TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

**SANTTU SEPPÄNEN**

# TIEDON KUVAUS JA SARJALLISTAMINEN PALVELUKESKEISESSÄ TIETOJÄRJESTELMÄSSÄ

Diplomityö

Tarkastaja: professori
Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston
kokouksessa 4. kesäkuuta 2014

# TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
**SEPPÄNEN, SANTTU**: **Tiedon kuvaus ja sarjallistaminen palvelukeskeisessä tietojärjestelmässä**
Diplomityö, 58 sivua
Tammikuu 2015
Pääaine: Hajautetut ohjelmistot
Tarkastaja: professori Tommi Mikkonen
Avainsanat: Tiedon sarjallistaminen, ORM, SOA, WCF, Entity Framework

Toiminnanohjausjärjestelmien tarkoitus on yhdistää organisaation tietovirtoja yhden tietojärjestelmän alle ja palvella montaa käyttäjäryhmää. Samanaikaisesti organisaatiot tavoittelevat tehokkaampaa resurssien hyödyntämistä ja parempaa tiedonkulkua organisaation elimien välillä. Toiminnanohjausjärjestelmä voidaan toteuttaa palvelukeskeisenä tietojärjestelmänä. Palvelupohjaisuus onkin ollut ohjaavana suunnittelulähtökohtana Suomessa terveydenhuollon tietojärjestelmissä.

Organisaation laajuisella toiminnanohjausjärjestelmällä voi olla useita käyttäjiä yhtä aikaa. Tällöin on välttämätöntä, että järjestelmä reagoi nopeasti muutoksiin ja tiedottaa muutoksista käyttäjiä. Yksi suurimmista suorituskykyyn vaikuttavista tekijöistä suurissa tietojärjestelmissä on tiedon haku, käsittely ja kuljetus verkon yli järjestelmän eri komponenteille. Tiedon välityksen toteutukselle tietokannasta aina käyttäjälle asti asetetaan kovia tehokkuushaasteita.

Tässä työssä selvitettiin palvelupohjaisen tietojärjestelmän ratkaisuvaihtoehtoja suorituskykyisen tiedonkulun toteuttamiselle. Työn tarkastelun kohteena toimi Turun PET-keskukselle tehtävä ERP-järjestelmä. Järjestelmä voidaan jakaa kokonaisuuksiin kerrosten pohjalta. Käyttöliittymäkerroksen kattava asiakassovellus on hajautettu palvelu- ja tietokantakerroksesta tiedonsiirtokerroksen avulla. Työssä ratkaisuksi tarjottiin teknologia- ja suunnitteluratkaisukokonaisuutta, jonka muodostavat .NET-sovelluskehyksen tarjoamat komponentit. Käyttöliittymäkerros toteutettiin graafisella rajapintakerroksella WPF ja MVVM-suunnitteluratkaisun avulla. Palvelu- ja tiedonsiirtokerros toteutettiin WCF-ohjelmistokehyksen avulla ja tietokantakerros Entity Framework -sovelluskehyksellä. Tiedonsiirto perustui päätepisteisiin ja kaksisuuntaiseen tiedonkulkuun suorituskykyistä TCP-protokollaa käyttäen. Tietokanta luotiin olio-relaatio-kuvauksella Entity Frameworkin käsitteellisestä mallista.

Työssä esitelty ratkaisukokonaisuus sopii suorituskykytestien ja arvioinnin pohjalta hyvin PET ERP -järjestelmän toteutukseen. Ratkaisua voidaan käyttää myös yleisesti palvelupohjaisten tietojärjestelmien toteutuksen perustana. Käytetyt sovelluskehykset ja komponentit ovat hyvin muunneltavia ja yleiskäyttöisiä.

# ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
**SEPPÄNEN, SANTTU**: **Data Mapping and Serialization in a Service-Oriented Information System**
Master of Science Thesis, 58 pages
January 2015
Major: Distributed Software
Examiner: Professor Tommi Mikkonen
Keywords: Data serialization, ORM, SOA, WCF, Entity Framework

The purpose of enterprise resource planning systems is to combine organizations data flows and serve different kind of user groups. At the same time organizations seek to improve their use of resources and communication between organization's different parts. Service-oriented data system is one of the methods to develop an enterprise resource planning system. Service-Orientation is actually a leading designing pattern for health care data systems in Finland.

Organization wide enterprise resource planning system can have multiple users at the same time. It is vital for the system to react changes rapidly and inform users at once. In these data systems one of the biggest performance factors is querying, handling and transferring of data. Implementing data transfer from database to the user is challenging at the performance point of view.

This master's thesis describes ways to implement server-oriented data systems with effective data transmission. This thesis examines an ERP-system developed to Turku PET-center. The system can be divided to layers based on their purpose. Presentation layer covering the client application is distributed from service and data access layers with data transfer layer. Solution for this challenge in this thesis is design patterns and components provided by .NET Framework. Presentation layer is implemented with graphical subsystem WPF and MVVM design pattern. Service layer and data transfer are implemented with WCF framework and data access layer uses Entity Framework. Data transfer is based on WCF endpoints with high performance TCP protocol and duplex communication. Database is created from Entity Frameworks conceptual model using object-relational mapping.

Solution described in this thesis fits well in the implementation of PET ERP -system based on performance comparisons and evaluations. Provided solution is also suitable for the implementation of server-oriented data systems in general. Frameworks and components used in this work are flexible and interoperable.

# ALKUSANAT

Ensimmäisenä haluan kiittää työni tarkastajaa professori Tommi Mikkosta ja ohjaajaa Risto Pitkästä laadukkaista neuvoista ja asiantuntevasta ohjauksesta kirjoitustyön aikana. Kiitän Atostek Oy:n työntekijöitä ja johtoa mahtavasta työilmapiiristä ja joustamisesta diplomityön tekemisessä. Tämän diplomityöaiheen on tarjonnut ja rahoittanut Atostek Oy. Haluan kiittää myös ystäviäni, isääni ja äitiäni tuesta pitkällä matkallani kohti valmistumista.

Lopuksi vielä kiitos opiskelutovereilleni, eräälle urrrheilujoukkueelle ja yliopiston yhteisölle mahtavista ja opettavaisista vuosista Tampereen teknillisessä yliopistossa. On paljon asioita mitä ei opi pelkästään luentosaleissa istumalla.

Tampereella 17.12.2014

Santtu Seppänen

# SISÄLLYS

# TERMIT JA NIIDEN MÄÄRITELMÄT

| | |
|---|---|
| .NET-sovelluskehys | Microsoftin kehittämä sovelluskehys ohjelmistojen kehittämiseen (.NET Framework). |
| ADO.NET | Tarjoaa rajapinnan, jolla tietokannan tietoja voidaan käsitellä verkon yli (ActiveX Data Objects). |
| ASP.NET | .NET:n web-ohjelmistokehys web-pohjaisten sivustojen, ohjelmien tai palveluiden tekoon. |
| CLR | Microsoft:n .NET ohjelmistokomponenttikirjaston virtuaalikoneen osa, joka vastaa .NET ohjelmien hallinnasta ja ajamisesta (Common language runtime). |
| DAO | Relaatiotietokannan käsittelyyn tarkoitettu abstrakti tiedonhakuolio (Data Access Object). |
| DataSet | Tietokannan valituista tauluista koostettu muistissa oleva olio, johon voidaan kohdentaa toimenpiteitä ja päivittää muutokset takaisin tietokantaan. |
| DTO | Tiedonsiirto-olio eli tietorakenne jossa tietoa kuljetetaan verkon yli, ja johon tieto on pakattu helpottamaan tiedonsiirtoa (Data Transfer Object). |
| EDM | Käsitteellinen entiteettitietomalli vastaa tietokannan skeemamallia. Entity Framework:ssa tiedot käsitellään tämän mallin avulla entiteettinä (Entity Data Model). |
| Entiteetti | Relaatiomallin tai entiteettimallin tietoa edustava abstrakti luokka tai olio (Entity). |
| Ensemble-palveluväylä | PET-keskuksen ja sairaanhoitopiirin tietojärjestelmien väliseen kommunikointiin tarkoitettu palveluväylä. |
| ERP-järjestelmä | Tietojärjestelmä, jonka toiminnallisuus kattaa organisaation toiminnan kaikki osa-alueet (Enterprise Resource Planning System). |
| GMP | Hyvien tuotantotapojen standardi (Good Manufacturing Practises). |
| HTTP | Protokolla asiakkaan ja palvelimen keskinäiseen tiedonsiirtoon (Hypertext Transfer Protokol). |
| IIS | Microsoftin kehittämä palvelinohjelmistokokonaisuus Windows-pohjaisille palvelimille (Internet Information Services). |
| JSON | Yksinkertainen, selkokielinen tiedonsiirtoformaatti (JavaScript Object Notation). |
| LIMS | Laboratorion tiedonhallintajärjestelmä (Laboratory Information Management System). |
| LINQ | Kyselykieli ohjelmallisen tiedon hakuun tietolähteestä (Language Integrated Query). |

| | |
|---|---|
| MSIL | Microsoft .NET-ohjelmistokehyksen kääntäjän tuottama välikieli (Microsoft Intermediate Language). |
| ORM | Oliorelaatikuvaus, joka mahdollistaa relaatiotietokannan tietomallin datan esittämisen olioina ja oliomallina (Object Relational Mapping). |
| PET | Positroniemissiotomografia on kuvantamismenetelmä, jossa käytetään radioaktiivista merkkiainetta (Positron emission tomography). |
| PET ERP | Turun PET-keskukselle toteutettavan toiminnanohjausjärjestelmän nimi. |
| POCO | Käsin luotava sovelluskehys-riippumaton ja puhdas luokka entiteettien luomista varten (Plain old CLR object). |
| Sitominen | Käyttöliittymäkomponentin sisällön yhdistäminen käsittelijään (Binding). |
| SOA | Palvelukeskeinen ohjelmistoarkkitehtuuri (Server-Oriented Architecture). |
| SOAP | Palvelukeskeinen ja yksinkertainen protokolla viestin vaihtoon ja funktiokutsuihin ohjelmien välillä verkon yli. (Single Object Access Protocol). |
| Sovelluspalvelu | Verkkopalvelimella toimiva ohjelma, joka tarjoaa standardoitujen internetyhteyskäytäntöjen(protokollien) avulla palveluja sovellusten käytettäväksi (Web Service). |
| SQL | Standardoitu relaatiotietokannan kyselykieli (Structured Query Language). |
| T4-malli | Entity Frameworkin entiteettien automaattiseen luontii tarkoitettu malli (T4 Template). |
| TCP | Tietoliikenneprotokolla yhteyden luomiselle kahden systeemin välille (Transmission Control Protocol). |
| UDDI | Alustariippumaton ja avoin palvelurekisteristandardi sovelluspalveluiden hakuun (Universal Description Discovery and Integration). |
| WCF | Microsoft .NET –pohjainen teknologia, joka sisältää joukon ohjelmointirajapintoja ja tarjoaa ajoympäristön ja sisäisen viestintäjärjestelmän Windows-pohjaisten sovellusten kehittämiseen (Windows Communication Foundation). |
| WPF | Graafinen .NET-sovelluskehyksen ohjelmistokomponentti käyttöliittymien tekoon (Windows Presentation Foundation). |
| WSDL | Palvelukeskeisten sovelluspalvelujen rajapintojen ja toimintojen kuvauskieli (Web Service Description Language). |

| | |
|---|---|
| XML | Standardoitu merkintäkieli, jota voidaan käyttää formaattina tiedonvälitykseen tai dokumenttien tallentamiseen (Extensible Markup Language). |

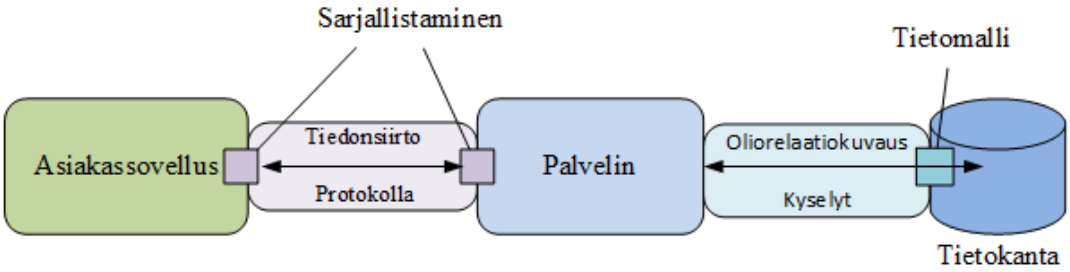
# 1 JOHDANTO

Erilaiset toiminnanohjausjärjestelmät (ERP-järjestelmä, Enterprice Resource Planning System) ovat yleisiä isoissa ja keskisuurissa organisaatioissa. Toiminnanohjausjärjestelmällä tavoitellaan muun muassa parempaa organisaation resurssien hyödyntämistä sekä tiedon kulkua. Tieto on keskitettynä yhteen paikkaan kaikkien saataville, jolloin järjestelmän toteutuksessa on otettava huomioon erilaisia vaatimuksia.

Palvelukeskeisyys on yksi suunnittelulähtökohta toiminnanohjausjärjestelmän toteutukselle. Palvelukeskeisessä tietojärjestelmässä tieto voi olla keskitettynä palvelimella sijaitsevaan tietokantaan ja järjestelmä tarjoaa avoimia palvelurajapintoja tiedon hyödyntämiseen asiakassovelluksissa. Asiakassovellus ottaa käynnistyessään yhteyden palvelimelle ja rekisteröityy palveluun. Kaikki toiminnallisuus perustuu pyyntöihin ja vastauksiin. Tällaisella tietojärjestelmällä voi olla useita käyttäjiä yhtä aikaa ja muutoksista tiedottamisen pitäisi olla reaaliaikaista. Verkon yli suuria tietomääriä käsiteltäessä tiedon muoto ja tietokantahakujen nopeus ovat ratkaisevassa roolissa. Tiedon välityksen toteutukselle tietokannasta aina käyttäjälle asti asetetaan kovia tehokkuushaasteita.

Tässä diplomityössä toteutettiin palvelukeskeisen toiminnanohjausjärjestelmän arkkitehtuuri suunnitteluratkaisuineen ja esitetään ratkaisuja edellä mainittujen haasteiden pohjalta asiakassovelluksen, palvelimen ja niiden välisen tiedonkulun toteuttamiselle. Tarkastelun kohteena käytetään Turun PET-keskukselle toteutettua ERP-järjestelmää, joka tehtiin helpottamaan potilaiden, laitteiden, aikataulujen, projektien, raportoinnin ja talouden hallintaa. Toiminnanohjausjärjestelmä toteutetaan Atostek Oy:n toimesta.

Työ rajattiin käsittelemään asiakassovellusta, palvelinta, niiden välistä tiedonsiirtoa ja tiedon sarjallistamista ja kuvausta. Toteutusteknologiaksi näihin järjestelmän osiin valikoitiin Microsoft:n .NET-sovelluskehys (.NET Framework) ja tätä voidaan pitää työssä lähtökohtana arkkitehtuurien vertailussa. Asiakassovelluksen kommunikointi perustuu pyyntöihin palvelupohjaiselle palvelimelle. Olio-relaatio-kuvauksen avulla tietokannan tietoja voidaan hakea oliomalliin kohdistetuilla kyselyillä. Palvelu sarjallistaa haetun tiedon optimaaliseen muotoon ja lähetetään verkon yli protokollan avulla. Asiakassovellus takaisin sarjallistaa viestin ja haettu tieto voidaan esittää käyttäjälle oliomallin olioina. Koko operaation tiedonkulku on esitetty kuvassa 1.1. Tiedon esitystavalla ja kuvauksella tarkoitetaan tässä työssä tietokannan tiedon esittämistä järjestelmässä oliorelaatiokuvauksen (ORM, Object Relational Mapping) avulla, ja formaatilla tarkoitetaan tiedon muotoa kuljetuksen aikana.




**Kuva 1.1. Asiakassovelluksen, palvelimen ja tietokannan väliset toiminnot**

PET ERP -järjestelmä ja sen toimintaympäristö on kuvattu tarkemmin luvussa 2. Aluksi esitellään toiminnanohjausjärjestelmän teoriaa, jotta saadaan kokonaiskuva toteutettavasta järjestelmästä. Tässä luvussa esitellään myös tärkeimmät vaatimukset ja reunaehdot järjestelmälle, ja rajataan työssä käsiteltävä järjestelmää.

Järjestelmän arkkitehtuuri esitellään luvussa 3. Järjestelmä jakautuu käyttöliittymä- tiedonsiirto-, palvelu- ja tietokantakerroksiin. Kerrosten arkkitehtuuri- ja teknologiavaihtoehtoja vertaillaan, jonka jälkeen valitaan ratkaisut asiakassovelluksen, palvelimen ja niiden välisen kommunikaation toteutukselle. Arkkitehtuurivalinnoissa on otettu huomioon järjestelmän reunaehdot ja vaatimukset, sekä myös muita, esimerkiksi toimittajasta ja asiakkaasta johtuvia perusteita.

