkaalle voi ilmoittaa. Tämän takia tarvitaan ominaisuus, että raporttiin voisi porautua. Tätä porautumista voisi käydä aina virheviestiin asti, kuten kuvio 11 esittää.



KUVIO 11. Liikennevalonäkymässä porautuminen

Kuten kuvio 11 esittämästä porautumisesta huomataan, on Kouvolan Silmä ja Lasi asiakkaalla ongelma OLAP-kuutioiden kanssa. Vaikka vapaa levytila lähestyykin kriittistä rajaa, niin ylimmällä tasolla näytetään punainen valo, koska asiakkaalla on vakavampi virhe. Myyntikuution liikennevalo on punaisena, koska sen prosessointipäivä ei ole tämä päivä. Tämä tarkoittaa sitä, että myynneissä ei ole otettu huomioon eilen tapahtuneita myyntejä, mikä voi vaikuttaa päätöksentekoon ja kuukausiraportointiin. Kun Myynti-kuutiosta porautuu syvemmälle, voi nähdä virheviestin, joka liittyy kuution prosessoimattomuuteen.

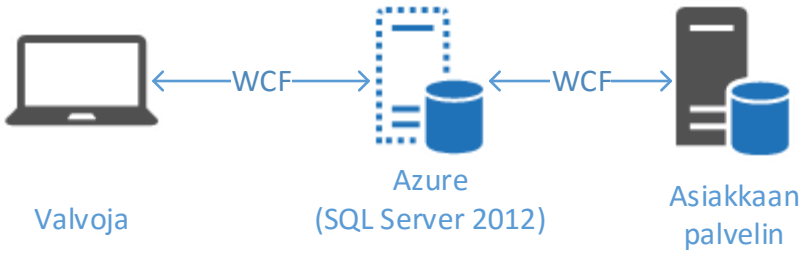
7.1.2 Liikennevaloista eteenpäin

Liikennevalonäkymä vei ratkaisun kohti hajautettua järjestelmää. Näin ollen on mahdollista lisätä helposti uusia tarkistuksia vanhojen rinnalle, eikä asiakkaalle tarvitse toimittaa kuin yksi aliohjelma, kokonaisen ohjelmiston sijaan. Yksi osa on CGI:n käyttämä näkymäosuus, jonka avulla tarkastellaan asiakkaan ympäristöjen tilaa. Toinen osa on tietovarasto, johon kaikki tämä tieto tallennetaan ja josta näkymäosuus hakee tietonsa. Kolmantena

osana on asiakasohjelmisto, joka kerää tiedot asiakkaan ympäristöstä, lähettää ne tietovarastoon. Tämä opinnäytetyö keskittyy tietovarastoon ja asiakasohjelmistoon.

Tiedot oli saatava asiakkaan ympäristöstä välitettyä joko suoraan CGI:lle, tai tietoturvalliseen kolmannen osapuolen sijoituspaikkaan. Tähän hyvin sopivan ratkaisun toi Azure-pilvipalvelu. Azuren palvelut on hajautettu, joten palvelun saatavuus on lähes sataprosenttinen. Azuressa toimiva virtuaalipalvelin ei vaadi tilaa konesalista ja Microsoft vastaa laitteiston ylläpidosta. Näin CGI:n ei tarvitse huolehtia mahdollisista laiterikoista ja muista fyysisen palvelimen ylläpitoon liittyvistä asioista. Azure -virtuaalipalvelimelle asennettiin SQL Server 2012, joka toimi tietokannan hallintajärjestelmänä. Siitä, millaiseen tauluratkaisuun tässä opinnäytetyössä päästiin, kerrotaan paremmin Tietokanta-luvussa.

Asiakkaan päähän tarvittiin ohjelman, joka pystyy suorittamaan valvonnassa tehtävät rutiinit. Siihen tehtävään olisi voitu tehdä yksi ohjelma, mutta aina muutoksia tehtäessä olisi täytynyt kääntää koko ohjelma uudelleen. Tämä olisi aiheuttanut myös sen, että asiakkaille olisi mennyt sellaisia toimintoja, joita heidän ympäristössään ei tarvitse olla. Ratkaisu oli yksi pääohjelma, joka suorittaa aliohjelmia. Pääohjelma on runko joka suoritetaan kun aamuvalvonta tehdään ja aliohjelma suoritetaan kun kyseistä tarkistusta tarvitaan. Tästä löytyy enemmän tietoa Asiakasohjelmisto-kappaleesta.



KUVIO 12. Valvontatiedon siirto

Tiedonvälityksessä päädyttiin käyttämään Windows Communication Foundation (WCF) -rajapintaa, koska se tarjoaa turvallisen ja helpon tavan siirtää tietoa koko ketjun läpi. Yleiskuva ketjusta löytyy kuviosta 12.

7.2 Tietokanta

7.2.1 Johdanto

Tietokannat ovat tärkeässä osassa asiakkaiden ympäristöissä. Tietokannat toimivat myös tämän opinnäytetyön runkona. Aina kun käytetään tietokantoja, on suunnittelulla suuri rooli. Jos tietokanta on huonosti suunniteltu ja toteutettu, voi sen käyttö ja ylläpito olla haastavaa.

Tietokannan suunnitteluun kului paljon aikaa, eikä ohjelmiston tekemistä voitu aloittaa ennen tietokantasuunnitelman valmistumista. Koska käytetyt teknologiat olivat jo ennalta tiedossa, tietokannan hallintajärjestelmän valintaan ei tarvinnut käyttää aikaa. Tietokannan hallintajärjestelmänä toimii Microsoft SQL Server 2012.

7.2.2 Tietokannan toteutus

Tietokannan toteutus alkoi valvontojen läpikäymisestä vaihe vaiheelta paperilla. Valvontoja läpikäydessä tarvittiin tieto siitä, mitä tarkistetaan, miten tarkistetaan ja mitkä ovat mahdolliset lopputulemat. Jokainen näistä tiedoista vaikuttaa tietokannan rakenteeseen.

Tieto siitä, mitä tarkastetaan, vaikuttaa asiakasohjelmistolle välitettäviin tietoihin. Nämä välitettävät tiedot ovat tarpeellisia, koska ne antavat tarkistuksille tarvittavat määritykset, kuten mitä tietokantaa tarkistetaan, mitä taulua halutaan tarkistaa.

Se, miten tarkistetaan, määrittää sen, millainen aliohjelman tarvitsee olla. Levytilan ja OLAP-kuution prosessointiajan hakeminen eroaa toisistaan prosessina niin paljon, ettei niitä voida toteuttaa samanlaisilla metodeilla. Tämän takia täytyi suunnitella, miten toteutetaan ratkaisu, jonka avulla voidaan välittää tieto siitä, millä aliohjelmalla kyseinen toiminto suoritetaan.

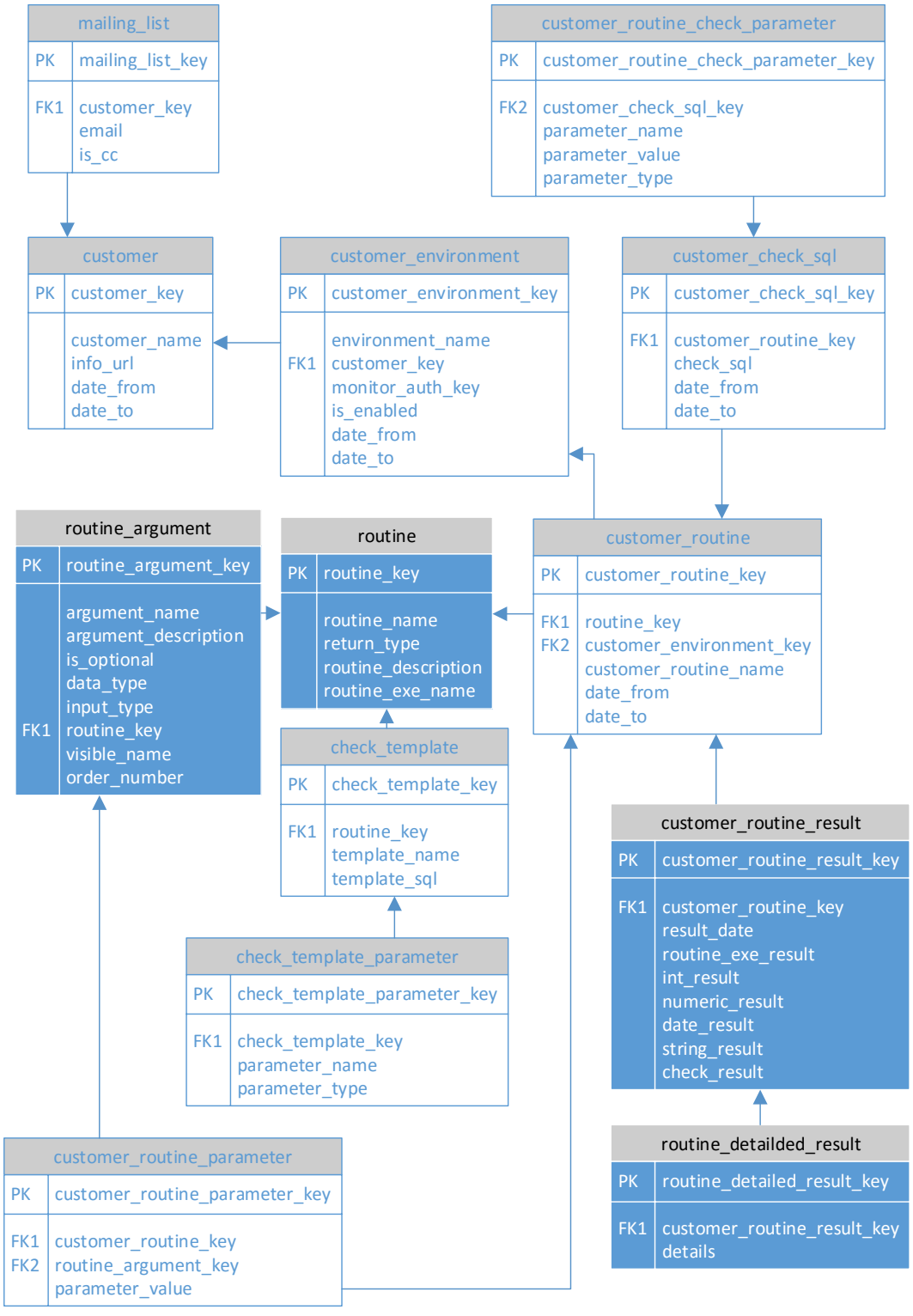
Mahdolliset lopputulemat määrittävät, minkälaisessa muodossa palautus tulee. Levytila palautetaan bitteinä ja kuution prosessointiaika aikaleimana. Nämä laitetaan omiin sarakkeisiinsa taulussa, jotta erityyppiset paluuarvot

saadaan helpommin käsiteltyä, ilman ETL-prosessia. Tietenkään ei ole mahdotonta sijoittaa kaikkea dataa samaan sarakkeeseen, mutta tiedonkäsittelyn kannalta tällainen ratkaisu ei olisi kannattava.

7.2.3 Tietokannan kokonaiskuva

Tietokantasuunnittelu ei noudata suoraan lumihiutale- tai tähtimallia. Tietokanta noudattaa enemmän lumihiutalemallia, koska routine-taulu on keskeisimpänä tauluna tietokannassa, kuvio 13.

Tietokanta koostuu yhteensä kolmestatoista eri taulusta. Sinisellä värillä olevat taulut, routine_argument, routine, customer_routine_result ja routine_detailed_result, kuuluvat asiakassovelluksen käyttämiin tauluihin, jotka näkyvät kuviossa 13. Muut taulut ovat valvontasovelluksen käytössä.



KUVIO 13. Tietokantakuvaus

7.2.4 Taulujen tarkemmat selitykset

Koska pelkästä tietokannan kaaviokuvasta ei välttämättä selviä, mikä on minkäkin taulun käyttötarkoitus, on ne lueteltu taulukossa 5. Taulukko 5 sisältää jokaisen taulun yleiskuvauksen. Tarkemmat selitykset löytyvät taulukon jälkeisestä osiosta.

TAULUKKO 5. Taulujen nimet ja selitykset

Taulu	Kuvaus
check_template	Check_template-taulussa on pohjat kyselyille, joita käytetään raporttia näyttäessä.
check_template_ parameter	Check_template_parameter-taulussa on kyselypohjien tarvitsemat argumentit.
customer	Tämä taulu pitää sisällään asiakkaan tiedot, kuten asiakkaan nimi ja linkin asiakkaan intra-sivulle. Tämä taulu täytetään valvontojen ylläpitosovelluksella.
customer_check_sql	Taulu sisältää pohjat tulosten vertailuun tarvittaviin kyselyihin. Tämä taulu täytetään valvontojen ylläpitosovelluksella.
customer_ environment	Customer_environment-taulussa on asiakkaan ympäristön tiedot, kuten onko kyseessä tuotanto- vai testiympäristö. Tämä taulu täytetään valvontojen ylläpitosovelluksella.
customer_routine	Customer_routine yhdistää tietyn rutiinin ja asiakkaan ympäristön johon kyseinen rutiini ajetaan. Tämä taulu täytetään valvontojen ylläpitosovelluksella.
customer_routine_ check_parameter	Tähän tauluun tulee check_templaten käyttämät parametrit. Tämä taulu täytetään valvontojen ylläpitosovelluksella.
customer_routine_ parameter	Tässä taulussa yhdistetään oikeat rutiiniargumentit oikealle ympäristölle. Tämä taulu täytetään valvontojen ylläpitosovelluksella.
customer_routine_ result	Tähän tauluun tulee jokaisen asiakkaan ympäristössä suoritetun tarkistuksen tulos. Asiakasohjelmisto lähettää tulokset tähän tauluun.
mailing_list	Mailing_list-tauluun laitetaan sähköpostiosoitteet, johon mahdolliset virheviestit, tai sovitusti virheettömän valvonnan tulokset lähetetään. Tämä taulu täytetään valvontojen ylläpitosovelluksella.
routine	Routine-tauluun lisätään jokainen uusi aliohjelma. Tauluun määritellään aliohjelman nimi, palautuksen tyyppi (string, date jne.), selite ja suoritettavan ohjelman nimi. Tähän lisätään manuaalisesti uusi rivi, kun uusia aliohjelmia valmistuu.
routine_argument	Routine_argument-taulussa määritellään kullekin aliohjelmalle argumentit, jotka aliohjelma ottaa sisään. Tähän lisätään manuaalisesti uusi rivi, kun uusia aliohjelmia valmistuu.
routine_detailed_ result	Taulu sisältää valvontatulosten tarkemmat tiedot. Tämän taulun toimintoja ei ole vielä tätä opinnäytetyötä kirjoittaessa implementoitu.