Luvussa 4 käsitellään tiedon kuvaus- ja sarjallistamisratkaisuja, perehdytään oliorelaatio-kuvaukseen ja tutkitaan .NET-sovelluskehyksen tarjoamia vaihtoehtoja oliomallinnukseen. Tiedonsiirtokerroksen yksityiskohtaisia suunnitteluratkaisuja tutkitaan sarjallistamisen yhteydessä. Tietokantakerrosta käsiteltäessä esitellään vaihtoehtoja tietokantahakujen tekemiselle, tietokannan kuvaamiselle ja sovelluskehyksen palveluiden hyödyntämiselle. Erilaisia suunnitteluratkaisuja tarkastellaan tehtyjen järjestelmän arkkitehtuuri- ja teknologiavalintojen perusteelta.

Luvussa 5 esiteltyjä vaihtoehtoja vertaillaan keskenään ja arvioidaan valittujen ratkaisujen toimivuutta toteutetussa järjestelmän osassa. Asiakassovelluksen, palvelimen ja niiden välisen tiedonsiirron suunnitteluratkaisuja ja tiedon sarjallistamis- ja kuvausratkaisuja arvioidaan suorituskyvyn, muunneltavuuden, yleiskäyttöisyyden ja toteutuksen helppouden ja nopeuden kannalta. Lopuksi esitellään jatkokehitysideoita, joilla järjestelmästä voisi saada edelleen tehokkaamman annettujen kriteerien valossa.

Luvussa 6 tehdään yhteenveto valituista ratkaisuista ja arvioidaan, kuinka valitut ratkaisut vastasivat työssä asetettuun tutkimusongelmaan. Seuraavaksi todetaan esitetyn ratkaisun soveltuvuus toteutettuun PET ERP -järjestelmään. Pohditaan myös valitun kokonaisuuden yleiskäyttöisyyttä palvelupohjaisten järjestelmien toteutuksessa.

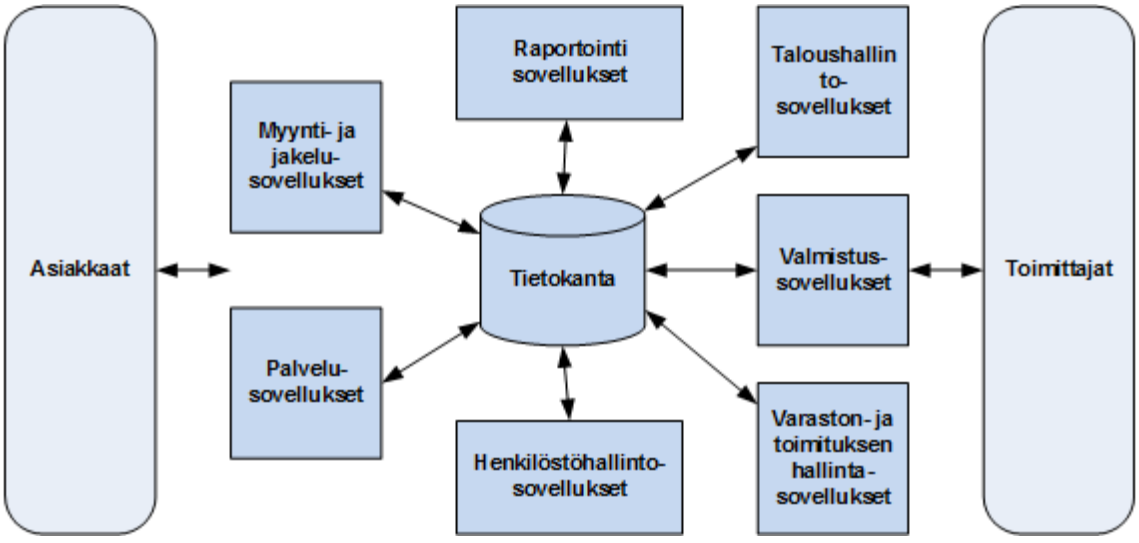
# 2 JÄRJESTELMÄN KUVAUS

Tässä luvussa kuvataan toteutettu toiminnanohjausjärjestelmä, ympäristö, jossa sitä käytetään ja liitännät muihin järjestelmiin. Kohdassa 2.1 esitellään toiminnanohjausjärjestelmän käsite ja perusperiaatteita. Kohta perustuu lähteeseen [1], jos ei toisin mainita. Kohdassa 2.2 kuvataan lääketieteellinen toimintaympäristö, Turun PET-keskus ja järjestelmälle asetetut vaatimukset. Ellei toisin mainita, tässä kohdassa käytetään lähteenä asiakkaan tarjouspyyntöä ja sen liitteitä [2]. Toimeksiantoon johtaneen julkisen tarjouskilpailun dokumentit eivät kuitenkaan ole enää yleisesti saatavilla. Kohdassa 2.3 kuvataan mitä laitteita ja muita järjestelmiä PET-keskuksen yhteydessä toimii ja mihin kaikkiin järjestelmiin PET ERP integroidaan.

## 2.1 Toiminnanohjausjärjestelmä

Isot organisaatiot keräävät, tuottavat ja tallentavat valtavan määrän tietoa. Monissa näistä tietoa ei kuitenkaan tallenneta yhteen tietokantaan, vaan se on pirstoutunut ympäri organisaation tai yksiköiden erilaisia järjestelmiä. Jos myynnin ja tuotannon järjestelmät eivät viesti keskenään, niin tuottavuus ja asiakkaan tilauksiin reagointi kärsivät. Tiedonvaihto eri yksiköiden välillä vaikeutuu ja liiketoiminta sirpaloituu.

Toiminnanohjausjärjestelmän tarkoitus on yhdistää kaikkien yrityksen päätoimintojen, kuten laskutuksen, kirjanpidon, henkilöstöresurssien, toimitusketjujen, tilausten ja asiakkuudenhallinnan tietovirrat yhden tietojärjestelmän alle. Oleellinen tieto tallennetaan keskitettyyn toiminnanohjausjärjestelmän tietokantaan. Kuva 2.1 havainnollistaa toiminnanohjausjärjestelmän käsitettä.




**Kuva 2.1. ERP-järjestelmän käsite.**

Kuvan toiminnanohjausjärjestelmän asiakkaina toimivat muiden järjestelmää käyttävien organisaatioiden henkilökunta ja potilaat. Toimittajina toimivat osaltaan PET-keskuksen henkilökunta, joka vastaa tutkimusten teosta ja lääkeainevalmistuksesta.

Tällainen järjestelmä on eri tekniikoilla toteutettujen komponenttien monimutkainen kokonaisuus, jonka suunnittelussa tulee ottaa huomioon organisaation kaikki siihen integroitavat järjestelmät. Järjestelmän kehittämiseen ja asennukseen täytyy usein investoida paljon rahaa, aikaa ja ammattitaitoa. Suurena haasteena on myös järjestelmän sovittaminen osaksi organisaatiota. Epäonnistumisen välttämiseksi toiminnanohjausjärjestelmän teknologiset pakotteet on saatava vastaamaan liiketoiminnan tarpeita.

Perinteisesti yritysten tietojärjestelmien kehitys tapahtui niin, että valittiin ensin tapa tehdä liiketoimintaa ja sitten ohjelmisto, joka vastasi parhaiten toiminnan tarpeita. Toiminnanohjausjärjestelmän osalta kehitys on päinvastainen. Liiketoimintaa tarvitsee usein muuttaa järjestelmää varten, sillä toiminnanohjausjärjestelmä on loppujen lopuksi yleispätevä ratkaisu.

Vaativien toiminnanohjausjärjestelmien toteuttamiseen tarvitaan laskentatehoa, korkean saatavuuden (high availability) järjestelmiä ja isoja, nopeita verkkopalvelimia ja tietovarastoja. Näiden vaatimusten vallitessa ERP-järjestelmistä on tullut hajautettuja ja verkon yli saatavia sovelluksia, joiden pitää pystyä palvelemaan montaa käyttäjää yhtä aikaa ja käsittelemään suurta määrä tietoa. Lisäksi järjestelmän pitää olla joustava ja skaalautuva, toteuttaa tietoturvamekanismeja luvatonta pääsyä ehkäisemään ja kyetä tekemään monimutkaisia ja ehyitä transaktioita tietokantaan.[3]

Toiminnanohjausjärjestelmien käytettävyydelle on välttämätöntä, että sovelluksen yleinen toiminnallisuus on nopeaa ja tehokasta. Yksi suurimmista nopeuteen ja tehokkuuteen vaikuttavista tekijöistä suurissa järjestelmissä on tiedon käsittely ja kuljetus järjestelmän eri komponenteille. Tällaisissa hajautetuissa järjestelmissä toiminnallisuus jakautuu yleensä asiakas- ja palvelinsovelluksiin, joiden yhteydenpito perustuu tietoverkkoon. Näitä vaatimuksia myös tässä työssä kuvatun ERP-järjestelmän tulee noudattaa, vaikka toimintaympäristö on poikkeuksellinen ja organisaatio voittoa tavoittelematon terveydenhuollon palveluntarjoaja.

## 2.2 Toimintaympäristö

Terveydenhuollon toimintaympäristö on tyypillisesti moniammatillinen ja saattaa koostua erilaisista organisaatioista erilaisine hoitoineen ja tutkimuksineen. Terveydenhuollossa syntyy paljon erilaista tietoa ja käytössä olevat järjestelmät ovat yleensä hyvin erilaisia. Kaikkien toimintojen ja laitteiden liittäminen osaksi isoa tietojärjestelmää tuo haasteta järjestelmän suunnitteluun. Järjestelmän muita tavoitteita ovat muun muassa käyttäjäystävällisyys ja olemassa olevan ja tuotettavan tiedon ja toimintojen päällekkäisyyksien karsiminen ja ehkäisy. Terveydenhuollon tietosuoja- ja tietoturvavaatimukset

asettavat lisävaatimuksia, erityisesti kun potilastiedon täytyy liikkua järjestelmästä toiseen.[4]

Alakohdassa 2.2.1 esitellään Turun PET-keskusta ja toimitettavaa järjestelmään. Alakohdassa 2.2.2 käydään läpi toteutettavan järjestelmän vaatimukset ja reunaehdot ja 2.2.3 alakohdassa liitännät muihin järjestelmiin.

### 2.2.1 Turku PET Centre ja PET ERP

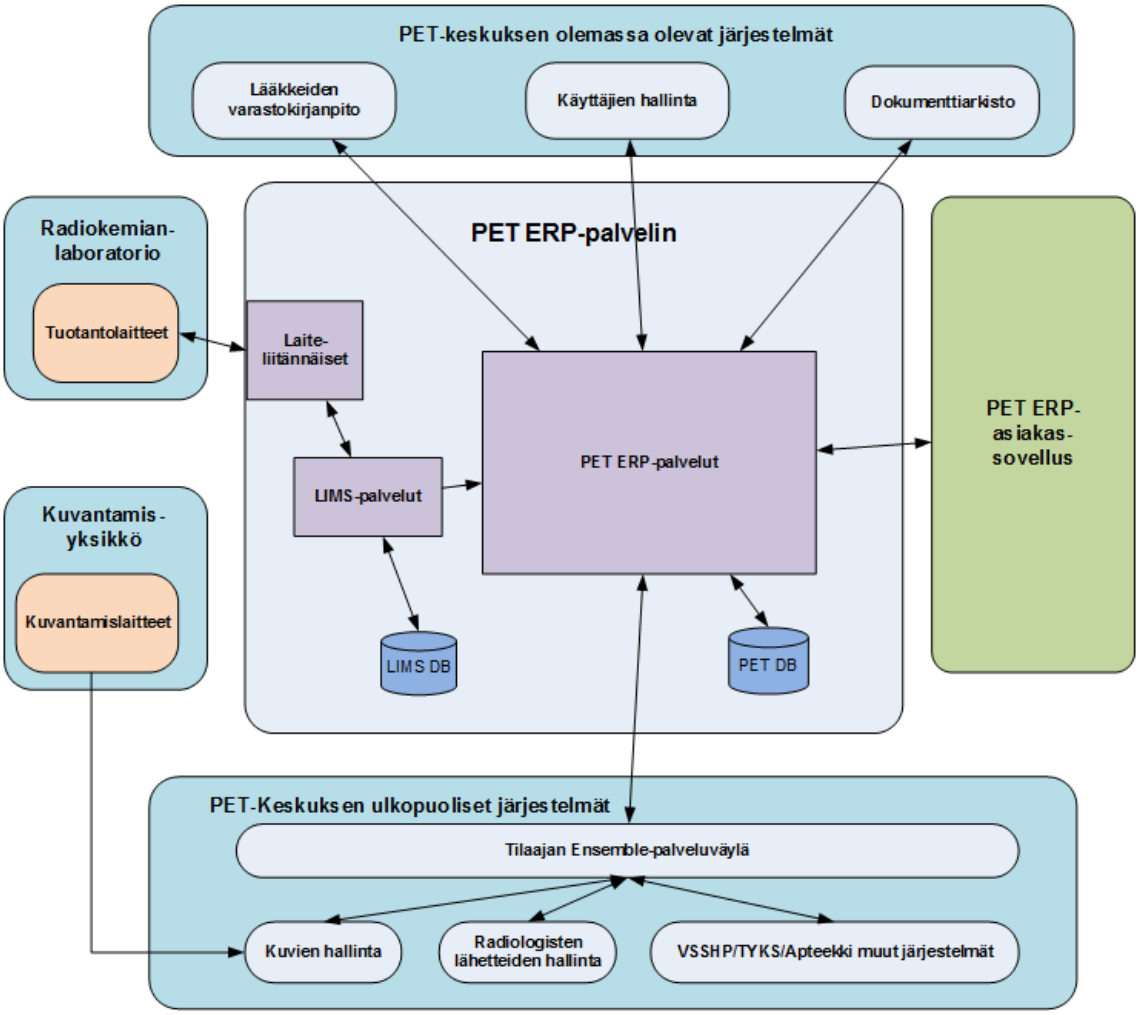
TURKU PET Centre on PET diagnostista palvelutoimintaa, kuvaustoimintaa, lääkevalmistusta, tutkimus- ja julkaisutoimintaa ja opetusta harjoittava yhteisö, jonka omistavat Varsinais-Suomen sairaanhoitopiiri (VSSHP), Turun Yliopisto (TY) ja Åbo Akademi (ÅA). PET-kuvaus (Positron emission tomography, positroniemissiotomografia) on leikekuvausmenetelmä, joka antaa tietoa kudosten toiminnasta, elimistön aineenvaihdunnasta ja lääkeaineiden käyttäytymisestä kudoksissa. Menetelmä perustuu syklotronilla tuotettuihin radioisotooppeihin, joista muodostetaan potilaisiin injisoitavia merkkiaineita. Merkkiaine kulkeutuu haluttuun tutkittavaan kohteeseen, jolloin kuvaamalla voidaan nähdä sen jakaantuminen elimistössä. Tätä kuvausta käytetään etenkin aivojen ja sydämen tutkimuksissa, sekä rakenteellisten muutosten ja tulehduspesäkkeiden etsimisessä.[5]

PET ERP -järjestelmän ja siihen liittyvien komponenttien ja liitännäisten toteutuksesta vastaa Atostek Oy. Järjestelmä tulee olemaan PET-keskuksen toiminnanohjausjärjestelmänä ja hoitamaan jatkossa kaikki operatiivisen liiketoiminnan muodot. Tähän ERP-ratkaisuun tulee sisältyä radiolääketuotantoon tarvittava GMP (good manufacturing practice) -tason LIMS (laboratory information management system) -järjestelmä. GMP-standardi sisältää asetuksen lääkkeiden hyvistä tuotantotavoista. Järjestelmän tulee sisältää aikataulutus ja projektien hallintaosio. Tämän lisäksi järjestelmä pitää voida integroida sairaalan muihin järjestelmiin sanomaliikenteen avulla. PET ERP ja siihen liittyvät järjestelmät on esitelty kuvassa 2.2.

PET ERP -järjestelmän toiminta perustuu palvelulähtöisyyteen. Kaikki toiminnot ovat jaettuina PET ERP -palveluihin omiksi palvelukokonaisuuksiin. PET-keskukselle hankittavan ERP-järjestelmäkokonaisuuden tulee kattaa seuraavat kokonaisuudet:

- laboratory information management system (LIMS),
- projektien, tutkimusten ja lääkeainevalmistuksen aikataulutus,
- integroituminen keskussairaalan tietojärjestelmiin,
- projektinhallinta,
- laitehallinta ja huoltohallinto,
- lähetteen käsittely,
- kuvauksen suoritus ja
- raportointi.

PET-keskuksen kaikki toiminnot käsitetään projekteina. Lyhytkestoiset projektit ovat diagnostisia yhden potilaan kuvauksia, ja pitkät ovat monta kuukautta kestäviä ja monia kuvauksia sisältäviä tieteellisiä tutkimusprojekteja. Näihin liittyy usein merkkiaineprojekti. Projekteihin liittyy läheisesti lähete. Diagnostisia lähetteitä varten asiakassovellukseen on toteutettu lähetteen käsittely. Diagnostisten lähetteiden pitää saapua suoraan järjestelmään ja käsittelyn sekä aikataulutuksen jälkeen palauttaa tieto aikataulutetusta kuvauksesta.