Kaikki taulut sisältävät pääavaimen (Primary key), joka on rivin yksilöivä tieto. Tämän takia sitä ei ole erikseen mainittu alla olevassa listassa. Lähes kaikista tauluista löytyy nimitieto, minkä takia sitä ei ole erikseen mainittu jokaisen taulun kohdalla, alla olevassa listassa. Melkein kaikista tauluista löytyy vierasavain (Foreign key) viittaus toiseen tauluun ja useimpiin tauluihin viitataan vierasavaimella. Seuraavassa on kerrottu tarkemmin taulujen viittauksista ja käytöstä:

- check_template: Taulu yhdistyy routine-tauluun, jotta tiedetään, minkä rutiinin kysely kyseinen rivi on. Taulusta löytyy sarake, josta löytyy kyseisen rivin tarvitsema sql-kysely.
- check_template_parameter: Tämä taulu yhdistyy check_template-tauluun, jonka avulla välitetään argumentit oikealle tarkistukselle.
- customer: Tässä taulussa on asiakkaan nimi ja linkki intrassa olevaan asiakassivuun, josta löytyvät myös päivämääräarvot, se, milloin asiakas on otettu valvontaan mukaan ja milloin se on lähtenyt valvonnan piiristä pois. Tällä saadaan varmistettua, ettei huomioida sellaisia asiakkaita, jotka eivät ole enää CGI:n asiakkaina valvonnan osalta.
- customer_check_sql: Taulussa määritellään raja-arvot tarkistuksen tuloksille. Esimerkiksi määritellään, milloin kiintolevytila on tietylle kiintolevylle liian alhainen, jolloin liikennevalonäkymässä se saa punaisen pallon. Taulu on yhteydessä customer_routine-tauluun, jonka avulla yhdistetään tieto oikealle asiakkaalle oikeaan ympäristöön. Tauluun on myös määritelty, milloin tarkistus on lähtenyt pois käytöstä tai muuttunut. Näin voidaan tarkistella historioidusti, jos esimerkiksi kiintolevyn kriittiset arvot muuttuvat.
- customer_environment: Taulu on yhteydessä customer-tauluun. Tämän liitoksen ansiosta yhdelle asiakkaalle voidaan määrittää useampi ympäristö, kuten tuotanto ja testi. Riveillä on customer-taulusta tuttu poistumispäivämäärä ja erillinen lippu jolla voidaan määrittää joku ympäristö pois tarkistuksesta, esimerkiksi huollon ajaksi.

- customer_routine: Taulussa yhdistetään routine- ja customer_environment-taulut. Tänne tallennetaan kaikkien ympäristöjen rutiinit. Tässä taulussa voidaan määritellä historioidusti poistumispäivämäärä.
- customer_routine_check_parameter: Taulu yhdistyy customer_check_sql-tauluun. Täällä määritellään sallitut arvot tarkistettaville asioille, esimerkiksi kiintolevytiloille.
- customer_routine_result: Tämä taulu on asiakasohjelmiston tuottaman tarkistustuloksen käyttämä taulu. Taulu yhdistyy customer_routine-tauluun, jonka avulla yhdistetään tulos oikeaan ympäristöön. Taulussa on paluuarvot ohjelman suorituksen onnistumiselle, int-tyyppiselle paluuarvolle, numeeriselle paluuarvolle, päivämäärä paluuarvolle ja tekstityppiselle paluuarvolle. Nämä helpottavat tiedon käsittelyä valvontaraportille.
- mailing_list: Taulu yhdistyy customer-tauluun. Tällä taululla voidaan määritellä tietyt postituslistat asiakkaan mukaan. Tämä mahdollistaa automaattisen sähköpostigeneroinnin, joka helpottaa valvontaviestin lähetystä.
- routine: Routine tauluun lisätään käsin uusi rivi, aina kun tulee uusi tarkistusohjelma mukaan. Taulussa määritellään ohjelman tuottama paluutyyppi, joka helpottaa paluuarvojen käsittelyä. Taulussa määritellään aliohjelman nimi, esimerkiksi OLAP.exe. Tämä taulu mahdollistaa ohjelman dynaamisen rakenteen. Kun jokainen aliohjelma on omana ohjelmanaan, voidaan asiakkaalle viedä vain tarvittavat ohjelmat. Tällaisen ratkaisun avulla on myös helppo luoda uusia ohjelmia, ilman että vanhoihin tarvitsee tehdä muutoksia.
- routine_argument: Routine_argument-taulu yhdistyy routine-tauluun. Tämä taulu täytetään myös manuaalisesti, aina uuden aliohjelman tullessa mukaan tarkistuksiin. Taulussa määritellään aliohjelman sisään ottamat argumentit, eli käynnistysparametrit, niiden oikea järjestys ja niiden muoto (string, int). Esimerkiksi OLAP.exe tarvitsee parametreiksi connection string arvon, eli tietokantaan yhdistämiseen tarvittavan komennon ja OLAP-kuution nimen.

- routine_detailed_result: Tämä taulu sisältää customer_routine_result-taulun rivien tarkemmat tiedot. Tähän tauluun tallennetaan tieto, jos halutaan esimerkiksi palauttaa virheviesti SSIS-paketin virheestä.

8 ASIAKASOHJELMISTO

8.1 Johdanto

Asiakasohjelmisto on ohjelmisto, joka suorittaa samat toiminnot, kuin manuaalisessa valvonnassa. Kun ohjelmisto on kerännyt tarvittavat tiedot, välittää se ne eteenpäin Azure-palvelimelle.

8.2 Asiakasohjelmiston suunnittelu

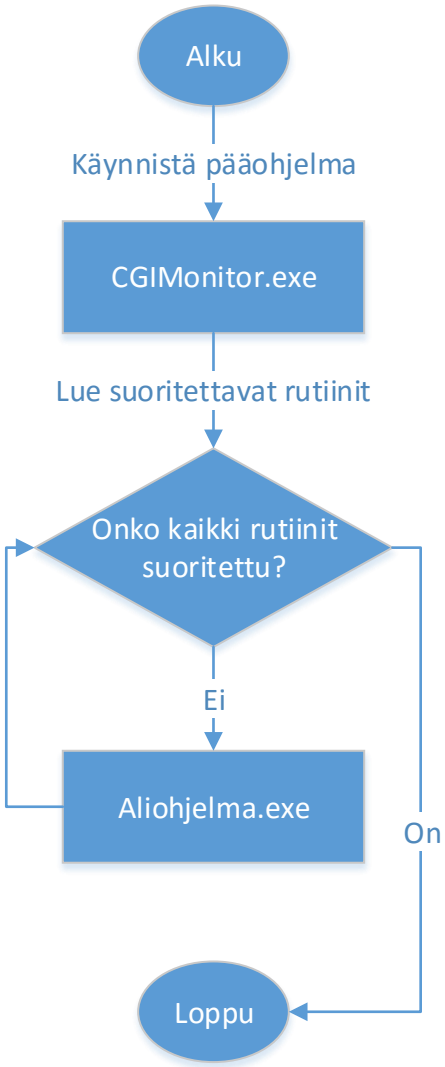
Suunnittelua aloittaessa oli otettava huomioon useita seikkoja. Ohjelmointikieli oli helpoin valinta, koska käytössä oli muutenkin Microsoftin teknologioita, ohjelmointikieleksi valikoitui C#. Tärkeintä suunnittelussa oli se, että ohjelmisto on varmatoiminen ja antaa oikean tuloksen. Väärä tulos aiheuttaisi sen, että valvontaraportilla kaikki näyttäisi olevan kunnossa, vaikka asiakkaan ympäristössä olisikin tapahtunut virhe.

Osittain tästä syystä ohjelmisto suuntautui kahteen eri osioon: pääohjelma ja aliohjelmat. Kuvio 14 näyttää karkean toimintamallin, tarkemmat toimintalogiikat löytyvät Pääohjelma- ja Aliohjelmat-otsikoiden alta.

Ohjelmiston jakaminen kahteen osa-alueeseen antaa myös mahdollisuuden helppoon laajennettavuuteen. Aliohjelmia voi luoda aina uusia pääohjelman pysyessä muuttumattomana. Tällainen suunnittelu tuo myös vikasietoisuutta, sillä aliohjelman kaatuminen ei kaada koko ohjelmistoa, vaan muut tarkastukset voidaan vielä suorittaa.

Se, miten asiakasohjelmisto keskustelee aliohjelmien kanssa ja aliohjelma pääohjelman kanssa, oli yksi suunnitteluun vaikuttava asia. Vaihtoehtoina oli tallentaa valvonnan tulos tiedostoon tai serialisointi. Näistä ratkaisuista serialisointi tarjoaa parhaan suorituskyvyn ja luotettavuuden. Serialisoinnin ansiosta tietoa ei tarvitse parsia tekstitiedostosta, vaan valvonnan tulos voidaan tallentaa omaan tietoluokkaansa. Tietoluokan serialisointi olisi voi-

nut tapahtua myös binääritiedostoon, mutta ratkaisussa päädyttiin käyttämään Named pipes tekniikkaa. Pääohjelma käynnistää aliohjelman ja jää odottamaan aliohjelman palauttamaa serialisoitua tulosta.



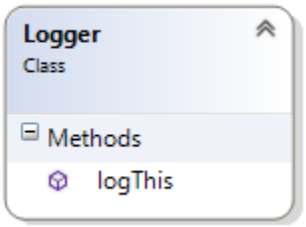
KUVIO 14. Ohjelmiston suorituskaavio

8.3 Luokat

Ilman luokkia ohjelmasta ei saanut yhtä modulaarista, kuin luokkien avulla. Luokilla on useita eri tarkoituksia, mutta suurin osa luokista pitää sisällään toiselle ohjelmalle tai ohjelman myöhempään vaiheeseen liittyvää tärkeää tietoa.

8.3.1 Logger

Logger-luokkaa käytetään virheiden kirjoittamiseen lokitiedostoon. Virheen sattuessa saadaan tallennettua tarkempi virheviesti ilman, että sitä täytyy lähettää Azuressa sijaitsevaan tietokantaan. Tämä luokka on käytössä kaikilla ohjelmilla ja luokissa, joissa käytetään Try-Catch -menetelmää. Luokka ottaa sisään kolme arvoa, joista selviää ohjelma, joka kaatui, lyhyt virheviesti ja tarkemman tason virheviesti.

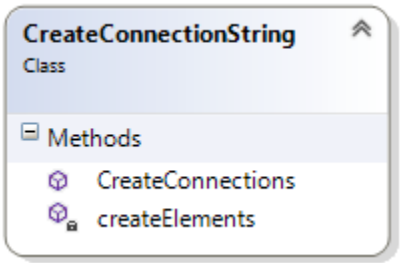


KUVIO 15. Logger-luokkakaavio

Logger-luokalla ei ole kuin yksi metodi: logThis. Se kirjoittaa lokitiedostoon kaatuneen ohjelman nimen, lyhyen virheviestin ja tarkemmat tiedot virheestä. Virheen tiedot välitetään kaatuneesta ohjelmasta Exception-luokkaa hyödyntämällä.

8.3.2 ConnectionStringCreater

Asiakkaan ympäristössä voi olla useampia eri tietokantoja, minkä vuoksi saatetaan tarvita useampia kantayhteystietoja. Tämä luokka on luotu helpottamaan kantayhteyksien lukua ja kantayhteystiedoston luontia.



KUVIO 16. CreateConnectionString-luokkakaavio

ConnectionStringCreater-luokka (kuvio 16) kirjoittaa kantayhteydet XML-tiedostoon. Luokka ottaa sisäänsä listan SQL-yhteyksistä ja OLAP-yhteyksistä, jotka käydään for-each-metodilla läpi. Kuviossa 17 on esimerkki kantayhteystiedostosta, jossa on kaksi SQL-tietokantaa ja yksi SQL Server Analysis Services -kanta.

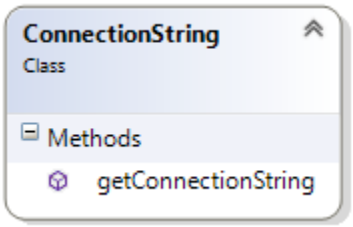
```
<?xml version="1.0" encoding="UTF-8"?>
<connections>
  <sqlConn1>Data Source=localhost;Initial Catalog=Production;
        User id=cgimon;Password=pwd;</sqlConn1>
  <sqlConn2>Data Source=localhost;Initial Catalog=Test;
        User id=cgimon;Password=pwd;</sqlConn2>
  <olapConn1>Data Source=localhost;Initial Catalog=MOLAP;</olapConn1>
</connections>
```

KUVIO 17. Tietokantayhteyksien XML-tiedosto

Luokkaa käytetään valvontoja tekevässä sovelluksessa. Valvontaa suorittavat aliohjelmat hyödyntävät luotua XML-tiedostoa.