**Kuva 2.2. PET ERP-järjestelmä ja liitännät muihin järjestelmiin.**

Resurssit jaetaan pääosin laitteisiin ja henkilöstöresursseihin. Järjestelmän on tarkoitus pitää kirjaa laitteista, sille aikataulutetuista kuvauksista ja huolloista, sekä huolehtia että toimintoihin on tarpeeksi tiettyjä henkilöresursseja. Järjestelmään voidaan syöttää päivää kohden käytettävissä olevat henkilöresurssit ja järjestelmä varoittaa, jos aikataulutettujen tutkimusten työmäärä ylittää henkilöresurssit. Laitteita varten on tehty laitehallinta ja huoltohallinto.

Tutkimukseen liittyy protokolla, jonka mukaan tutkimus suoritetaan. Protokollaan on valmiiksi mallinnettu tutkimuksen vaiheet, suhteellinen aikataulu ja resurssien kulutus.

Erilaisia protokollia määritellään protokollaeditorin avulla. Protokollien avulla tutkimuksista tulee aikataulutettavia kokonaisuuksia.

Aikataulutuksessa erilaisia vaiheita sisältävät kuvausprojektit muuttuvat kuvauskalenterissa kutakin vaihetta kuvaaviksi blokeiksi, jolloin kaikilla operaatioilla on minuuttiaikataulu ja tietty määrä varattuja henkilöresursseja. Merkkiainetuotannon kalenteri luodaan automaattisesti aikataulutetun tutkimuksen mukaan, sillä tutkimuksessa käytettävä merkkiaine vaatii aina valmistusprosessin. Tälle merkkiaineelle on oma protokollansa, josta muodostuvat merkkiainetuotannon kalenteriin merkkiaineen valmistusprosessiin liittyvät blokit.

Projektinhallinnassa tutkijat voivat luoda projektikohtaisia kalentereja ja tilata kuvauksia merkkiainetutkimuksia. Projektinhallinnassa voi luoda pitkiä, monta tutkimusta sisältäviä tutkimusprojekteja, joille voidaan määrittää esiehtoja tai tarvittavia lupia, projektin vaiheita ja tehtäviä. Myös projektin etenemistä voidaan tarkkailla projektinhallinnassa.

Raportoinnin avulla voidaan luoda raportteja esimerkiksi potilaalle suoritetuista kuvauksista, tietyllä aikavälillä tehdyistä tutkimuksista halutuilla laitteilla tai tilastotietoa projektien tai kuvausten määristä PET-keskuksessa. Toiminnanohjausjärjestelmä sisältää kaiken tarvittavan tiedon raporttien luomiseen.

Järjestelmään liittyy myös joukko laitteita ja järjestelmiä, jotka integroidaan järjestelmään. Näihin lukeutuvat seuraavat tukitoiminnot: materiaalivarastot, olosuhdevalvonta, dokumenttien hallinta ja laiteintegraatiot.

Laiteintegraatioilla tarkoitetaan PET-keskuksen laitteita, jotka ovat mukana kuvantamisessa tai merkkiaineen valmistuksessa ja tuottavat tietoa, jota halutaan tarkkailla ja tallentaa toiminnanohjausjärjestelmään.

Olosuhdevalvonnalla tarkkaillaan laitteita ja tiloja ja järjestelmä tiedottaa, jos esimerkiksi säilytystilassa lämpötila poikkeaa tietyistä raja-arvoista.

### 2.2.2 PET ERP -järjestelmän vaatimukset ja reunaehdot

PET ERP-järjestelmän vaatimukset perustuvat tilaajan vaatimusmäärittelyyn, joka ei ole enää julkisesti saatavilla. Tilaajan asettamat vaatimukset on tätä työtä varten kiteytetty kuuteen kohtaan, joista varsinkin 1, 2 ja 3 vaikuttavat merkittävästi tässä työssä esitetyn järjestelmän osan suunnitteluun ja toteutukseen.

**Vaatimus 1: sata yhtäaikaista käyttäjää.** Järjestelmän toiminnot on toteutettava niin, että se mahdollistaa sadan yhtäaikaisen käyttäjän käytön. Yhtäaikaisesta käyttämisestä seuraa eheyteen, rinnakkaisuuteen ja muutoksista tiedottamiseen liittyviä ongelmia. Jär-

jestelmän täytyy varmistaa, ettei kaksi käyttäjää muokkaa samaa tietoa yhtä aikaa, ja että tieto on ajan tasalla kaikilla käyttäjillä.

**Vaatimus 2: reaaliaikainen muutoksista tiedotus.** Reaaliaikavaatimuksiin kuuluu, että järjestelmässä tapahtuvista muutoksista tulee heti tiedottaa kaikille järjestelmää käyttäville asiakassovelluksille. Projektin aikataulutuksen tai muutoksen aikataulussa tulee siis päivittyä kaikkiin käyttöliittymiin.

**Vaatimus 3: tuki sadalle kuvantamis-, mittaus- ja lääketuotantolaitteelle.** Järjestelmään tulee pystyä lisäämään laitteita, joita käytetään järjestelmässä muun muassa aikatauluttamalla niille projekteja. Laitteille voidaan kirjata huoltoja ja pitää huoltopäiväkirjaa. Erilaisten toimintojen tulee muokkautua dynaamisesti sen mukaan, mitä laitteita järjestelmään on lisätty.

**Vaatimus 4: käyttöoikeudet, sähköinen allekirjoitus ja korttikirjautuminen.** Käyttäjillä on järjestelmässä erilaisia oikeuksia ja käyttöoikeudet tulee tarkistaa Active Directoryn (AD) avulla. Toimintoja, kuten tutkimuksen valmistumista, pitää pystyä sähköisesti allekirjoittamaan toimikortin avulla. Samoin järjestelmään pitää pystyä kirjautumaan käyttäjätunnuksella tai toimikortilla.

**Vaatimus 5: Integroituminen ulkopuolisiin laitteisiin ja järjestelmiin.** Järjestelmän on kyettävä hakemaan PET-keskuksen laitteilta tietoja ja integroitumaan niihin liitännäisten avulla. PET-keskuksen muilta tietojärjestelmiltä on kyettävä vastaanottamaan ja tallentamaan tietoa, ja lähettämään tietoa takaisin järjestelmän tapahtumista.

**Vaatimus 6: GCP-, GMP- ja GAMP5 –standardit**. GMP (Good Manufacturing Practice) -standardi on kansainvälinen ohjeistus lääke- ja elintarviketeollisuudelle ja osa EU-lainsäädäntöä. Lääkevalmistuksen LIMS-järjestelmän on toteutettava GMP- ja GAMP5 (Good Automated Manufacturing Practice) -ohjeistusta. Potilaiden tietoja käsitellessä järjestelmän tulee täyttää hyvän kliinisen tutkimustavan periaatteita (GCP, Good Clinical Pratice), joihin liittyy tietojen tallentamiseen, käsittelyyn, säilytykseen ja jäljitettävyyteen liittyviä seikkoja [6]. Järjestelmästä pitää pystyä hakemaan tietylle potilaalle tehdyt tutkimukset.

### 2.2.3 Liitännät muihin järjestelmiin

PET ERP-järjestelmässä kaikki integraatiot muihin järjestelmiin toteutetaan rajapintojen avulla ja integraatiot laitteisiin liitännäisten avulla. Suurimpaan osaan laitteista ja järjestelmistä PET ERP -palvelin on suoraan yhteydessä verkon yli. Jos laite ei tarjoa rajapintaa, toteutetaan integraatio laitteen tietokantaa vasten SQL-kyselykielen (Structured Query Language) avulla.

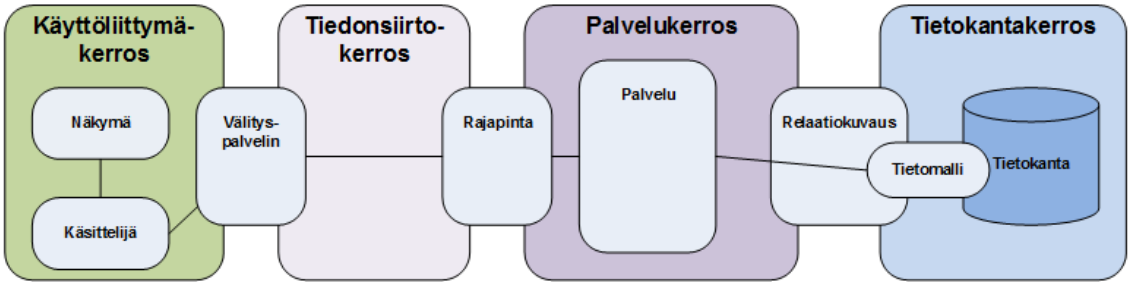
Alueen terveydenhuollon tietojärjestelmät kommunikoivat yhteisen ensemble palveluväylän välityksellä. Kuvassa 2.2 esitelty Ensemble-palveluväylä vastaa merkittävästä osasta sanomaliikennettä PET ERP -järjestelmään ja sieltä ulospäin. Järjestelmän toiminnan kannalta keskeisessä roolissa on radiologisten lähetteiden hallintajärjestelmä, josta tieto muun muassa uusista ja perutuista lähetteistä saapuu PET ERP -järjestelmään. PET ERP -järjestelmä lähettää puolestaan Ensemblen avulla lähetteiden hallintajärjestelmälle tiedon suoritetuista ja keskeytyneistä tutkimuksista.[2]

PET-keskuksella on käytössä Microsoft:n Windows Server, SQL Server -tietokanta ja dokumenttienhallintajärjestelmänä Sharepoint[7]. Koska toiminnanohjausjärjestelmissä yleensä käytetään keskitettyä tietokantaa, PET ERP -järjestelmä käyttää tätä PET-keskuksen palvelinta, ja keskitettynä tietokantana toimii palvelimen SQL Server -tietokanta.

Lääketieteellinen toimintaympäristö, järjestelmän vaatimukset ja reunaehdot, sekä liitännät muihin järjestelmiin vaikuttivat teknologian valintaan. Olemassa olevista järjestelmiä ja tietokantaa ei haluttu lähteä vaihtamaan, jolloin Microsoftin tuotteista koettiin olevan hyötyä muun muassa yhteensopivuuden kannalta. Myös toimittajan kokemus työpöytäsovelluksista ja Microsoftin tuotteista vaikutti siihen, että toteutusteknologiaksi valikoitui Microsoft .NET Framework. Työssä voidaan olettaa, että vaihtoehtoja lähdetään tarkastelemaan tasolta, jossa toteutusteknologia on Microsoft .NET ja tietokantana asiakkaalla käytössä oleva SQL Server.

# 3 JÄRJESTELMÄN ARKKITEHTUURI

Ohjelmistoarkkitehtuuri tarkoittaa keskeisten järjestelmän osien ja niiden välisten suhteiden kuvaamista korkealla abstraktiotasolla. Arkkitehtuurisuunnittelussa suunnitellaan järjestelmän jako osiin, merkittävimmät komponentit, ja päätetään käytettävät arkkitehtuurityylit. Ohjelmistoarkkitehtuurin tehtävänä on selkeyttää ohjelmiston rakennetta sekä helpottaa ohjelmistotuotantoprosessia ja järjestelmän ylläpidettävyyttä.[8]




**Kuva 3.1. Yleiskuva valitun järjestelmän arkkitehtuurista.**

Tässä luvussa esitetään valitun järjestelmän osan arkkitehtuurillisia ratkaisuvaihtoehtoja ja kuvataan asiakassovelluksen ja palvelimen välistä tiedonsiirtoa ja viestinvälitystä koskevia suunnitteluvaihtoehtoja .NET-teknologioilla. Tässä työssä järjestelmällä käsitetään kuvassa 3.1 esitelty osakokonaisuus. Kuvassa järjestelmän arkkitehtuuria on esitelty yleisellä tasolla ottamatta vielä kantaa toteutusteknologioihin tai suunnitteluratkaisuihin. Taulukossa 3.1 esitellään tässä luvussa käsiteltävät teknologia- ja arkkitehtuurivaihtoehdot.

**Taulukko 3.1 Esiteltävät teknologiavaihtoehdot**

| | | |
|---|---|---|
| Käyttöliittymäkerros | Web-sovellus | ASP.NET Web Forms |
| | | ASP.NET MVC |
| | Työpöytäsovellus | WPF [9] |
| | | WPF ja MVVM [10] |
| Palvelukerros | Web-palvelin | ASP.NET Web API |
| | Palvelin | Sovelluspalvelut |
| | | WCF |
| Tiedonsiirtokerros | Etäkutsut | WCF |
| | | SignalR |
| | Viestinvälitys | WCF |
| | | MSMQ |

Virheellinen valinta toteutusteknologiassa tai arkkitehtuurissa voi johtaa tilanteeseen, jossa järjestelmän vaatimuksia ei pystytä saavuttamaan. Tästä syystä on tärkeä käydä läpi erilaisia toteutusvaihtoehtoja järjestelmän vaatimusten kannalta optimaalisen ratkaisun löytämiseksi. Toteutusvaihtoehtoja on merkityksellistä tarkastella myös työssä esitettyjen arviointikriteerien kannalta niin toteutetun järjestelmän, kuin yleisesti palvelukeskeisten toiminnanohjausjärjestelmien näkökulmasta.

Kohdassa 3.1 esitellään .NET-ohjelmistokehys yleisesti. Kohdassa 3.2 esitellään arkkitehtuurillisia vaihtoehtoja käyttöliittymäkerroksen toteutukselle ja kohdassa 3.3 vaihtoehtoja palvelukerroksen toteutukselle. Asiakkaan ja palvelimen välisiä tiedonsiirtoratkaisuja esitellään kohdassa 3.4. Kohdassa 3.5. todetaan valitut arkkitehtuurilliset ratkaisut perusteluineen. Pois jääneihin ratkaisuihin palataan arviointikriteerien ja arvioinnin yhteydessä luvussa 5.

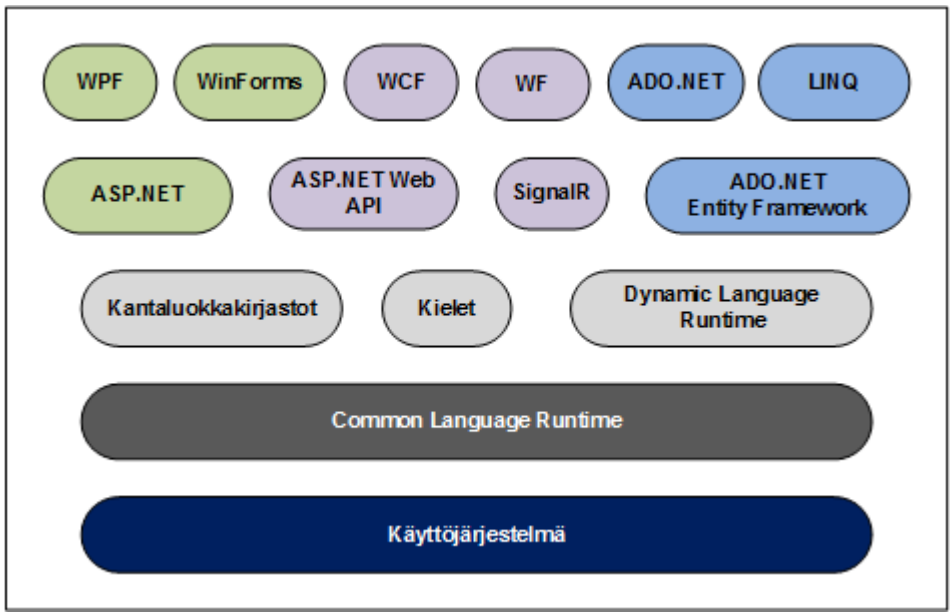
## 3.1 .NET-ohjelmistokehys

.NET-ohjelmistokehys (.NET Framework) on Microsoftin julkaisema ohjelmistokomponenttikirjasto ohjelmistojen kehittämiseen. Se on tehty helpottamaan ohjelmistokehitystä yhdistämällä teknologioita saman kehyksen alle ja käyttämällä avoimia internetstandardeja, kuten HTTP (Hypertext Transfer Protocol), XML (Extensible Markup Language) ja SOAP (Simple Object Access Protocol). [11]

Ohjelmien kääntäminen on jaettu kahteen osaan, joista ensimmäinen kääntää ohjelmointikielen Microsoftin omalle välikielelle (Microsoft Intermediate Language, MSIL) mahdollistaen kieliriippumattomuuden. Jälkimmäinen käännös suoritetaan juuri ennen ohjelman ensimmäistä käynnistystä, koska kääntäjä tarvitsee tiedon ajoympäristön käyttöjärjestelmästä ja laitteistosta. Kaksiportainen kääntäminen mahdollistaa ohjelman jakelun eri alustoille parantaen ohjelmien siirrettävyyttä.[12]

Ohjelmistokehys voidaan jakaa pääpirteittäin kolmeen osa-alueeseen. Ohjelmistokehyksen arkkitehtuuri työn kannalta merkityksellisin osin on esitelty kuvassa 3.1. Alimpana käyttöjärjestelmän jälkeen on ajonaikainen suoritusympäristö (Common Language Runtime, CLR), joka kääntää ja ajaa ohjelmia, ja sisältää muistinhallinta- ja roskienkeruuominaisuuksia. Seuraava taso sisältää kaikille .NET:n ohjelmointikielille yhteisiä luokkakirjastoja, jotka tukevat kieliriippumatonta ohjelmistokehitystä. Tällä tasolla ovat myös eri ohjelmointikielet ja dynaaminen suoritusympäristö (Dynamic Language Runtime, DLR), joka lisää palveluita dynaamisille kielille. Ylimmällä tasolla ovat käyttöliittymäkomponenttikirjastot (Windows Presentation Foundation, WPF, ASP.NET, Windows Forms), kommunikointikirjastot (Windows Communication Foundation, WCF, Windows Workflow Foundation, WF, ASP.NET Web API ja SignalR) ja tietokantoihin liittyvät kirjastot (LINQ, ADO.NET ja Entity Framework). ASP.NET-kirjasto koostuu selainpohjaisten sovellusten kehitykseen soveltuvista Web Forms tai WPF –

komponenteista, Web Services –osasta ja ASP.NET-palveluista. Kirjastoja ja teknologioita käsitellään tarkemmin seuraavissa kappaleissa ja luvuissa.[11]




**Kuva 3.2. .NET Ohjelmistokehys**

## 3.2 Käyttöliittymäkerros

Asiakassovellukseksi kutsutaan tässä asiakaspään toteutusta, joka sisältää toimintoja tekevän työkalukokonaisuuden käyttöliittymineen. Asiakassovelluksen toteutustekniikat voidaan jakaa karkeasti kahteen kategoriaan: web-pohjaisiin ja työpöytäsovelluksiin. Web-sovelluksissa käyttöliittymää navigoitaisiin selaimella ja toteutus olisi alustariippumatonta. Työpöytäsovelluksen toteutusvaihtoehtoja ohjaa Microsoftin valinta ohjaavaksi teknologiaksi. Alakohdassa 3.2.1, joka perustuu lähteeseen [12], käydään läpi web-sovelluksen ja alakohdassa 2.2.2 työpöytäsovelluksen tekniikoita.