8.3.3 ConnectionString

ConnectionString-luokka lukee kuviossa 16 kuvatun CreateConnectionString-luokan luomaa yhteystiedostoa. Aliohjelma, joka tarvitsee tietokannan yhteystietoja, saa yhdeksi käynnistysparametriksi XML-solmun nimen, jonka avulla se löytää tarvittavan yhteystiedon samanlaisesta tiedostosta, joka on kuvattuna kuviossa 17. Kuten kuviosta 18 huomaa, luokka sisältää vain yhden metodin.



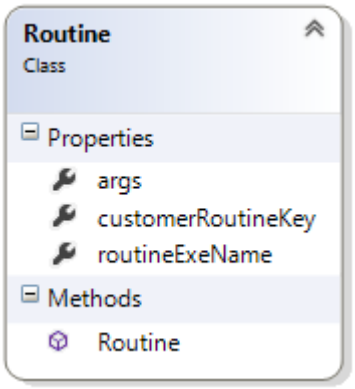
KUVIO 18. ConnectionString-luokkakaavio

XML-toteutus helpottaa paljon ympäristöissä, joissa on paljon eri tietokantoja ja joihin valvonta kohdistuu. Näin ollen aliohjelmaa käynnistäessä ei

tarvitse välittää pitkää yhteystieto tekstiä, vaan voidaan välittää lyhyt tietokantayhteyden määrittävän solmun nimi.

8.3.4 Routine

Routine, eli rutiini on aliohjelman suoritukseen tarvittava luokka. Routine-luokan avulla pääohjelma osaa käynnistää oikean aliohjelman ja antaa sille tarvittavat parametrit ja tunnistetiedon tietokantaan.

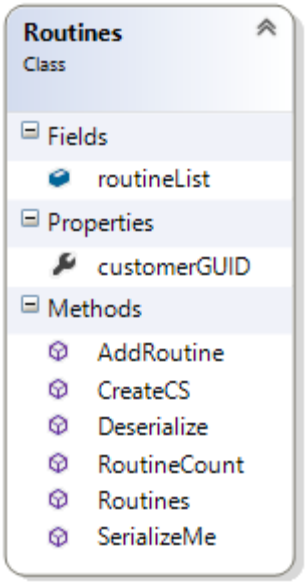


KUVIO 19. Routine-luokkakaavio

Kuten kuviossa 19 näkyy, luokasta muodostettuun olioon tallennetaan aliohjelman tarvitsemat argumentit, tietokantaan liittyvän rutiiniavaimen ja rutiinin nimi. Nämä tiedot tallennetaan Routines-säiliöluokkaan.

8.3.5 Routines

Routines-luokka kokoaa kaikki routine-luokan oliot yhteen asiakasympäristökohtaiseen tiedostoon. Routines-luokka sisältää useita toimintoja (kuvio 20), ja sitä käytetään valvontaprofiilienluonnissa ja valvontaprofiilia läpikäydessä asiakkaan ympäristössä.

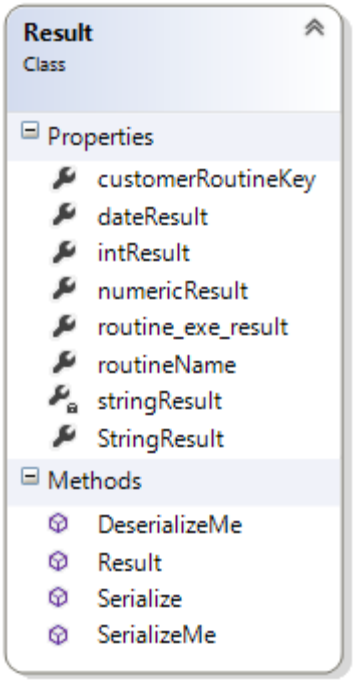


KUVIO 20. Routines-luokkakaavio

Metodi AddRoutine lisää rutiinin routineList-listaan. CreateCS on käytössä tietokantayhteyksiä luodessa, ja se hyödyntää aiemmin mainittua Create-ConnectionString-luokkaa. SerializeMe ja Deserialize liittyvät nimensä mukaisesti serialisointiin. Kun kaikki asiakkaan ympäristöön liittyvät rutiinit on lisätty routineList-luokkaan, serialisoidaan routines-luokan olio SerializeMe-metodilla. Tämä luo binääritiedoston, jossa on kaikki lisätyt rutiinit. Tiedosto viedään asiakkaan palvelimelle, josta pääohjelma lukee kaikki tiedot serialisoidusta binääritiedostosta Deserialize-metodilla.

8.3.6 Result

Result-luokkaa käytetään tulosten tallentamiseen aliohjelmassa. Serialisoitu Result-luokan olio serialisoidaan XML-serialisointia hyväksikäyttäen ja se välitetään pääohjelmalle named pipesin välityksellä. Result-luokka, kuvio 21, sisältää samanlaisen rakenteen kuin customer_routine_result-taulu tietokannassa.

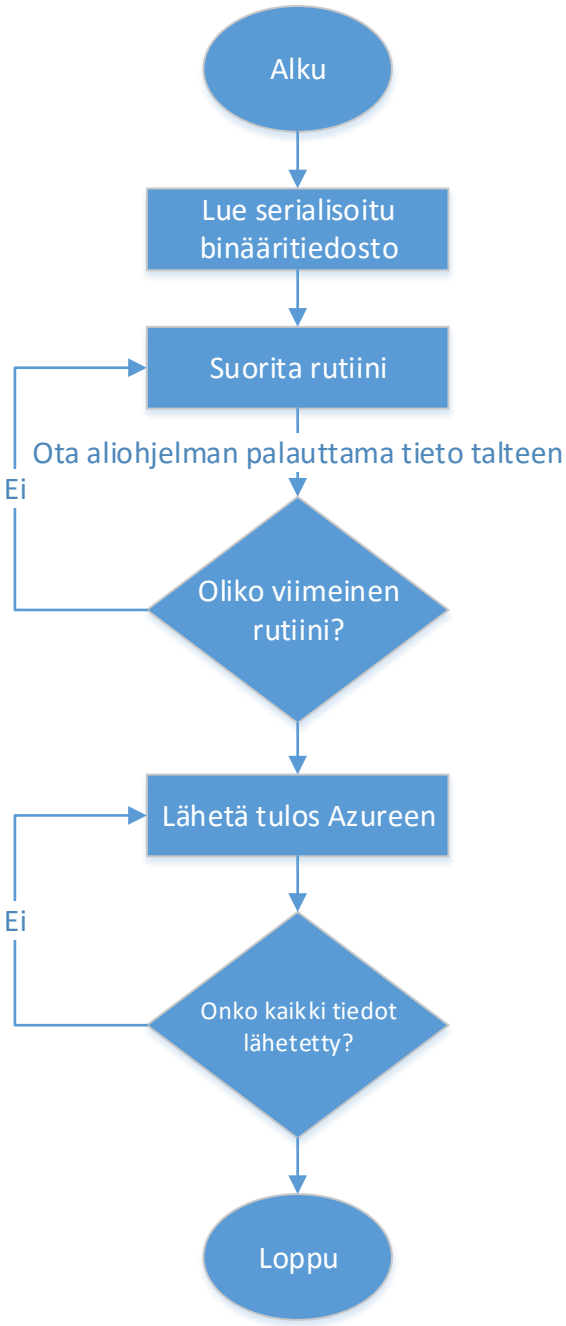


KUVIO 21. Result-luokkakaavio

Kun aliohjelma suorittaa toimenpiteitä, se ottaa talteen saadun datan Result-luokan olioon, intResult ottaa vastaan int-arvon. routine_exe_result-muuttujaan tallennetaan aliohjelman virhekoodi.

8.4 Pääohjelma

Pääohjelma on tämän opinnäytetyön ohjelmiston runko. Se suorittaa useita eri toimenpiteitä, kuten lukee asiakasympäristöön suoritettavat rutiinit, käynnistää aliohjelmat ja lähettää tulokset Azuressa sijaitsevaan tietokantaan. Pääohjelman kulku on kuvattu kuviossa 22.



KUVIO 22. Pääohjelman kulku

Kuviossa 22 kuvattu pääohjelman kulku ei sellaisenaan kerro kaikkea, joten on syytä käydä ohjelman runko tarkemmin. Ohjelman käynnistymisen jälkeen pääohjelma lukee serialisoidun binääritiedoston. Tämä tiedosto on valvontaohjelmiston luoma rutiinikokoelma, joka on määritelty asiakaskohtaisesti. Sieltä löytyvät tiedot, joita vaaditaan valvonnan suorittamiseen. Tiedosto tuodaan palvelimelle manuaalisesti, koska valvontarutiinit muut-

tuvat niin harvoin, ettei ole syytä hakea joka päivä uutta tiedostoa automaattisesti. Koska tiedosto on serialisoitu, sen sisältöä ei voida sellaisenaan näyttää luettavassa muodossa. Tiedostossa on N kappaletta rutiineja. Jokainen rutiini koostuu suoritettavan ohjelman nimestä, sille annettavista argumenteista ja tietokantaan viittaavasta avaimesta, tämä on kuviossa 20 kuvatun Routines-luokan olio serialisoituna.

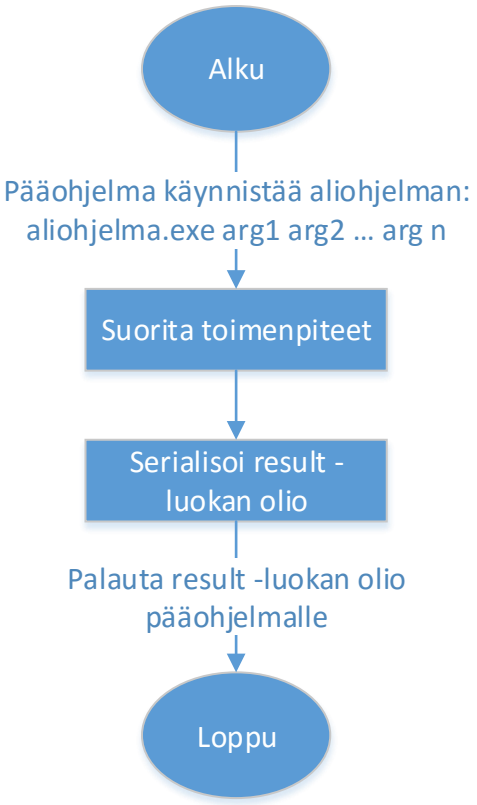
Kun pääohjelma on lukenut kaikki rutiinit, alkaa itse valvonnan suorittaminen. Pääohjelma käynnistää sisäisesti uuden prosessin rutiinitiedostosta tulleen ohjelman nimen perusteella ja odottaa aliohjelman palautusta ja päättymiskoodia. Tiedon välitys näiden prosessien välillä tapahtuu named-pipes -tekniikkaa hyödyntämällä. Näin pääohjelma pystyy lukemaan käynnistämänsä ohjelman tulostaman tekstin. Aliohjelma tulostaa kuviossa 21 olevan Result-luokan olion serialisoituna ja lisää sen Result-luokan olioita sisältävään listaan.

Lopuksi, kun kaikki aliohjelmat on suoritettu, käynnistyy tiedonsiirtoprosessi. Pääohjelma käy Result-luokan olioita sisältävän listan yksitellen läpi ja lähettää jokaisen tuloksen WCF-teknologiaa hyödyntäen Azuressa sijaitsevalle palvelimelle.

Koska pääohjelma lähettää valvontaan tarvittavat tiedot, on sen vikasietoisuuden oltava suuri. Tämän takia koko ohjelmisto käyttää try-catch-metodia virhesietoisuuteen ja virhetietojen tallennukseen asiakkaan palvelimen lokitiedostoon. Jos pääohjelman suoritus keskeytyy virheeseen tai jostain aliohjelmasta ei saada tulosta, se näkyy valvontaraportilla.

8.5 Aliohjelmat

Aliohjelmat ovat ohjelmia, jotka suoritetaan rutiinitiedoston määritysten mukaan. Nämä ohjelmat tekevät toimenpiteitä, jotka normaalissa valvonnassa suoritetaan manuaalisesti. Kaikki aliohjelmat käyttävät samaa perusrunkoa, joka näkyy kuviossa 23.



KUVIO 23. Aliohjelmien perustoiminnallisuus

Koska pääohjelman ei tarvitse tietää aliohjelman toiminnallisuudesta mitään ja riittää, että palautuksena tulee Result-luokan olio, pystytään aliohjelmia lisäämään tarpeen tullen lähes loputtomasti. Aliohjelmien toiminnallisuus on riippuvainen siitä, että pääohjelma tietää oikean sovellusnimen ja oikeat argumentit. Argumentit ovat ohjelmasta riippuvaisia, kuten levytilaa tarkistettaessa argumenttina annetaan tarkistettavan aseman nimi.

Seuraavassa esitellään jokainen tähän mennessä tehty aliohjelma. Aliohjelmista kerrotaan sen käyttötarkoitus, input- ja output-arvot ja muut tekniset ratkaisut. Jokainen ohjelma tallentaa virhetilanteissa -1 virhekoodin, jota ei ole erikseen mainittu ulostulossa. Jos ohjelman sisällä tapahtuu virhe, tallennetaan se myös lokitiedostoon, jotta manuaalisesti tarkisteltuna nähdään tarkempi virheseloste asiakkaan palvelimelta.

8.5.1 Vapaa tila

Vapaan tilan tarkistaminen on tärkeää monesta eri syystä. Jos asemalla, jolla tietokanta sijaitsee, ei ole vapaata tilaa, uusien rivien lisääminen tietokantaan ei onnistu. Tämä aiheuttaa virheellistä raportointia, ja jotkut tärkeät tiedot voivat olla menetetty kokonaan. Tärkeää on myös se, että varmuuskopiot sisältävällä asemalla on riittävästi tilaa, jotta mahdollisista virhetilanteista voidaan palautua.