### 3.2.1 Web-sovellus

ASP.NET on web-ohjelmistokehys web-pohjaisten sivustojen, ohjelmien tai palveluiden tekoon. Aidolla olioperustaisuudella voidaan eriyttää sivu ja sen tekninen toteutus, sekä uudelleenkäyttää kaikkia .NET-kehyksen kirjastoja ja olioita. ASP.NET tarjoaa myös tuen selainriippumattomalle kehitykselle web server –käsittelijöillä (controller). Käsittelijät ottavat huomioon selainten erilaiset tuet eri teknologioille hahmontaessaan (render) merkkauskieltä näytettäväksi käyttäjälle. Tuotantovalmiiden ohjelmien käyttöönotto ja konfigurointi, sekä komponenttien rekisteröinti on tehty tavallista helpommaksi .NET-kehyksen avulla. Palvelinkoneelle asennettu .NET-ohjelmistokehys tunnistaa ohjelman uudet komponentit tai web-sivut ja rekisteröi ne ohjelman käyttöön. Konfigurointi on koottu yhteen helposti ymmärrettävään xml-tiedostoon, jonka muutokset ASP.NET tunnistaa. Silloin ASP.NET käynnistää ohjelmasta uuden instanssin (application domain), vanhan instanssin suorittaessa keskeneräiset pyynnöt loppuun.

ASP.NET:ssä kaikki toiminnallisuus toteutetaan yleensä web-palvelimella, joten aina kun joku toiminto tapahtuu, selaimen pitää lähettää pyyntö palvelimelle, vastaanottaa uusi kopio sivusta ja päivittää sivu. Muun muassa ASP.NET AJAX:n (Asynchronous JavaScript and XML) avulla voidaan toiminnallisuus tehdä asiakaspäässä, ja pyyntö palvelimelle asynkronisesti. Lisäksi palvelimen ei tarvitse lähettää kokonaan uutta päivitettyä sivua, vaan vain päivitystä kaipaavan osan. Web-ohjelmien teossa voidaan käyttää .NET-ohjelmistokehyksen luokkia siinä missä tavallisessa työpöytäsovelluksessa. Web-ohjelmat eroavat tavallisista ohjelmista kuitenkin kahdella tavalla:

- Web-ohjelmat suoritetaan palvelimella. Käyttäjän syötettyä tietoa sivulle, sivu lähetetään palvelimelle käsittelyä varten. Palvelimella tehdään tarvittavat operaatiot, muutokset tietokantaan ja lähetetään päivitetty sivu takaisin asiakkaalle.
- Web-ohjelmat ovat tilattomia. Kun sivu on hahmonnettu käyttäjälle HTML-sivuksi, web-lomakkeen oliot tuhotaan ja kaikki tieto asiakkaasta katoaa. Tämä vaikeuttaa saumattomasti toimivien web-ohjelmien tekoa.

**ASP.NET Web Forms (Web pages)** sisältää web-ohjelman käyttöliittymäkomponentit, eli web-sivut, joita asiakas pyytää ja selain näyttää. Web Forms sallii käsittelijöiden käytön sivuissa. Jokaisella sivulla on oma käsittelijänsä. Ajettaessa ASP.NET lukee web-lomakkeen, luo tarvittavat oliot, tekee toimenpiteet ja luo sivun.

**Taulukko 3.2. ASP.NET Web Forms ja MVC –teknologioiden keskeiset eroavaisuudet. [13]**

| **ASP.NET Web Forms** | **ASP.NET MVC** |
|---|---|
| Sivupohjainen malli, jossa jokaisella sivulla on oma käsittelijänsä. | malli-näkymä-käsittelijä –arkkitehtuurissa näkymillä on yksi tai useampi yhteinen käsittelijä. |
| Ei toimintojen eriyttämistä, koska käsittelijä on tiukasti sidottu malliinsa. | Haasteet voidaan eriyttää, sillä käsittelijä ja malli hyvin eriytetty. |
| Tilaton, mutta tilallisuus voidaan saavuttaa erillisellä viewstate-komponentilla. | Tilaton. |
| Oma sivupohjainen elinkaari. | Ei sivupohjaista elinkaarta. |
| Palvelin pään käsittelijöiden avulla voidaan HTML:n tiedoista luoda abstraktio. Vain minimaalinen tieto HTML:stä, JavaScript:stä ja CSS:stä tarvitaan. | Tarvitaan yksityiskohtainen tieto HTML:stä, JavaScript:stä ja CSS:stä. |
| Yllä olevan abstraktion avulla saavutetaan rajallinen ohjattavuus HTML:stä, JavaScript:stä ja CSS:stä. | Täysi ohjattavuus HTML:stä, JavaScript:stä ja CSS:stä. |
| Nopea ja helposti opittava teknologia, joka soveltuu hyvin pienten ohjelmien tekoon pienellä henkilömäärällä. | Vaatii enemmän oppimista ja hidastaa tuottavuutta. MVC on suositeltava vaihtoehto isommille ohjelmille. |

**ASP.NET MVC** on erilainen arkkitehtuurityyli web-ohjelmien tekoon kuin perinteinen Web Forms. MVC (Model-View-Controller) -arkkitehtuurityylissä ohjelma jaetaan kolmeen loogiseen osaan: malleihin (model), näkymiin (view) ja käsittelijöihin (controller). Malli sisältää tietokantakuvauksen, tietokannan käsittelyn ja tiedon validoinnin. Näkymä määrittelee sivun ulkoasun ja luo sopivan esitysmuodon hahmontamalla HTML-sivun. Käsittelijä reagoi käyttäjän toimintaan, päivittää mallia ja vie tiedon näkymälle. Keskeisimmät erot ASP.NET MVC:n ja Web Forms:n välillä on esitetty taulukossa 3.2.

### 3.2.2 Työpöytäsovellus

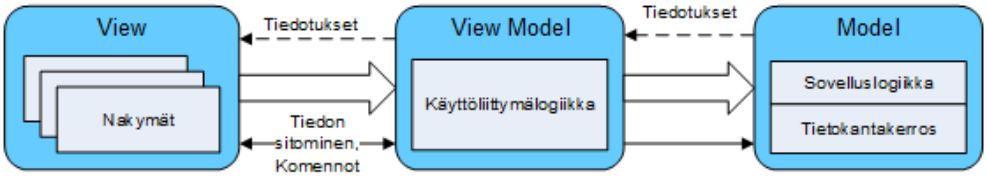
**Windows Presentation Foundation** (WPF) on .NET:n graafinen ohjelmistokomponentti, joka koostuu graafisista kirjastoista ja niiden käytön mahdollistavasta ohjelmointirajapinnasta. Kirjastoilla voi tehdä rikkaita ja visuaalisia käyttöliittymiä ja yhdistää ne käsittelijöihin vahvan sitomisen (binding) avulla. Käyttöliittymät hahmonnetaan tehokkaasti käyttämällä Windows:n Direct3D-ohjelmointirajapintaa. Käyttöliittymäkomponenteille voi luoda omia mukautettuja malleja (template) ja määritellä tyylejä (style) samaan tapaan kuin web-sivulle tyylitiedoston avulla. Visuaalinen puoli voidaan erottaa ohjelmakoodista uuden merkintäkielen (eXtensible Application Markup Language, XAML) avulla.[9]

Microsoftin käyttöliittymäteknologiat perustuvat tapahtumapohjaiseen ohjelmointiin (event-based programming). Mallissa tapahtuva muutos tai nappulan painaminen käynnistää tapahtuman, jonka tapahtumalle rekisteröity käsittelijä ottaa vastaan ja toimittaa sovelluslogiikalle. Tämä tekee näkymistä riippuvaisia sovelluslogiikasta ja tietomallista.[14]

Ohjelmistotuotteessa on kaksi selkeästi erottuvaa osaa, joiden vastuut on selkeästi määritelty: malli, joka on ohjelmiston ratkaisu tiettyyn ongelmaan, ja näkymä, jonka kautta käyttäjä voi päästä käsiksi malliin ongelman ratkaisemiseksi. Miksi erottaa nämä kaksi toisistaan? Toimintojen eriyttäminen tarkoittaa, että ohjelmakoodi on olemassa yhtä, tarkoin määriteltyä tarkoitusta varten. Tämä pätee kaikilla koodin tasoilla, yksittäisistä funktioista aina komponentteihin ja osakokonaisuuksiin. Toimintojen eriyttämisellä tähdätään komponenttien riippuvuuksien minimoimiseen. Pakolliset riippuvuudet abstrahoidaan siten, että muutokset eivät vaikuta asiakkaan ohjelmakoodiin.[10]

**MVVM-suunnitteluratkaisu** erottaa mallin ja näkymän toisistaan tarjoamalla väliin komponentin, joka huolehtii käyttöliittymälogiikasta. Riippuvuuksien, eritoten kaksisuuntaisten riippuvuuksien ratkaisemiseksi suunnittelumalli määrittelee riippuvuushierarkian. Kuvassa 3.3. on valkoisilla nuolilla esitelty suunnittelumallin riippuvuussuhteet, jossa jokainen taso on riippuvainen vain alemmasta tasosta. Näkymä ei tiedä mallista mitään, vaan se saa kaiken tarpeellisen tiedon käyttöliittymälogiikalta. Käyttöliittymälogiikka puolestaan hankkii tarpeellisen tiedon mallilta ja tarvittaessa

tiedottaa muutoksista näkymää. Yksi View Model -luokka voi vastata monen näkymän logiikasta, ja tiedon ja komentojen sitomisella muutokset päivittyvät molempiin suuntiin.[10]




**Kuva 3.3 MVVM-suunnittelumalli [14]**

MVVM-suunnitteluratkaisulla saavutetaan selkeät roolit asiakkaan ja palvelimen toteutuksessa: asiakasohjelma vastaa käyttöliittymälogiikasta ja palvelin toimii mallina pitäen sisällään sovelluslogiikan, tietomallin ja tietokannan. Näin saadaan kolmikerroksinen järjestelmä, jossa jokainen kerros on oma kokonaisuutensa ja vastaa määriteltyyn tarkoitukseen. Kerrokset on myös väljästi yhdistetty sisältäen vain vähän, ja abstrakteja keskinäisiä riippuvuuksia. Testaus helpottuu, kun eri kerrokset voidaan eristää muusta kokonaisuudesta.[10]

## 3.3 Palvelu- ja tietokantakerros

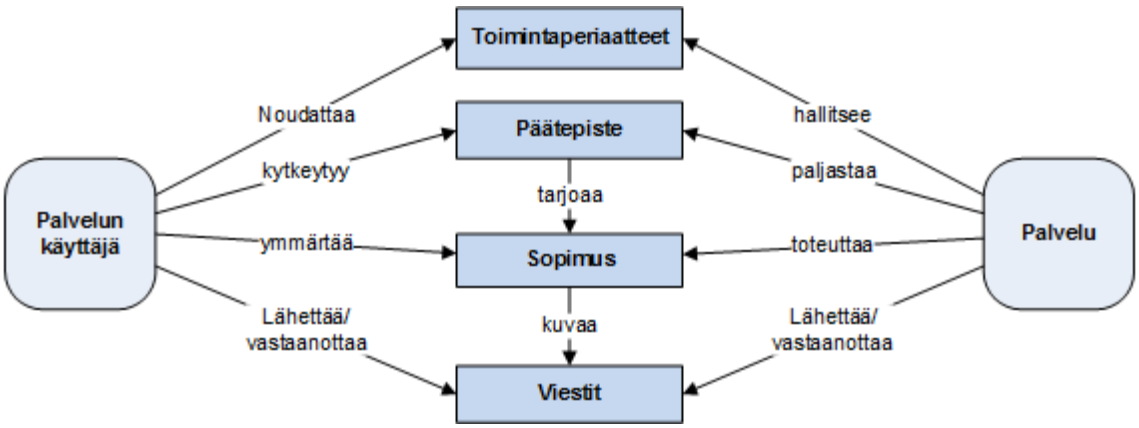
Terveydenhuollolle tyypillinen arkkitehtuurisuunnittelun periaate on liiketoimintalähtöinen kehittäminen sekä jaettujen ja yhteisten sovelluspalvelujen tarjoaminen erilaisille toiminnoille kuten PET-kuvausten suorittamiselle. Tätä palvelukeskeisen arkkitehtuurin (Service Oriented Architecture, SOA) mallia sovelletaan määriteltäessä toiminnallisia kokonaisuuksia uudelleenkäytettäviksi, avoimiksi sovelluspalveluiksi. [4]

Suomessa sosiaali- ja terveyspalveluiden järjestelmäkehityksen suunnittelulähtökohtana on tyypillisesti palvelulähtöisyys. Järjestelmiä koostetaan liiketoimintaprosesseista ja palvelut toteutetaan SOA-periaatteiden mukaisesti. Palvelut pohjautuvat avoimiin rajapintoihin ja ne voivat olla karkeustasoltaan erilaisia ja toisistaan koostuvia. Löyhästi kytketyt palvelut toteuttavat integraatiolle asetettuja tavoitteita palveluiden läpinäkyvyydestä sekä parantavat informaation saatavuutta, synkronointia ja muunnoksia eri sovelluksien ja käyttöliittymien välillä.[4]

Tässä kohdassa esitellään palvelu- ja tietokantakerroksen arkkitehtuurille vaihtoehtoja. Alakohdassa 3.3.1 esitellään palvelukeskeinen arkkitehtuuri ja alakohdassa 3.3.2 asiakas-palvelin-arkkitehtuuri. Sovelluspalveluita ja niiden toteutusperiaatteita kuvataan alakohdassa 3.3.3, joka perustuu lähteisiin [15] ja [16]. Alakodassa 3.3.4 esitellään ASP.NET Web API. Alakohta 3.3.4 perustuu lähteeseen [17]. Relaatiotietokantaa ja sen periaatteita kuvataan alakohdassa 3.3.5, joka perustuu lähteisiin [18] ja [19].