TAULUKKO 6. Vapaan tilan tarkistavan ohjelman input/output

Input	Output
Aseman asematunnus	Vapaa tila tavuina

Kuten taulukosta 6 huomataan, ohjelma ottaa sisäänsä vain yhden arvon, jolla määritellään, minkä aseman vapaa tila katsotaan. Koska tietokoneessa voi olla eri käyttötarkoituksiin olevia kiintolevyjä, voivat niiden vapaan tilan tarpeet erota toisistaan. Tämän takia kaikkia kiintolevyjä ei tarkisteta yhdellä kertaa. Jos ohjelma ei löydä kyseistä kovalevyä, palautetaan siitä virheviesti. Läpi mennyt tarkistus palauttaa vapaan tilan määrän tavuina, josta se voidaan helposti muuttaa haluttuun muotoon.

8.5.2 Tiedoston koko

Yksi syy tiedoston koon seurantaan on esimerkiksi tietokantatiedoston koon historiointi. Toinen syy voi olla se, onko ZIP-tiedoston sisälle tullut tarvittavat tiedostot. Joissain tilanteissa ZIP-tiedosto luodaan ja sen jälkeen sinne siirretään tarvittavat tiedostot. Jos koko on 0 tavua, tiedetään, ettei sinne ole siirtynyt tarvittavia tiedostoja.

TAULUKKO 7. Tiedoston koon tarkistavan ohjelman input/output

Input	Output
Tiedoston sijainti	Tiedoston koko
	Tiedoston muokkauspäivämäärä

Vaikka ohjelma tarkistaa tiedoston koon, palauttaa se silti myös tiedoston muokkauspäivämäärän, kuten taulukossa 7 on kuvattu. Tämä on siitä syystä, että joskus tiedosto poistetaan vasta silloin, kun uudet tiedot luodaan. Muuten tiedoston koko voi olla oikea, mutta tiedot eivät ole silti uusia. Ohjelma palauttaa virheviestin, jos tarkistettavaa tiedostoa ei löydy.

8.5.3 Lokitiedoston tarkistus

Joissain tilanteissa lokeja kerätään tekstitiedostoihin. Tällaisissa tilanteissa tarvitsee lokitiedosto käydä läpi virheviestien takia.

TAULUKKO 8. Lokitiedoston tarkistavan ohjelman input/output

Input	Output
Tiedoston sijainti	Löytyikö virheitä
Etsittävät virheviestit, 1 - N kpl	

Taulukosta 8 selviää, että ohjelmisto ottaa input-parametreina tiedoston sijainnin ja etsittävät virheviestit. Eri ohjelmat tallentavat virheitä eri tavalla, kuten esimerkiksi Error, Error:, error ja niin edelleen. Näiden virheiden määrittely tehdään manuaalisesti, ettei ohjelman tarvitse käydä kymmeniä eri virheviestimahdollisuuksia läpi, mikä hidastaisi ohjelman suoritusta.

8.5.4 SQL Server -huoltosuunnitelma (Maintenance Plan)

SQL Serverissä monet toiminnot suoritetaan huoltosuunnitelmilla. Näihin kuuluvat varmuuskopiointi, vanhojen varmuuskopioiden poisto ja muita ajastettuja tehtäviä. Ajastettuihin tehtäviin voi kuulua joitain laskentoja,

jotka vaikuttavat raportointiin. Tämä aliohjelma tarkistaa, ovatko kaikki sellaiset huoltosuunnitelmat menneet läpi, joita ei ole poistettu käytöstä, ja onko sillä hetkellä käynnissä olevia huoltosuunnitelmia.

TAULUKKO 9. Huoltosuunnitelmien tarkistavan ohjelman input/output

Input	Output
Tietokantayhteyden nimi	Käynnissä olevan tai kaatuneen huoltosuunnitelman nimi

Huoltosuunnitelmien tarkistus on kaksivaiheinen. Tarkistetaan sellaiset ajot, jotka ovat kesken, ja sellaiset, jotka ovat päättyneet virhetilanteeseen. Ohjelma tarvitsee sisään tietokantayhteyden nimen (taulukko 9). Tämän nimen perusteella tietokantayhteys haetaan XML-tiedostosta, joka sisältää kaikki tietokantayhteydet. Ohjelma palauttaa kesken olevan huoltoajon nimen, jos sellainen löytyy. Jos joku huoltoajo on kaatunut, palautetaan sen nimi.

8.5.5 OLAP

OLAP-kuutiot toimivat esimerkiksi Reporting Services -raporttien tietolähteenä. Raportteja käytetään Business Intelligence -analysointiin. Tämän vuoksi on tärkeää, että asiakkaalla on tiedossa, jos kuutioiden prosessointi ei ole mennyt läpi kuluvana päivänä. Prosessoinnilla tarkoitetaan tapahtumaa, joka laskee eri laskutoimitukset ja mahdollistaa datan yhdistämisen nopeasti saataville.

TAULUKKO 10. OLAP-tarkistuksen tekevän ohjelman input/output

Input	Output
OLAP yhteyden nimi	Kuution prosessointipäivämäärä ja aika
OLAP kuution nimi	Viimeisimmän skeemapäivityksen aika ja päivämäärä

OLAP-tarkistus ottaa sisäänsä yhteyden nimen ja OLAP-kuution nimen, kuten taulukosta 10 selviää. OLAP-yhteyden nimi haetaan kantayhteydet sisältävästä tiedostosta OLAP-yhteyden nimellä. OLAP-kuution nimen perusteella haetaan tarvittavat tiedot, kuten prosessointipäivämäärä ja skeemapäiväys. Skeemapäiväys voi auttaa selvittämään vikaa, jos skeemaan on tehty muutos ja ongelmat ovat alkaneet vasta sen jälkeen.

8.5.6 Tapahtumienvalvonta

Tapahtumienvalvonta on Windows-käyttöjärjestelmän sisäänrakennettu työkalu, johon kirjautuu järjestelmä- ja ohjelmistovirheitä. Sen tarkistus on tärkeää, jos halutaan varmistaa, etteivät SQL Serveriin liittyvät komponentit ole kaatuneet.

TAULUKKO 11. Tapahtumienvalvonnan suorittavan ohjelman input/output

Input	Output
	Virheen aika ja aiheuttaja

Kuten taulukosta 11 huomataan, ohjelma ei ota sisään yhtään argumenttia. Tämä johtuu siitä, että mahdolliset virheet ovat samalla tietokoneella, jossa aliohjelma suoritetaan. Ohjelma käy läpi kaikki virheet viimeisen vuorokauden ajalta. Mahdolliset suodatukset voidaan tehdä valvontaraportilla.

8.5.7 SQL-kyselyn suorittaja

Monet tarkistukset tehdään SQL Server -tietokannoista. Tämän aliohjelman avulla voidaan suorittaa kysely SQL-tietokantaan. Normaalisti tällaisen ohjelman teko olisi vastoin suosituksia, koska sillä voisi tehdä tuhoavia toimenpiteitä, kuten poistaa koko tietokanta.

TAULUKKO 12. SQL-kyselyn suorittavan ohjelman input/output

Input	Output
Tietokantayhteyden nimi	Kyselyn tulos
SQL-kysely	

Ohjelma ottaa sisään muistakin aliohjelmista tutun tietokantayhteyden nimen ja SQL-kyselyn (taulukko 12). Ohjelma palauttaa kyselyn tuloksen tekstimuodossa, joka voidaan parsia raportilla haluttuun muotoon.

8.5.8 SSIS-paketin valmistuminen

Microsoftin teknologioita hyödyntävillä asiakkailla on käytössä SSIS-paketteja, jotka suorittavat ETL-toimintoja ja muita toimintoja. Tämän ohjelman avulla voidaan varmistaa, onko joku tietty SSIS-paketti suoritettu onnistuneesti. Tietokantaan tallentuu tapahtumalokiin tietoa, josta saadaan ajon alkamisaika ja loppumisaika.

TAULUKKO 13. SSIS-paketin valmistumista seuraavan ohjelman input/output

Input	Output
Tietokantayhteyden nimi	Paketin tila
SSIS paketin nimi	Kesto
	Päättymisaika

Kuten taulukosta 13 huomataan, aliohjelma käynnistetään antamalla argumenteiksi tietokantayhteyden nimi ja SSIS-paketin nimi. Aliohjelma palauttaa paketin tilan, eli sen, onko paketti ajettu vai ajossa. Ohjelman suorituksen kesto sekunteina palautetaan aina, riippumatta siitä, onko ohjelma jo suoritettu. Jos ohjelma on suoritettu läpi, palautetaan sen päättymisaika sille varatussa muuttujassa.

8.5.9 SYSSSISlogin tarkistus

SYSSSISlog-tarkistus on yksi tärkeimmistä tarkistuksista, koska kaikki SSIS-paketit tallentavat ajonaikaiset tapahtumat sinne. Jos SYSSSISlog-taulusta löytyy virhe, aiheuttaa se aina selvitystoimenpiteitä.

TAULUKKO 14. SYSSSISlog-tarkistuksen suorittavan ohjelman input/output

Input	Output
Tietokantayhteyden nimi	Kaatuneiden pakettien nimet
Monen päivän takaa virheitä etsitään	Viimeisimmän virherivin aikaleima

Koska SYSSSISlog-tauluja on aina yksi tietokantaa kohden, ei ohjelma tarvitse erikoisia argumentteja, kuten taulukosta 14 huomataan. Yleensä virheitä etsitään vain viimeisen 24 tunnin ajalta, mutta ohjelmaan on jätetty mahdollisuus etsiä virheitä pidemmältäkin aikajaksolta. Aliohjelma palauttaa kaatuneiden pakettien nimet, jos sellaisia löytyy. Jos virheitä löytyy, tallennetaan myös viimeisimmän virheen aikaleima. Virheviestejä ei lähetetä asiakkaan palvelimelta eteenpäin, koska joskus virheviestit sisältävät epäonnistuneen Insert-lauseen, joka saattaa sisältää HR-dataa.

9 YHTEENVETO

Tavoitteena oli saada automatisoitu ohjelmistokokonaisuus manuaalisesti suoritettavan päivittäisvalvonnan tilalle. Tämän ohjelmistokokonaisuuden tarkoituksena oli koota kaikkien asiakkaiden tietojärjestelmien tilat yhteen hallittuun raporttiin, josta selviää yhdellä silmäyksellä kaikkien asiakkaiden mahdolliset virhetilanteet. Koska kerättyä dataa kerätään tietokantaan, tämä mahdollistaa myös tiedon hyödyntämisen Business Intelligencen tapaan. Asiakaskohtaiset virhetiedot voidaan esittää graafisesti, jolloin on helpompi näyttää asiakkaalle latausketjujen heikot kohdat. Kokonaisuudessaan projektille asetetut tavoitteet täyttyivät onnistuneesti. Ohjelmiston ja tietokannan hyvä suunnittelu jätti tilaa jatkokehitykselle.

Valvontasovelluksen toteutus jakautui kahteen osaan: suunnittelu ja toteutus. Suunnitteluvaihe oli tärkeä, jotta monipuolinen laajennettavuus olisi mahdollista. Suunnittelu aloitettiin vuoden 2014 kesällä muiden töiden ohessa. Toteutus alkoi 2014 syksyllä kesän aikana laadittujen spesifikaatioiden perusteella. Asiakasohjelmiston esiversioiden osalta testaus alkoi toteutuksen ohessa. Ohjelmistoon kuuluu niin paljon eri tekniikoita, että uuden teknologian implementointi vaati aina perusteellisen testauksen. Kokonaisuuden testaus palvelimella alkoi vuoden 2015 keväällä. Tällöin ohjelmisto sisälsi tärkeimmät toiminnallisuudet ja ohjelmisto toimi vikasietoisesti.

Asiakasohjelmisto suoritetaan ajastetusti asiakkaan ympäristössä, joten paras ratkaisu oli toteuttaa ohjelmisto konsolisovelluksena. Konsolisovellus vie vähemmän resursseja kuin Windows Forms -toteutus. Koska suunnittelu oli Microsoftin teknologioiden ympärillä, looginen valinta ohjelmointikieleksi oli C#. Ohjelmiston kehitys tapahtui Microsoft Visual Studio -kehitystyökaluilla. Ohjelmiston versionhallinta oli Microsoft Team Foundation Server. Tietokantana toimii Microsoft SQL Server 2012, joka asennettiin Microsoft Azure-virtuaalipalvelimelle. Asiakasohjelmisto lähettää tiedot tietokantaan Windows Communication Foundationin avulla.

Vaikka ohjelmisto on sellaisella tasolla, että sillä voidaan suorittaa päivittäisvalvontaan liittyviä toimintoja, on sen kehitys vielä kesken. Toiminnallisuuksia lisätään tarpeen tullen ja olemassa olevia sovelluksia kehitetään parhaan suorituskyvyn ja luotettavuuden saavuttamiseksi. Tavoitteena on saada ohjelmistoon ennakoivaa analytiikkaa asiakkaiden tietojen perusteella.

LÄHTEET

Boettger, A., Cierkowski, M., Leiter, C. & Wood, D. 2009. Beginning Microsoft® SQL Server® 2008 Administration. New York, United States: Wiley Publishing, Inc..

Bowen, C., Crane, R. & Resnick, S. 2008. Essential Windows Communication Foundation. Boston, United States: Addison-Wesley.

Chauhan, A., Fontama, V., Hart, M., Tok, W. H. & Woody, B. 2014. Introducing Windows Azure Hdinsight. Redmond, United States: Microsoft Press.

Esposito, D. 2003. Applied XML Programming for Microsoft .NET. Redmond, United States: Microsoft Press.

Ferrari, A., Russo, M. & Webb, C. 2012. Microsoft SQL Server 2012 Analysis Services. Califoria, United States: O'Reilly Media, Inc.

Machado, R. 2014. SSIS Succinctly. North Carolina, United States: Syncfusion Inc.

Rad, R. 2014. Microsoft SQL Server 2014 Business Intelligence Development. Birmingham, United Kingdom: Packt Publishing.

Thai, T. L. & Lam, H. 2002. .NET Framework Essentials. 2nd Edition. Califoria, United States: O'Reilly Media, Inc.