### 3.3.1 Palvelukeskeinen arkkitehtuuri

Palvelukeskeisten arkkitehtuurityylien perusajatus on järjestelmän osien jakaminen rooleihin: palvelun tarjoajiin (service) ja palvelun käyttäjiin (service consumer). Palvelu mielletään usein resurssina, joita palvelun tarjoaja valvoo ja tarjoaa ympäristölleen.[4] SOA määrittelee komponentit, niiden suhteet, rajat komponenttien käytölle ja keskinäisille vuorovaikutuksille. Järjestelmät koostuvat omavaraisista palveluista, joista jokainen vastaa johonkin tarpeeseen. Palvelut tarjoavat toimintoja sopimusten (contract) kautta. Toiminnot ovat viestejä (message), joista sopimus koostuu. Sopimusta voi verrata olio-ohjelmoinnin rajapintaan. Viestit voivat olla muun muassa http-pyyntöjä, SOAP-viestejä tai SMTP-viestejä. Viesteillä on tyypillisesti otsake ja runko, eikä lähettäjän tarvitse tietää vastaanottavan järjestelmän toteutusta, mikä puolestaan vähentää komponenttien välisiä riippuvuuksia. Palveluun päästään käsiksi päätepisteiksi (endpoint) kutsuttavien osoitteiden kautta. Kuva 3.3 selventää palvelukeskeisen arkkitehtuurin kommunikointia. Palvelu hallitsee toimintaperiaatteita (policy), joka määrittelee ehdot palvelun käyttäjälle palvelua käyttääkseen. Ehtoihin kuuluu esimerkiksi turvallisuuteen liittyviä seikkoja kuten salaus, todentaminen ja auktorisointi. Palvelun käyttäjiä voi olla erilaiset ohjelmistokomponentit, kuten asiakassovellukset ja toiset palvelut.[21]




**Kuva 3.4 kommunikoinnin periaatteet SOA:ssa. [20]**

Palvelukeskeisessä arkkitehtuurissa rajapinnat suunnitellaan yleiskäyttöisiksi, jolloin palvelu voi palvella montaa tuntematonta käyttäjää. Tällä vähennetään pisteestä pisteeseen -integraation (point-to-point integration) riippuvuusongelmia ja voidaan saavuttaa seuraavia hyötyjä:

- Uudelleenkäytettävyys. Hyvin suunniteltuja palveluja voidaan uudelleen käyttää yleensä pienillä muutoksilla.
- Sovitettavuus. Eristämällä palvelun sisäinen rakenne ja toteutus muusta järjestelmästä voidaan palveluun tehdä muutoksia helpommin. Muutosten jälkeen tarvitsee vain julkaista uudet sopimukset. Yleiskäyttöiset sopimukset ovat alustariippumattomia.
- Ylläpidettävyys. Palveluita voi ylläpitää pienemmillä tiimeillä ja erillään muusta järjestelmästä. Toimintaperiaatteisiin nojaava kommunikaatio keskittää turvalli-

suus- ja seuranta-asetusten muuttamista järjestelmässä ja helpottaa vastuunjakoa ylläpidon ja kehittäjien välillä.[20]

### 3.3.2 Asiakas-palvelin-arkkitehtuuri

Asiakas-palvelin-arkkitehtuuri on yksi yleisimmistä arkkitehtuureista, kun on käytössä palvelin. Palvelukeskeiset arkkitehtuurit ovat tulleet jälkeenpäin osittain asiakas-palvelin-arkkitehtuurin pohjalta, ja siihen sisältyy myös ajatus kapseloida tietty resurssin hallinta palvelimeen, siten että käyttäjän ei tarvitse huolehtia resurssin käyttöön liittyvistä ongelmista kuten poissulkemisesta tai eheydestä. Asiakas-palvelin-arkkitehtuurissa asiakas vastaa yleensä graafisen käyttöliittymän tarjoamisesta, ja palvelin vastaa suurimmasta osasta prosessointia, sekä tietokannan käsittelystä. Kommunikointi perustuu molempien osapuolten rajapintoihin. Rajapinta määrittelee, miten palvelimen palveluita on mahdollista käyttää tai hyödyntää. Rajapinnan taakse kätkeytyy komponentin sisäinen toteutus.[21]

Tyypillisesti asiakas-palvelin-arkkitehtuurin vuorovaikutus pohjautuu istuntoihin: Asiakas rekisteröityy tietylle palvelimelle, käyttää sen tarjoamia palveluita, ja lopuksi päättää session. Asiakas voi käyttää useaa eri palvelimen tarjoamaa palvelua yhtä aikaa, ja myös palvelin voi palvella useaa asiakasta rinnakkain. Suurimpina etuina voidaan pitää selkeää työnjakoa, arkkitehtuurin ja järjestelmän osien kuvaamista, sekä ohjelman hajauttamismahdollisuuksia, joskin tämä voi aiheuttaa ongelmia suorituskykyyn.[21]

### 3.3.3 Sovelluspalvelut

Sovelluspalvelut (web services) ovat integrointiteknologiastandardeja, jotka tehtiin alun perin vastaamaan palvelukeskeisten arkkitehtuurien ja järjestelmien vaatimuksiin. Sovelluspalveluita voidaan pitää jollain tasolla väliohjelmistoina (middleware), mutta ne keskittyvät helpottamaan järjestelmien yksinkertaisuutta ja yhteentoimivuutta (interoperability). Sovelluspalvelu suoritetaan käyttäjän koneen sijasta koneella joka isännöi ohjelmaa. Sovelluspalvelu voi vastaanottaa pyyntöjä asiakkaalta (käyttäjän koneelta), suorittaa operaatioita isännöivällä koneella ja lähettää vastauksia asiakkaalle.

Sovelluspalvelut ovat yleisesti tuettuja ohjelmistotoimittajien keskuudessa. Sovelluspalvelut, kuten muutkin integrointiteknologiat, tarjoavat kehittäjien käyttöön neljä toimintoa. Niillä voidaan:

- Löytää sopivat palvelut. Sovelluspalveluita voi hakea keskitetyiltä palveluhakemistoilta kuten UDDI (Universal Description Discovery and Integration).
- Löytää tietoa palvelusta. Sovelluspalvelu voi julkaista itsestään dokumentin, jolla löydetään tietoa palvelusta. Dokumentissa kuvataan sovelluspalvelun hyväksymät viestit, niiden vastausten rakenne, joita sovelluspalvelu lähettää takaisin. Yksi esimerkki tällaisesta kuvauksesta on WSDL (Web Service Description Language).

- Pyytää palvelua tekemään jotakin. Sovelluspalvelu määrittelee kommunikointiprotokollan, jolla viestintä sovelluspalvelun ja asiakkaan välillä toteutetaan.
- Hyödyntää palvelun ominaisuuksia. Sovelluspalvelut ovat alusta- ja kieliriippumattomia, mutta tietyistä kohdista on sovittu erikseen: protokolla pyyntöjen ja vastausten lähetykseen, tiedon formaatti ja turvallisuuden käsittely.

Sovelluspalvelut käyttävät yleisesti XML-standardia. Ne määritellään XML-kielen avulla ja ohjelmat lähettävät pyyntöjä palveluille käyttäen XML-viestejä. Sovelluspalvelut tukevat myös välittäjien/välityspalvelimien käyttöä.

### 3.3.4 ASP.NET Web API

Web-ohjelmointirajapinnat eli Web API:t (application programming interface) ovat ohjelmoitavia rajapintoja, joihin voidaan ottaa yhteys useilla HTTP-asiakasohjelmilla, kuten selaimella ja matkapuhelimella. Web API:n avulla selain voi autentikoitua palvelimen kansa ilman erillisiä liitännäisiä, käyttää standardeja http-metodeja ja otsakkeita ja Web API tukee yleiskäyttöisiä formaatteja kuten XML, XHTML ja JSON.

ASP.NET Web API on alusta alkaen kehitetty niin läpinäkyväksi ohjelmistokehykseksi kuin mahdollista. ASP.NET luotiin vastaamaan seuraaviin tavoitteisiin: parantamaan HTTP-ohjelmointia, yhdenmukaistamaan asiakkaan ja palvelimen ohjelmointikokemusta, tarjoamaan joustavan tuen eri formaateille, välttämään ohjelmointia kulmasuluilla, parantamaan yksikkötestattavuutta ja tarjoamaan useita vaihtoehtoja isännöintiin (hosting).

Web API on vain HTTP-protokollaa käyttävien palveluiden toteutukseen soveltuva teknologia. Se pohjautuu vain pyyntö-vaste-malliin ja on yksisuuntainen. Tästä syystä se ei yksinään sovi toteutettavan järjestelmän teknologiaksi, koska sillä ei voi toteuttaa vaatimus 2:n mukaista reaaliaikaista muutoksista tiedottamista. Muille suunnittelumalleille saadaan tuki yhdistelemällä muita .NET-teknologioita, kuten WCF, ASP.NET MCV ja SignalR.

### 3.3.5 Relaatiotietokanta

Keskitetty tietokanta, vaatimus 2 reaaliaikaisuudesta ja terveydenhuollon ominainen kriittinen potilastieto asettaa haasteita tietokannalle. Tutkimuksia ja raportteja varten pitää pystyä tekemään hakuja, joilla selvitetään esimerkiksi tietoja potilaan viime käynneistä. Palvelupohjaisessa järjestelmässä tietokannasta tulee järjestelmän palvelu sen käyttäjille. Tietokannan keskeisen roolin takia sen täytyy integroitua ulkopuolisiin järjestelmiin. Palveluiden avulla tiedon saaminen helpottuu, koska kaikki järjestelmän palvelut toteutetaan käyttämään samaa tiedon muotoa.

Relaatiotietokannan ominaisuuksiin kuuluvat atomisuus (atomicity), eheys (consistency), eristyneisyys (isolation) ja pysyvyys (durability). Relaatiotietokanta on tyypillistä seuraavat ominaisuudet:

- Tietokannanhallintajärjestelmä (DBMS, Database management system) tarjoaa palvelut tietokannan rakentamiselle, muokkaamiselle ja tiedon hakemiselle. Hallintajärjestelmä pitää huolen, ettei tietokannassa ole toistoa tai ristiriitaisuuksia.
- Tietomalli, fyysinen kuvaus tietokannasta, kuvaa tietokannan entiteetit ja niiden suhteet.
- Tiedonhallintajärjestelmä luo tietomallin relaatiokuvauksesta, joka on looginen kuvaus tietokannasta relaatioineen.

Tietokantakerros tuodaan palvelinkerroksen käyttöön relaatiokuvauksella. On yleensä nopeampaa tehdä kyselyitä keskusmuistissa olevaan tietokannan esitykseen. Tämän takia myös käyttöliittymäkerrokselle tuodaan tietokantakerros käyttöön relaatiokuvauksella esimerkiksi olio-relaatio-kuvauksena. Olio-ohjelmointimalli hyödyttää myös testausta, kun tietokantaa voidaan mallintaa olioina.

## 3.4 Tiedonsiirtokerros

Tiedonsiirtokerros käsittää tiedon kuljetukseen, tietoturvaan ja sarjallistamiseen liittyviä ominaisuuksia. Tiedonsiirtokerroksen tehtävä on muodostaa yhteys asiakassovelluksen ja palvelun välille, rakentaa ja sarjallistaa viesti oikeaan muotoon ja välittää viesti osapuolelta toiselle. Tässä kohdassa esitellään teknologioita tiedonsiirron toteutukselle. Alakohdassa 3.4.1 esitellään viestinvälitystä, alakohdassa 3.4.2 taas jatkuvaa ja kaksisuuntaista yhteyttä. Alakohdassa 3.4.3, joka perustuu lähteeseen [22], esitellään WCF-sovelluskehys (Windows Communication Foundation). kohdassa 3.4.4 ja 3.4.5, jotka perustuvat lähteeseen [23], esitellään TCP- ja HTTP-protokollat, niiden käyttö sovelluskehyksessä ja niiden perusominaisuuksia.

### 3.4.1 Viestinvälitys ja MSMQ

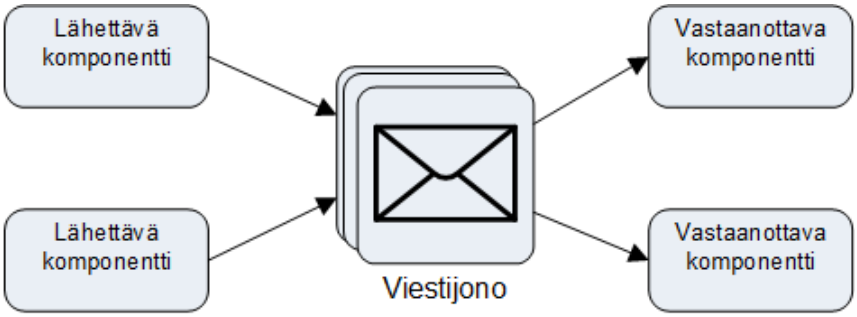
Viestinvälitys on yksi tapa ratkaista tiedonkulku. Yleinen komponentti viestinvälitykseen on suunnittelumalli välittäjä (broker). Tämä vastaa kommunikoinnin ohjaamisesta hajautetuissa järjestelmissä välittäen viestit eteenpäin ja palauttaen vastaukset kutsun tehneelle komponentille. Hajautetuissa järjestelmissä, kuten yleensä toiminnanohjausjärjestelmissä välittäjää voidaan käyttää seuraavissa tilanteissa:

- komponentit eivät tunne toistensa sijaintia vaan käyttävät niiden palveluita etäpyyntöinä,
- komponentteja halutaan lisätä, poistaa tai vaihtaa ajonaikaisesti,
- hajautus ja järjestelmän rakenne halutaan piilottaa käyttäviltä asiakkailta.

Välittäjän avulla saadaan eri komponenttien sidoksista löyhempiä, ja asiakkaat pääsevät käyttämään palvelimen palveluita lähettämällä pyynnön välittäjälle. Tämän jälkeen välittäjä tunnistaa tarvittavan palvelun ja välittää pyynnön sille. Palvelimen paluuviestin välittäjä palauttaa vastauksen takaisin kutsuvalle asiakkaalle. Välittäjää käyttämällä saadaan järjestelmästä alusta- ja ohjelmistoriippumaton, sillä välittäjän tehtäväksi jää muuttaa kutsut oikeaan muotoon.[24]

Viestiväyläarkkitehtuurissa on yksi keskuskomponentti, joka ohjaa koko järjestelmän viestinvälitystä. Muut osat lähettävät viestejä, ja viestinvälittäjä käsittelee ja uudelleenohjaa ne oikeaan paikkaan. Tämä sopii tilanteisiin, jossa järjestelmän komponenttien määrästä ei ole tarkkaa tietoa. Komponentteja voidaan lisätä ja poistaa jopa ajonaikaisesti sotkematta järjestelmän muuta toimintaa. Viestin epäonnistunut lähettäminen ei kaada järjestelmää, vaan viestejä voidaan uudelleen lähettää niin kauan kuin lähetys onnistuu. Viestiväyläarkkitehtuurin ongelmana pidetään viestin muodostamista ja tulkintaa, sekä tästä seuraavaa tehottomuutta ja ylläpidettävyyttä.[21]

.NET-ohjelmistokehys tarjoaa viestinvälityksen toteutukseen Message Queuing-teknologian (MSMQ), jolla voidaan toteuttaa kommunikointia heterogeenisessä ympäristössä. Järjestelmän komponenttien ei tarvitse olla ajossa yhtä aikaa, sillä viestit tallentuvat välittäjinä toimiville viestijonoille. Kuva 3.6 havainnollistaa, miten jonoon tallentuu usean lähettävän komponentin viestit, joita sitten vastaanottavat komponentit lukevat. MSMQ-teknologia tarjoaa taatun viestin toimituksen, tehokkaan reitityksen, turvallisuuden ja tärkeysjärjestyksessä viestittelyn. Viestinvälitys eroaa pyyntö-vaste-mallista siinä, ettei lähetetylle viestille voida paluuarvona toimittaa tietoa.[25]




**Kuva 3.5 MSMQ.[21]**

### 3.4.2 HTTP persistent connection ja SignalR

HTTP persistent connection tai HTTP keep-alive on yhden TCP-protokollan päälle rakennettu jatkuva yhteys. Sen tarkoitus on pysyä auki usean HTTP-pyynnön ja HTTP-vastauksen ajan. Yleensä sen kanssa käytetään aikarajaa, jonka jälkeen yhteys suljetaan, jos uusia pyyntöjä ei palvelimelle ole tullut. Tällä tekniikalla voidaan tehdä pitkiä toimintokokonaisuuksia järjestelmään, mutta turhaan auki olevat yhteydet hidastavat suorituskykyä. Tämä tekniikka ei myöskään tue reaaliaikaista muutoksista tiedottamista.[22]

Web-järjestelmissä on palvelimen joskus tarpeen suorittaa varsinainen pyyntö. Tämä voidaan toteuttaa palvelimen aktiivisella kyselyllä (polling), jolloin asiakas ottaa säännöllisesti yhteyttä palvelimeen päivitetyn tiedon toivossa. Tästä seuraa kuitenkin paljon kaistan joutokäyttöä ja palvelin saattaa ruuhkautua kyselyistä, jos asiakassovelluksia on paljon. Tätä varten kehitettiin palvelimen työntötekniikka (service push). Viestien lähettäminen palvelimelta asiakassovellukselle ei kuitenkaan aina riitä, esimerkiksi tilanteissa jossa palvelimen pitää:

- tietää mitkä käyttäjät ovat yhteydessä palvelimeen,
- tietää mitkä viestit lähetetään millekin käyttäjille,
- pystyä vastaanottamaan tietoa, prosessoimaan se ja lähettämään takaisin ja
- tarjota joustava ohjelmointirajapinta, jota usea asiakas voi helposti käyttää.

Tätä varten Microsoft on kehittänyt SignalR-ohjelmistokehyksen. Se mahdollistaa interaktiivisten, reaaliaikaisen ja monia käyttäjiä tukevan web-järjestelmän toteuttamisen asynkronisilla tekniikoilla. Avoimen lähdekoodin ohjelmistokehys lupaa reaaliaikaisuutta ja erinomaista suorituskykyä.[26]


**Kuva 3.6 SignalR-virtuaaliyhteys.[26]**

Kuvassa 3.5 esitellään, kuinka virtuaalinen ja jatkuva yhteys muodostetaan SignalR-kehyksen avulla. SignalR erottaa alakerrosten toteutuksen, antaen vaikutelman kehittäjälle, että käytetään jatkuvasti aukinaista yhteyttä asiakkaan ja palvelimen välillä. SignalR pitää sisällään joukon kehittäjälle läpinäkyviä komponentteja, joista se valitsee parhaan tekniikan asiakkaalle ja palvelimelle ja käyttää sitä alla olevan yhteyden tekoon. SignalR automaattisesti myös hallitsee yhteyden katkaisuja ja uudelleenyhdistämistä. Yhteyden muodostuksessa suoritetaan neuvottelu (negotiation), jossa SignalR valitsee tehokkaimman kuljetustekniikan. Tekniikat voi myös määrittää käsin, eikä ohjelmointirajapinta ole riippuvainen alla olevasta tekniikasta.[26]

### 3.4.3 Windows Communication Foundation

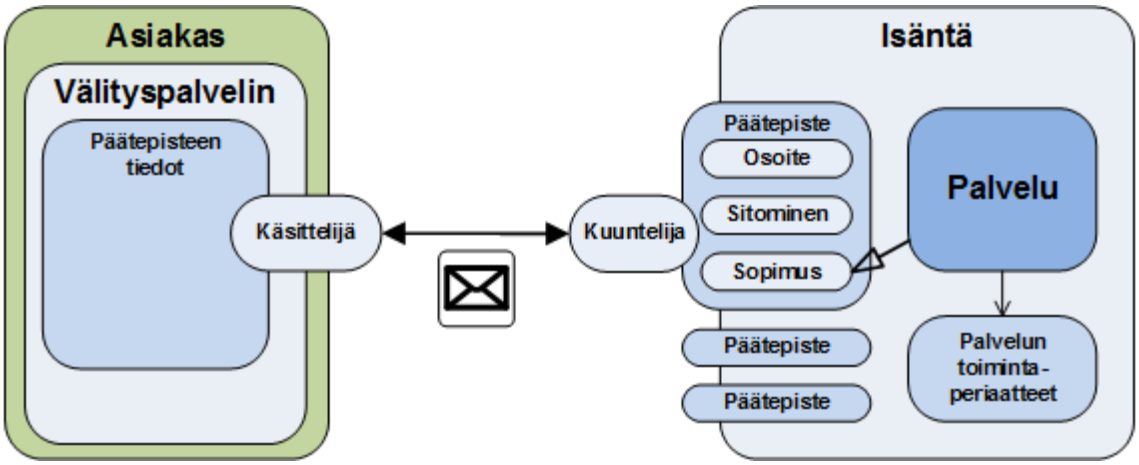
WCF (Windows Communication Foundation) on Microsoftin kehittämä sovelluskehys palvelupohjaisten sovellusten tekemiseen palvelukeskeisen arkkitehtuurin periaatteita hyödyntäen. WCF tukee monia yleisesti hyväksyttyjä tyylejä ja standardeja, kuten REST, SOAP, XLM ja JSON. Nämä ja Microsoftin muut teknologiat mahdollistavat yhteyden muihin ei Microsoft-pohjaisiin www-sovelluspalveluihin. Yleensä on vaikea pitää ohjelmallinen rakenne erossa tiedonsiirtoteknologiasta, mutta WCF:llä on mahdollista toteuttaa nämä osat lähes erillään toisistaan, eikä WCF rajoita tiedonsiirtoprotokol-

lan valintaa. WCF tarjoaa myös valmiit työkalut pääsynhallinnan ja viestien salauksen toteuttamiseen. [15]

WCF tarjoaa helpoimman tavan tuottaa ja käyttää palveluita Microsoftin alustalla. WCF:ssä on sisään rakennettu malli sovelluspalvelujen isännöimiselle (hosting), jolloin sovelluspalvelut voivat olla IIS:n tai Windowsin hallinnoimien palvelujen isännöiminä. WCF tukee monia kommunikointimekanismeja, kuten pyyntö-vaste-mallia, yksi- ja kaksisuuntaista viestittelyä ja vertaisverkkoja. Modernit ohjelmat ja järjestelmät perustuvat usein palveluihin, ja siitä WCF:ssä on juuri kyse.

Kommunikointi WCF:ssä perustuu päätepisteisiin (endpoint). Päätepiste on tietoverkon resurssi, johon yhdistämällä asiakas saa käyttöönsä päätepistettä kuuntelevan palvelun toiminnot. Merkityksellisen kommunikoinnin takaamiseksi asiakkaan pitää tietää sovelluskehyksen ABC, jonka asiakassovellus hakee ja muodostaa siitä itselleen WSDL-kuvauksen. Tämän jälkeen asiakassovellus rakentaa välityspalvelimen, jota se käyttää lähettääkseen viestejä palvelulle. Asiakkaan ja palvelun välinen kommunikaatio ja WCF:n keskeiset termit on esitelty kuvassa 3.6.

**Address** (osoite) määrittelee mihin verkon osoitteeseen asiakkaan tulisi viestejä lähettää, jotta päätepiste voi niitä vastaanottaa. Jokainen päätepiste määrittelee omat toiminnot palveluluokasta (service class), joita sen kautta voidaan kutsua. Osoite ei määrittele protokollaa, mutta se näkyy usein osoitteesta. TCP:lle palvelun osoite voisi olla esimerkiksi net.tcp://omaPalvelin:8080/esimerkkiPalvelu.




**Kuva 3.7 Kommunikointi WCF:ssä [27]**

**Binding** (sitominen) määrittelee kuinka kommunikointi asiakkaan ja palvelun välillä tapahtuu. Viestit WCF:ssä kulkevat valittua kanavaa (channel) pitkin päätepisteestä päätepisteeseen. Kanava koostuu erilaisista sitomisen osista, joista keskeisiä työn kannalta on esitelty taulukossa 3.3. Alimpana osana on tiedonsiirtotapa, mikä vie viestin perille tietoverkossa. Protokollasta huolimatta kaikki viestit pakataan kanavassa kuljettamista

varten käyttäen SOAPia. Tämän päällä on osat, jotka määrittelevät turvallisuusmekanismit ja transaktiot. WCF tukee montaa sitomistapaa erilaisiin käyttötarkoituksiin helpottaen näin integroitumista erilaisin tekniikoin toteutettuihin komponentteihin.

**Taulukko 3.3 Sitomisvaihtoehtoja ja niiden ominaisuuksia**

| Binding | Selite | Kuljetuskerroksen turvallisuus | Viestikerroksen turvallisuus | Kommunikaatio | | |
|---|---|---|---|---|---|---|
| | | | | pyyntö-vaste | yksisuuntainen | Kaksisuuntainen |
| basicHttpBinding | Sovelluspalveluille, kuten ASMX sovelluspalvelut. | x | x | x | x | |
| wsHttpBinding | Edistyneille WS-*-pohjaisille sovelluspalveluille. | x | x | x | x | |
| wsDualHttpBinding | http, joka tukee kaksisuuntaista kommunikaatiota. | x | x | x | x | x |
| netTCPBinding | .NET-pohjaisten sovellusten kommunikoinnille. | x | x | x | x | x |
| netMsmqBinding | Asynkroninen kommunikointi MSMQ-viestinvälityksellä. | x | x | | x | |
| netPeerTcpBinding | vertaisverkoille | x | | | x | x |
| msmqIntegrationBinding | MSMQ jonoilla | x | | | x | |

**Contract** (sopimus) määrittelee mitä palvelulla voidaan tehdä: mitkä toiminnot palvelu tarjoaa asiakkaalle, millaisia tietotyyppejä palvelun ja asiakaan välillä kulkee, miten virheet käsitellään, ja millaisia viestit ovat. Sopimusta voidaan pitää asiakkaan ja palvelun välisenä rajapintana, jonka palvelu toteuttaa.

Palvelu voi määrittää monta päätepistettä erilaisille asiakkaille tai palvelukokonaisuuksille. Palvelu tarvitsee isännöidä käyttöjärjestelmän puolesta. Isäntänä voi toimia vapaasti jokin käyttöjärjestelmän prosessi, sovelluspalvelu tai jopa asiakassovellus. Palvelulla on toimintaperiaatteet (service behavior), jotka määritellään asetustiedostossa.

### 3.4.4 TCP-protokolla

TCP on nopea oletukseltaan binääriformaatin protokolla. Se tukee myös muita tiedon formaatteja. Protokolla avaa putkimaisen kaksisuuntaisen kanavan päätepisteiden välille. Kaksisuuntaisuuden ansiosta palvelun on helppo tiedottaa asiakassovelluksia muutoksista. Protokolla tarjoaa korkean suorituskyvyn WCF-sovelluskehyksellä tehtyjen sovellusten välille ja on parhaimmillaan organisaatioiden sisäverkoissa. NetTcpBinding tarjoaa monipuoliset turvallisuusasetukset ja tukee luotettavaa ja tietyssä järjestyksessä tapahtuvaa viestin toimitusta.

### 3.4.5 HTTP-protokolla ja SOAP

SOAP on XML-määrittely tiedon rakenteesta viesteissä. Se standardoi kuinka tietoa vaihdetaan eri osapuolten kesken. SOAP-viesti yksinkertaisemmillaan kuljettaa tiedon viestinä ja SOAP-kirjekuori pitää sisällään viestin otsakkeen ja rungon. Esimerkissä 3.1 esitellään SOAP-kirjekuoren sisältö. Vapaavalintainen otsake-tieto voi sisältää tietoja kuljettavasta tekniikasta ja muuta metatietoa esimerkiksi sisällön koodauksesta. Runko koostuu tietosisällöstä, jossa jokainen parametri kuvaa palvelun toimintaa sarjallistetussa muodossa.

```
<soap:Envelope xmlns:soap=”http://www.w3.org/2001/12/soap-envelope”
        soap:encodingStyle=”http://www.w3.org/2001/12/soap-encoding” >
        < soap:Body xmlns:m=”http://www.example.org/stock” >
        < m:HaeTietoa >
        < m:Hakusana > WCF < /m:Hakusana >
        < /m: HaeTietoa >
        < /soap:Body >
< /soap:Envelope >
```

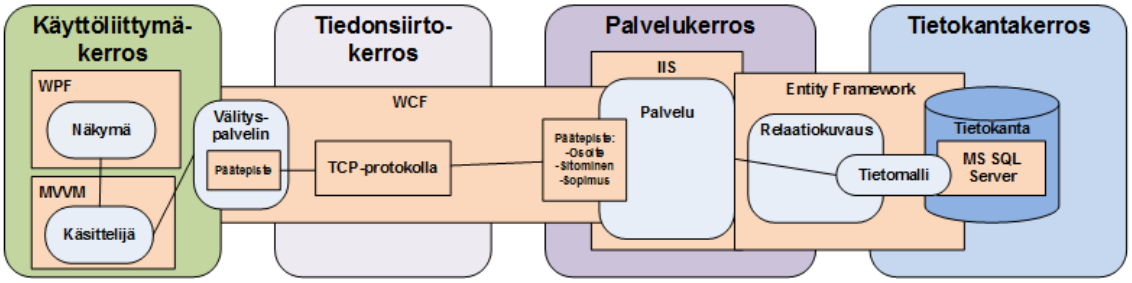
**Esimerkki 3.1. SOAP-viestin runko.**

HTTP-protokolla on yleisin protokolla verkossa kommunikointiin. HTTP antaa mahdollisuuden integroida palvelut avoimilla standardeilla mahdollistaen asiakkaiden palvelun erilaisilla arkkitehtuureilla. Jos kyseessä on joukko heterogeenisiä asiakkaita, on yhteensopivuuden kannalta http-protokollan käyttö perusedellytys. WCF tarjoaa muutaman vaihtoehdon http-protokollan käytölle, jotka esitellään taulukossa 3.3. BasicHttpBinding ei tue WS-* protokollia, mutta tarjoaa parhaan yhteensopivuuden. Tietosisältö viestissä kulkee selkokielisenä, joka tekee tiedonsiirrosta turvattoman.

Kehittyneillä WS-* protokollilla voi täydentää SOAP-viesteihin erilaisia tarpeita ja toimintaperiaatteita. Protokollilla voidaan viestinvaihtoon määritellä esimerkiksi turvallisuus- ja transaktiokäytäntöjä luotettavan kommunikaation aikaansaamiseksi. Käytännöt määritellään SOAP-viestin otsakkeeseen.

## 3.5 Valitut arkkitehtuurilliset ratkaisut

Työssä käsiteltävä järjestelmä jakautuu selkeästi kolmeen kokonaisuuteen: Käyttöliittymäkerrokseen, joka sisältää näkymät ja käyttöliittymälogiikan, palvelukerrokseen, jossa sijaitsee sovelluslogiikka, ja tietokantakerrokseen, jossa ovat tietokantapalvelut ja tietokanta. Järjestelmän arkkitehtuuri on sekoitus kerrosarkkitehtuuria, asiakas-palvelin-arkkitehtuuria ja palvelukeskeistä arkkitehtuuria. Koko PET ERP -järjestelmä perustuu palvelukeskeiseen arkkitehtuuriin ja palveluihin: LIMS:lle, muille ulkopuolisille järjestelmille ja asiakassovellukselle tarjotaan kullekin oma palvelunsa. Toisaalta taas asiakassovellus ja palvelin muodostavat keskenään asiakas-palvelin-arkkitehtuurille ominaisen asetelman. Valitut arkkitehtuurilliset ratkaisut ja teknologiat on esitelty kuvassa 3.8.




**Kuva 3.8 Valitut toteutusratkaisut**

Käyttöliittymäkerros toteutetaan työpöytäsovelluksena. Tähän syinä olivat sekä teknologiset että organisaatiolliset syyt. Toteutustiimillä oli kokemusta enemmän työpöytäsovelluksista kuin web-sovelluksista ja työpöytäsovellus nähtiin vastaaman paremmin järjestelmän vaatimuksiin. Työpöytäsovelluksen toteutusteknologiaksi valittiin yhdistelmä WPF + MVVM, jolla toteutus voidaan jakaa selkeästi näkymiin ja käyttöliittymälogiikkaan, ja työpöytäsovellus erottuu selkeästi omaksi eristäytyneeksi kokonaisuudekseen. Käyttöliittymäkerros kytkeytyy käynnistyessään palvelukerroksen palveluun ja luo siitä itselleen välityspalvelimen kommunikointia varten. Käyttöliittymäkerros toteuttaa myös palvelun määrittelemän takaisinkutsurajapinnan (callback), ja ryhtyy kuuntelemaan sitä välityspalvelimen avulla.

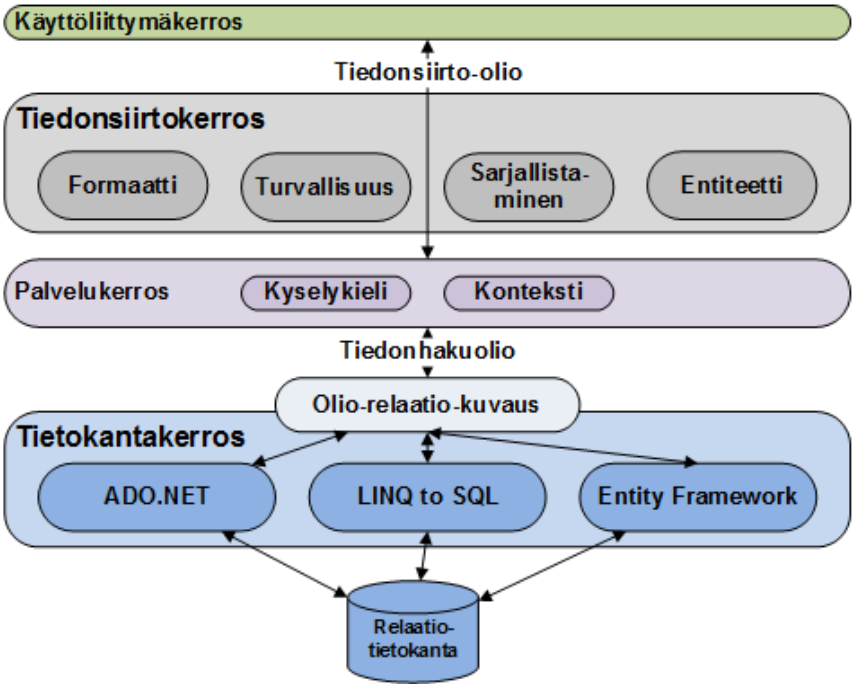
Palvelukerroksessa toteutettiin asiakassovellusta varten oma palvelunsa. Asiakassovelluksen eri toiminnot näkyvät palvelussa omina palvelukeskeisinä ja tapahtumalähtöisinä kokonaisuuksinaan: Projektit ja kuvaukset ovat omina kokonaisuuksinaan, joihin liittyy tarvittava tieto potilaasta suorittajaan, alkuperäiseen lähteeseen ja kuvauksen läpivienin kuvaavaan protokollaan. Palvelu lähettää tapahtumasta tarpeen tullen takaisinkutsun palveluun kytkeytyneille asiakassovelluksille. Näin saavutetaan reaaliaikainen ja kaikki asiakassovellukset tavoittava tapahtumista tiedottaminen. Palvelukerroksen palvelut toteuttavat WCF:n mukaisen sopimuksen päätepistettä varten. Palvelukerros on rakennettu IIS:n (Internet Information Services) päälle: kaikki palvelut isännöidään sen puolesta.

Tietokantana ja tietokannan hallintajärjestelmänä toimii PET-keskuksen olemassa oleva MS SQL Server. Relaatiotietokannanhallintajärjestelmä mahdollistaa .NET-ohjelmistokehyksen relaatiokuvauksiin perustuvien teknologioiden käytön tietokanta-kerroksessa. Relaatiokuvausteknologiaksi valittiin Entity Framework.

PET ERP -järjestelmässä komponenttien välinen kommunikaatio eli tiedonsiirtokerros päätettiin toteuttaa WCF–sovelluskehystä käyttäen. WCF antaa toteuttajalle monipuoliset vaihtoehdot kommunikaation toteutukseen. Koska tiedonsiirtokerros toteutetaan WCF-sovelluskehyksellä ja työssä käsiteltävä asiakas-palvelin osuus koko PET ERP -järjestelmästä on kokonaisuudessaan PET-keskuksen sisäisessä käytössä, on TCP-protokollan käyttö perusteltua. TCP-protokollan kaksisuuntaisuus ja lupaama suorituskyky täyttää lähtökohtaisesti vaatimus 2:n.

# 4 TIEDON SARJALLISTAMIS- JA KUVAUS-RATKAISUT

Tässä luvussa esitellään valittujen arkkitehtuurien pohjalta suunnitteluratkaisuja ja toteutusteknologioita järjestelmälle. Tarkastelun kohteena ovat tiedonsiirto- ja tietokantakerroksen teknologiat ja suunnitteluratkaisut, jotka on esitelty kuvassa 4.1. Tiedonsiirtokerros tarjoaa ratkaisun käyttöliittymä- ja palvelukerroksen väliselle kommunikoinnille, kun taas tietokantakerros auttaa meitä eriyttämään tietokantalogiikan sovellus- ja käyttöliittymälogiikasta. Tietokantakerroksen toteutuksessa on varmistettava tehokkaat kyselyt, vastattava tiedon mallinnuksen tuomiin haasteisiin, hallittava rinnakkaita transaktioita ja tiedon synkronointia. Näiden kahden kerroksen toteutus ratkaisee työn aihepiirin haasteet ja ne ovat ratkaisevassa roolissa vastatessa järjestelmälle asetettuihin vaatimuksiin.[28]




**Kuva 4.1 Tiedonsiirto- ja tietokantakerroksen suunnitteluratkaisuja.**

Kohdassa 4.1 käydään läpi oliomallinnuksen teoriaa, olio-relaatiokuvausta ja vaihtoehtoja tiedon kuvaukselle toteutettavassa järjestelmässä.. Kohdassa 4.2 esitellään tietokannan ja palvelukerroksen erottava tiedonhakuolio (data access object, DAO) ja palvelukerroksen ja käyttöliittymän välissä kulkeva tiedonsiirto-olio (data transfer object, DTO) ja vaihtoehtoja näiden toteuttamiselle. Kohdassa käydään myös läpi tiedonsiirtokerroksen suunnitteluratkaisuja muun muassa sarjallistamiseen, tiedon koodaukseen ja turvallisuuteen liittyen. Tiedonhakua tietokannasta ja tietokantakyselyjen toteutusta eri vaihtoehdoilla tarkastellaan kohdassa 4.3. Lisäksi kohdassa esitellään oliorelaation vaikutuksia tietomalliin ja ratkaisuja tietomallin toteutukselle. Kohdassa 4.4 käydään läpi

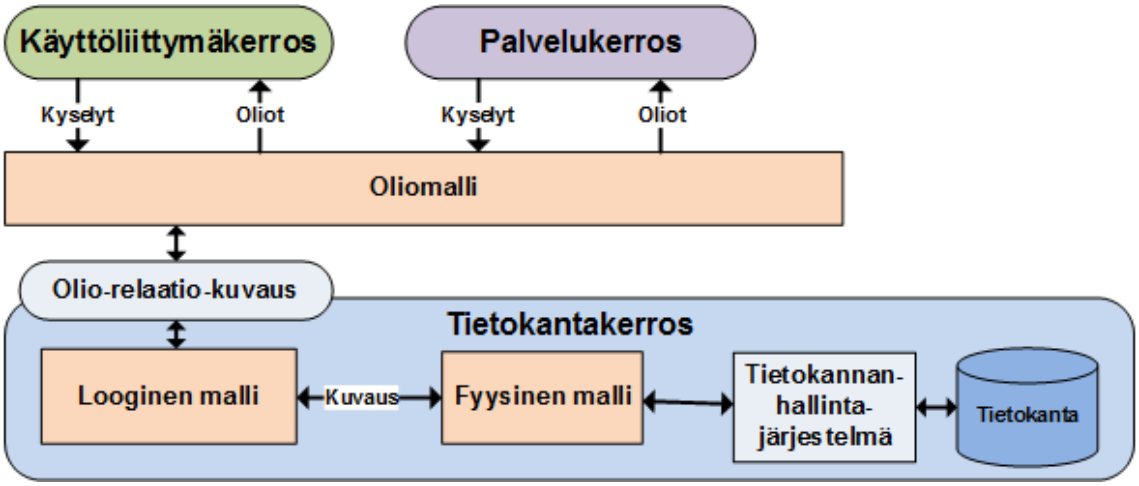
järjestelmän toteutukseen valitut ratkaisut ja annetaan perusteet arvioinnin ja tulosten mittaamisen toteuttamiselle. Valitut ratkaisut on valittu taulukossa 4.1 esitellyistä tiedon sarjallistamis- ja kuvausvaihtoehdoista.[29]

**Taulukko 4.1 Tiedon sarjallistamis- ja kuvausvaihtoehtoja**

| | |
|---|---|
| Olio-relaatio-kuvaus | ADO.NET |
| | LINQ to SQL |
| | Entity Framework |
| Tiedonhakuolio | Entity object |
| | POCO |
| | POCO proxy |
| | Self-tracking entities |
| Sarjallistaminen ja koodaus | XML |
| | JSON |
| | Binääri |
| Kyselykieli | SQL |
| | EntitySQL |
| | LINQ to Entities |

## 4.1 Olio- ja relaatiomallinnus

Ohjelmistosuunnittelussa on yleisesti käytössä lähtökohta, jossa ohjelma tai palvelu jaetaan kolmeen osaan tietosisällön perusteella. Ylimpänä ja palvelulle näkyvänä osana on sovellusalue (domain model), joka määrittelee entiteetit ja niiden suhteet järjestelmässä. Sen alapuolella on looginen malli (logical model) relaatiotietokannasta, jossa normalisoidaan entiteetit ja niiden suhteet tauluiksi ja niiden vierasavaimiksi. Fyysinen malli (physical model), joka määrittää tietovaraston yksityiskohdat, kuten indeksoinnin ja osittamisen (partitioning). Relaatiotietokantojen yhteydessä loogisesta mallista käytetään nimitystä relaatiomalli ja se kuvataan tietokannan fyysiseen malliin. Nämä käsitteet on esitelty kuvassa 4.2.[29]




**Kuva 4.2 Olio-relaatio-kuvaus ja tietokantakerroksen käsitteet.**

Oliomallinnusta ei ole standardoitu, jolloin jokainen ohjelmointiympäristö toteuttaa mallinnuksen omalla tavallaan. Oliomallinnus kuvaa järjestelmän kuin se olisi rakennettu olioista: abstraktioista, joilla on identiteetti, tila ja käyttäytyminen (behavior). Käyttäytyminen sisältää olion operaatiot, joita varten olio tarjoaa itsestään rajapinnan. Nämä ominaisuudet erottavat oliot relaatioista.[30]

Relaatiotietokannat ovat olleet vuosikymmeniä puutteellisia vastaamaan relaation perusperiaatteisiin. Relaatiomallinnus kuvaa tiedon predikaattilogiikkana ja totuuslauseina. Näiden muuttamiseksi tietokannan ymmärtämään muotoon tarvitaan termejä, jotka on esitelty taulukossa 4.1.[30]

**Taulukko 4.2 Relaatiomallinnuksen termit ja tietokannan vastaavat termit.**

| **Termi** | **Selite** | **Tietokannassa** |
|---|---|---|
| Relaatio | Samanlaisten monikkojen joukko | Taulu |
| Monikko | Lista, jossa yksi arvo kutakin ominaisuutta kohden | Rivi |
| Ominaisuus | Arvon merkitys relaation monikossa | Sarake |
| Tyyppi | Ominaisuuden tyyppi, ja sen mahdolliset arvot | Tietotyyppi |

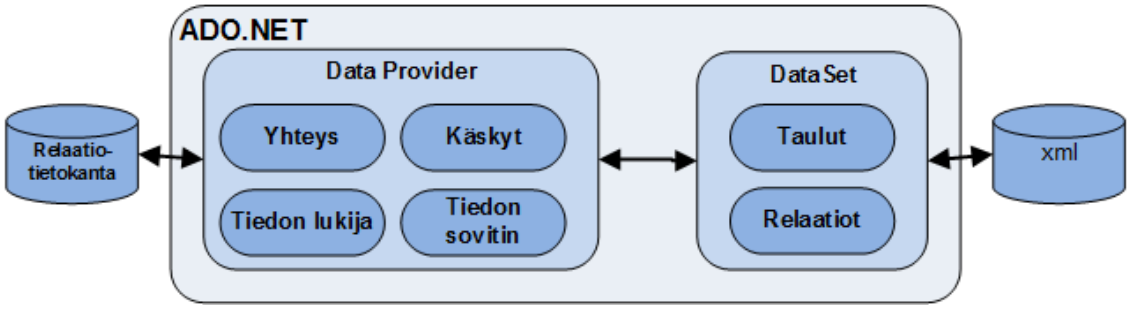
Mallien erilaisuus tuo haasteita oliopohjaisen sovelluksen ja sitä tukevan relaatiotietokannan välille. Ideaalitilanteessa olisi yksi integroitu malli, joka kuvailisi molempia lähestymistapoja. Tätä relaatiomallinnuksen ja oliomallinnuksen välistä kuilua poistamaan tehtiin olio-relaatio-kuvaus (object-relational mapping, ORM). Oliorelaatiokuvaus on prosessi, jossa tietokannan relaatiomallista tehdään oliomalli järjestelmissä, jotka tukevat näitä lähestymistapoja. Palvelukeskeisessä arkkitehtuurisuunnittelussa tietokantakerros sijaitsee yleensä palvelimella ja tarjoaa palvelukerroksen palveluille pääsyn tietokantaan olio-relaatio-kuvauksen-avulla, kuten kuvassa 4.2 esitetään. Tietokannan tauluissa olevat rivit pystytään muuttamaan olioiksi ja takaisin kuvaksen avulla. Olio-relaatio-kuvaus tarjoaa sovelluksille tietokantariippumattomuuden, joka mahdollistaa käytettävän tietokannan piilottamisen varsinaiselta sovellukselta. Olio-relaatio-kuvaus tarjoaa myös tiedon validointia ja tietoturvaa. Kuvaus voi huolehtia, ettei tietokantaan voida tallentaa virheellistä tai puutteellista tietoa.[30]

Seuraavissa alakohdissa esitellään toteutusteknologioita olio-relaatio-kuvauksen ja tietokantakerroksen toteuttamiselle PET ERP -järjestelmässä .NET-ohjelmistokehyksen tietokantatekniikoilla. Alakohdat 4.1.1-4.1.3 perustuvat lähteeseen [28] ellei toisin mainita.

### 4.1.1 ADO.NET

ADO.NET on joukko ohjelmistokomponentteja, jotka voidaan jakaa pääpiirteittäin kahteen luokkaan: tietoa sisältäviin komponentteihin (DataSet) ja tiedon tarjoajakom-

ponentteihin (Data Provider), jotka vastaavat tiedon luvusta, hausta ja päivittämisestä. ADO.NET-komponenttien korkean tason arkkitehtuuri on esitetty kuvassa 4.3




**Kuva 4.3 ADO.NET.**

DataSet-luokkaa voidaan pitää kevyenä tietokannan tietyn hetkisenä kätkönä. Se vastaa tiedon relaatiomallista esitystä tietokannan taulusta. Tiedon tarjoajakomponentin avulla luodaan DataSet-luokka tietokannan tiedoista, sille voidaan suorittaa operaatioita, ja siihen kohdistuneet muutokset voidaan sovittaa takaisin tietokantaa. Tämä tekee DataSet-luokasta riippumattoman tietolähteestä. DataSet-luokka voidaan jäsentää xml-tiedostoksi ja takaisin parantamaan yhteensopivuutta tiedonsiirrossa DataSet-luokan ja asiakkaan välillä.

ADO.NET tarjoaa seuraavat hyödyt:

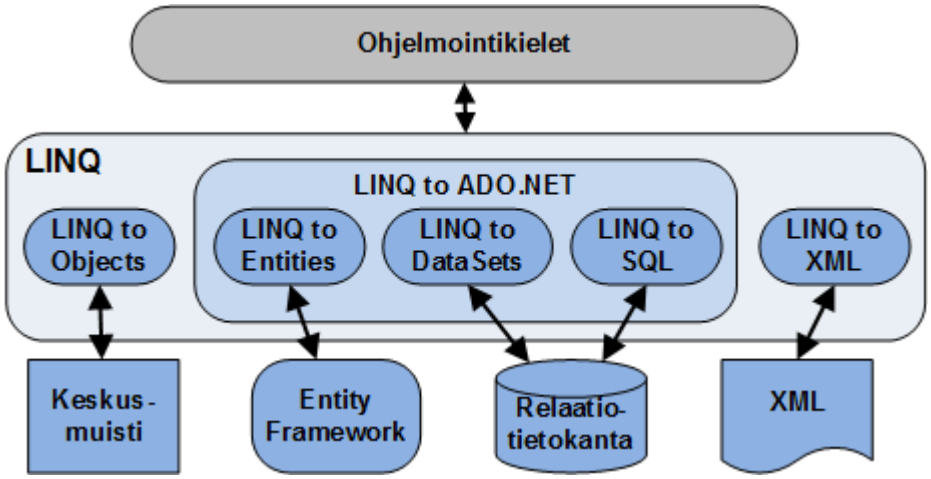
- Yhteentoimivuus. Kyky kommunikoida heterogeenisten ympäristöjen kanssa.
- Skaalautuvuus. Kyky palvella kasvavaa määrää asiakassovelluksia heikentämättä järjestelmän suorituskykyä.
- Tuottavuus. Kyky tuottaa nopeasti kestäviä tietokantaohjelmistoja käyttäen ADO.NETin rikasta ja laajennettavaa ohjelmistokehystä.
- Suorituskyky. Yhteydetön tila tietomalliin vähentää tietokantakutsujen määrä palvelimelle, kun kutsuja voidaan tehdä suoraan luotuihin DataSet-luokkiin.[11]

ADO.NET kärsii olio-relaatio-kuvauksen puutteellisuudesta: relaatioista olioiksi muuttamiseen tarvitsee tehdä paljon mukautettua ohjelmointikoodia. Ohjelmakoodin ylläpito on muutosten yhteydessä työlästä. Tietokantakyselyt ovat riippuvaisia tietokannasta, jolloin kehittäjän täytyy kirjoittaa tietokantaspesifit kyselyt käsin. Myös SQL-käskyihin perustuvat kyselyt ovat alttiita injektiohyökkäyksille.

### 4.1.2 LINQ-komponenttikirjasto ja LINQ to SQL

LINQ (Lanquage Integrated Query) on .NET-ohjelmistokehyksen komponenttikirjasto, jolla voi tehdä erilaisia kyselyjä muistissa tai tietokannassa olevaan tietoon. LINQ tarjoaa .NET-ohjelmointirajapinnan ja noin 40 erilaistaa kyselyä. Komponenttikirjasto on esitelty kuvassa 4.4. Muistissa olevien olioiden ja niiden tietojen keruuseen on oma LINQ to Objects -komponentti. Samoin XML-tiedolle on LINQ to XML ja tietokannan tietoja relaatioina kuvaaville DataSet-luokille LINQ to DataSet. LINQ to Entities on

Entity Framework -kehyksen oliomallia varten tehty kyselykomponentti. Komponenttikirjasto luo LINQ-kyselyistä parametrisoituja SQL-kyselyjä, jolloin ohjelmoijan ei tarvitse luoda tai ylläpitää vaikeita SQL-kyselyjä.[12]




**Kuva 4.4 LINQ-komponenttikirjasto.[31]**

LINQ to SQL on LINQ-perheen komponentti, joka tarjoaa graafisen työkalun olio-relaatio-kuvauksen luomiseen ja kyselyrajapinnan relaatiotietokantoihin. Sen avulla luodaan ohut abstraktiotaso SLQ-palvelimen tietokannan päälle. Jokaista taulua kohden luodaan entiteetti-luokka ja jokaista vierasavainta kohden assosiaatiot luokkien välille. Assosisaatioiden avulla voidaan siirtyä luokasta toiseen, eikä LINQ-kyselyihin tarvitse tehdä liitos-operaatioita. Tätä entiteettiluokista koostuvaa oliomallia (object model) vasten voidaan tehdä LINQ-kyselyitä.[31]

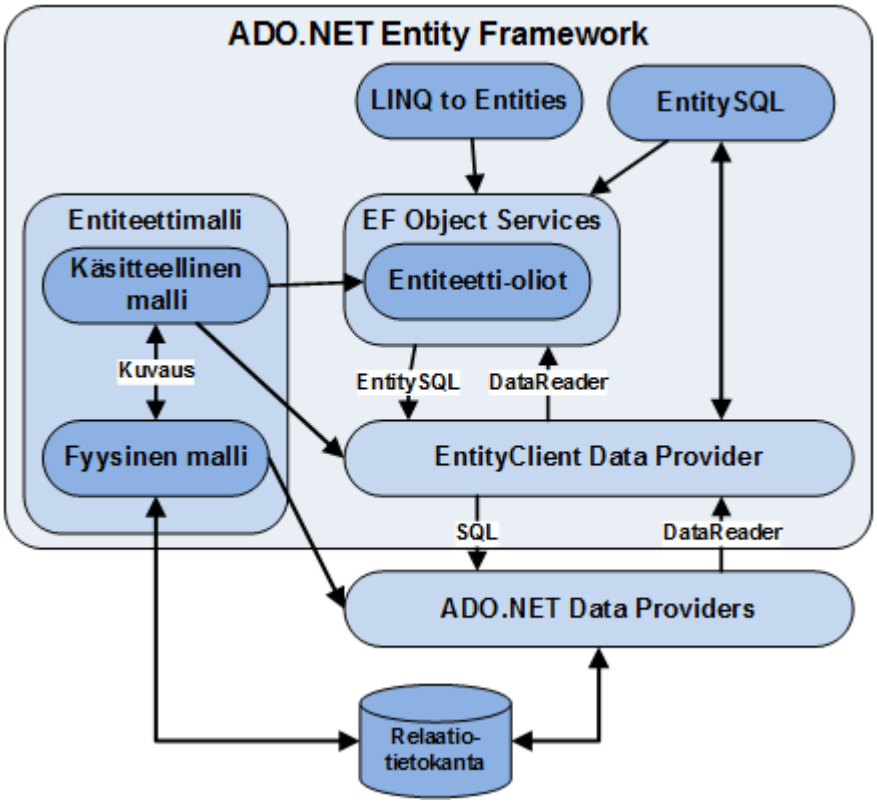
Yhteys tietokantaan luodaan DataContext-oliolla. Konteksti-olio määrittelee yhteyden tietokantaan, hallitsee tietokannan taulujen kuvauksen entiteeteiksi. Olio myös luo parametrisoidut SQL-kyselyt LINQ-kyselyistä ja välittää ne alla olevalle ADO.NET:n Data Provider –komponentille. Konteksti-olio huolehtii myös muutosten hallinnasta entiteeteissä ja niiden assosiaatioissa. Kontekstin sisällä ladattuihin entiteetteihin kohdistuneet muutokset ovat kontekstikohtaisia. Entiteettien arvoja voidaan verrata tietokannan alkuperäisiin ja tarvittaessa päivittää tietokanta olion tarjoamilla operaatioilla. Rinnakkaisuusongelmat käsitellään kun muutokset viedään tietokantaan.[31]

LINQ to SQL -komponentti kärsii myös puutteellisesta olio-relaatiokuvauksesta. Oliomalli yhdistää entiteetti-luokat tietokannan sarakkeisiin tiukasti: sillä voidaan luoda vain yksi yhteen -suhteista entiteettejä. Myöskään entiteettien muodostamaa kokonaisuutta ei voi muokata ja tallentaa kerralla, vaan jokainen entiteetti pitää hoitaa itsenäisesti.

### 4.1.3 Entity Framework

ADO.NET Entity Framework on .NET-sovelluskehyksen tietokantakerroksen ohjelmistokomponenteista kehittynein. Se on rakennettu ADO.NET ja LINQ to SQL -

komponenttien jatkoksi ja mahdollistaa relaatiotietokannan taulujen kuvauksen relaatiomallia abstraktisemmalle käsitteelliselle tasolle. Tätä olio-relaatio-kuvausta muistuttavaa kokonaisuutta kutsutaan entiteettimalliksi (entity data model). Eniteettimallin päälle on rakennettu EntityClient Data Provider -komponentti ja Object Services -komponentti palvelu- ja käyttöliittymäkerrosten käyttöön ja entiteettien (Entity) hallintaan. Entity Framework tarjoaa myös graafisen työkalun entiteettimallin luomiseksi. Työkalun avulla käsitteellisestä mallista voidaan luoda relaatiotietokannan taulut tai toisinpäin. Entity Frameworkin rakennetta esitellään kuvassa 4.5.[31]




**Kuva 4.5 Entity Framework [32]**

Entiteettimallin käsitteellinen malli koostuu entiteeteistä ja assosiaatioista. Entiteetin tyyppi määrittä sen nimen ja entiteetti voi koostua yhdestä tai useasta tietokannan saraketta vastaavasta ominaisuudesta, jolla on myös tyyppitieto. Entiteettimalli tukee kompleksejа tyyppejä kuten aikaa. Entiteettejä voi myös periyttää ja entiteettimalli tukee yhdestä-moneen ja monesta-moneen assosiaatioita. EntitySet vastaa ADO.NET:n DataSet –komponenttia ja säilöö samankaltaisia entiteettejä. Vastaavasti AssociationSet säilöö samanlaisia assosiaatioita. Kokonainen entiteettimallin instanssi, EntityContainer, sisältää loogisen ryhmittelyn EntitySet ja AssociationSet –kokoelmia.

EntityClient Data Provider -komponentti tarjoaa ADO.NET Data Provider -komponentin tapaan kolmen laista palvelua: yhteyksiä yhdistää alla olevaan tietolähteeseen, käskyjä tehdä kyselyjä ja kutsuja tietolähteeseen ja DataReader –komponentin näyttämään käskyjen tuloksia. EntityClient tarjoaa nämä palvelut käsitteellistä mallia vasten

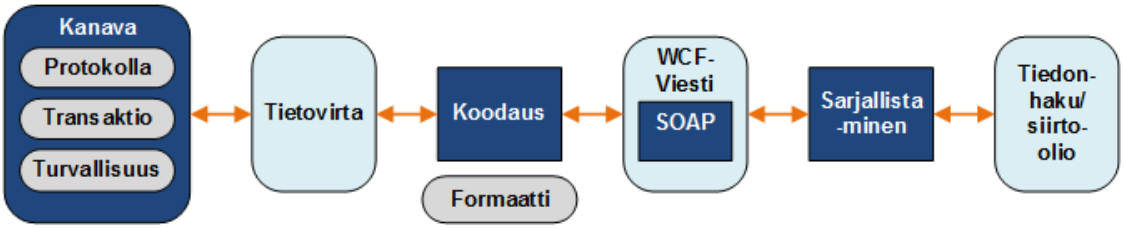
eriyttäen tiedonhaun toteutuksen kokonaan muusta järjestelmästä. Kyselyt käsitteelliseen mallia vasteen tehdään entiteetteihin perustuvalla EntitySQL-kyselykielellä. EntityClien tarjoaa kerroksen, jonka avulla on mahdollista abstrahoida tietokannan ohjelmointirajapinta ja näin tehdä Entity Framework:stä tietokantariippumaton. Tällä tavoin saavutetaan tuki myös ei-relaationaallisille tietokannoille.

Entity Framework Object Services -komponentti tarjoaa entiteettien abstrakoinnin olioiksi, palveluita entiteetti-olioiden käsittelylle ja olio-relaatio-kuvauksen kaltaista toiminnallisuutta. Palveluihin kuuluu entiteetti-olioiden tilan ja muutostenhallinta, olioiden ja niiden suhteiden lataus ja navigointi ja tuki kyselykielille kuten LINQ to Entities ja Entity SQL. Entiteetti-oliot voidaan määrittää käsin tai valita. Entiteeteistä kerrotaan lisää alakohdassa 4.2.4 tiedonhakuoliot. Object Services -komponentti sisältää myös konteksti-luokan, jonka sisällä entiteettimalliin voidaan kohdistaa kyselyjä.[32]

Etuna Entity Framework:ssa LINQ to SQL:ään on entiteettimalli, joka ylläpitää taulujen ja olioiden välisiä kuvauksia. Kuvaukset voidaan vaihtaa milloin vain, tarvitsematta tehdä muutoksia tietoa käsittelevään ohjelmakoodiin. Entity Framework myös korjaa LING to SQL:n puutteellisen olio-relaatio-kuvauksen.

## 4.2 Tiedonsiirto

Tietokannasta haettu tiedonhakuolio käy läpi erilaisia operaatioita ennen sen lähettämistä kuljetuskelpoisessa muodossa palvelulta asiakassovellukselle. Tässä kohdassa käsitellään tiedonsiirtokerroksen toteutukseen valitun WCF-teknologian tarjoamia vaihtoehtoja tiedon muutokselle ja kuljetukselle asiakassovelluksen ja palvelun välillä. Olioiden läpikäymää muutosta kuljetuskelpoiseen muotoon esitellään kuvassa 4.6.




**Kuva 4.6 Tiedon muutos kuljetuskelpoiseen muotoon WCF:ssä.[33]**

Tiedonhaku-, tiedonsiirto-olioita ja niiden välistä suhdetta käsitellään alakohdassa 4.2.1. Ensimmäisenä tehtävää operaatiota, sarjallistamista, käsitellään alakohdassa 4.2.2, joka perustuu lähteeseen [22]. Tieto koodataan tiettyyn formaattiin lähetystä varten, ja koodausta ja formaatteja käsitellään alakohdassa 4.2.3, joka perustuu lähteeseen [22]. WCF:n päätepisteen sitomisessa voidaan määrittää protokollan lisäksi ominaisuuksia, kuten turvallisuutta ja transaktioita koskevia määrityksiä. Transaktioita ja rinnakkaisuutta koskevia ominaisuuksia ja haasteita käsitellään alakohdassa 4.2.4 ja turvallisuusnäkökohtia alakohdassa 4.2.5.

### 4.2.1 Tiedonhaku- ja Tiedonsiirto-oliot

Tiedonhakuolio eriyttää tietokantakerroksen ja palvelukerroksen toisistaan. Sovellusalue voi siis olla riippumaton olio-relaatio-kuvauksesta, tietokannasta ja tietokannanhallintajärjestelmästä tiedonhakuolioiden avulla. Tiedonhakuoliot voidaan käsittää myös suunnittelumallina, jonka avulla sovellusaluetta voidaan uudelleen käyttää eri tietokantakerrosten kanssa. [34]

Entity Framework -sovelluskehyksessä tiedonhakuolioita voidaan kutsua entiteettiluokiksi tai entiteeteiksi. Entiteettimallin kantaluokat tarjoavat Object Services -komponentin, jonka avulla entiteeteistä materialisoidaan olioita. Entiteettiluokat ovat laajennettavia ja niihin voidaan lisätä jäsenmuuttujia käsin. Entity Framework tukee myös omien luokkien käyttämistä entiteettimallin kuvaamiseen. Seuraavia luokkia voidaan käyttää entiteettimallin kuvaamiseen oliotasolla:[28]

- **EntityObject** on Entity Frameworkin oletusluokka entiteettimallin kuvaamiseen, jonka kantaluokasta entiteettiluokat periytyvät. Entiteetit ovat pysyvyydestään tietoisia (persistence aware) ja niiden ominaisuuksiin kuuluu laiska lataaminen (lazy loading) ja muutosten jäljitys (change tracking). Tietokantakerroksessa kontekstin sisällä Entity Framework huolehtii entiteettien päivittämisestä, jos entiteetin tietoihin tietokannassa kohdistuu muutoksia. EntityObject-luokkaan perustuvat entiteetit ovat riippuvaisia sovelluskehyksestä ja sisältävät paljon metatietoja.[28]

- **POCO**t (Plain Old CRL Object) ovat puhtaita olioita, jotka eivät sisällä minkäänlaisia viitteitä eivätkä periydy mistään kolmannen osapuolen ohjelmakoodista, eikä niiden ei tarvitse toteuttaa erikoisia rajapintoja. POCOt voidaan määritellä itsenäisesti joko käsin tai mallin avulla, ja niiden avulla luodaan oma oliomalli entiteettimallista. POCOlle täytyy määritellä oma konteksti, jonka avulla tietokantakyselyt mahdollistetaan. POCO:t eivät tue suoraan Entity Framework:n hienouksia kuten laiskaa latausta ja muutosten seurantaa, mutta nämä voidaan toteuttaa oman kontekstin sisällä. POCOt ovat löyhästi kytkettyjä Entity Frameworkiin. [35]

  **POCO proxy** -luokat sisältävät välityspalvelimen. Välityspalvelimet luodaan ajonaikaisesti, ja ne periytyvät POCO-luokista, joka tekee niistä erilaisia kuin POCOt. Välityspalvelimen avulla toteutetaan muutosten seurantaa tiedonantomekanismilla [28].

- **Self-Tracking Entities -**luokkia tarvitaan kun entiteettejä pitää jakaa järjestelmän eri kerrosten välille. Entiteetit pystyvät jäljittämään sen sisältämään tietoon kohdistuneet muutokset. Ne säilyttävät tilansa Entiteetit luodaan mallin avulla,

ja ne voivat olla pohjimmiltaan POCOja. Entiteetin voi lisätä Entity Frameworkin kontekstiin, jolloin kontekstin sisällä muutokset voidaan tallentaa tietokantaan. Tallennus huomio koko entiteetti-puun muutokset entiteettiin ja sen assosiaatioihin. [28]

Tiedonsiirto-olio on olio, joka kuljettaa tietoa prosessien tai komponenttien välillä ja sen tarkoitus on vähentää operaatiokutsujen määrää [36]. Tiedonsiirto-olio yhdistää tietokannan taulujen tiedot sovelluksen käyttöön, eikä sisällä olioille tyypillistä käyttäytymistä, kuten operaatioita. Tiedonhakuolioiden käytöllä voidaan vähentää hyötykuorman kokoa ja monimutkaisuutta ja karsia metatietoja tehokkuuden parantamiseksi. Entity Framework -sovelluskehyksen entiteeteistä voidaan käsin muokata tiedonhakuolioita. Oman tiedonsiirtokerroksen luominen on työlästä, koska kaikkien entiteettien tiedot pitää käsin kuvata tiedonsiirto-olioihin.[37]

Palvelussa tiedonsiirto-oliosta tulee tyypillisesti tiedonhakuolio, johon kohdistuu joitakin tietokantaoperaatioita. Myös asiakassovelluksessa tiedonsiirto-oliosta tulee tiedonhakuolio, mutta tähän kohdistuu eri tarkoitukseen kohdistuvia operaatioita, kuten tiedon näyttämistä erilaisessa muodossa tai tiedon muokkaamista.[34]

Tiedonsiirtokerroksen luominen tiedonsiirto-olioilla ei kuitenkaan ole välttämätöntä. Sovelluskerroksen oliot soveltuvat tiedonsiirto-olioiksi, jos ne on toteutettu sarjallistamiskelpoisiksi. Tiedonhakuoliot voivat kelvata sellaisenaan tiedonsiirto-olioiksi. Tiedonhakuolioista voidaan myös karsia turhaa lisäkuormaa (overhead) pois tehokkuuden parantamiseksi. Näin säästytään tiedonsiirto-olioiden käsin luomiselta ja samoja tiedonhakuolioita voidaan käyttää läpi järjestelmän. [34]

### 4.2.2 Sarjallistaminen

Sarjallistaminen (serialization) on tiedon muuntamista kuljetus- tai tallennuskelpoiseen muotoon. Yleisesti se tarkoittaa oliograafin (object graph) muuttamista biteiksi, minkä avulla olion tila voidaan tallentaa pysyvästi esimerkiksi tietokantaan tai siirtää tietoverkon yli. WCF määrittelee sarjallistamisen prosessiksi, jossa oliograafi muutetaan XML Infoset -tietomallin mukaiseksi ja käyttää sitä sisäisesti viestin muotona. XML Infoset ei määrittele tavallisen XML:n tapaan formaattia, jonka vuoksi WCF voi esittää viestin eri formaateissa.

WCF:ssä sarjallistamista varten on omat mekanisminsa rajapinnoille, viestille, olioille ja operaatioille. Palvelu merkitsee tietyn rajapinnan sopimukseksi ServiceContract -attribuutilla ja lupaa toteuttaa kyseisen rajapinnan sisällön. Rajapinnan operaatiot, jotka halutaan sopimukseen mukaan, merkataan OperationContract -attribuutilla. WCF-viestejä varten on oma MessageContract -attribuutti, jolla voi muokata muun muassa SOAP-viestin otsaketietoja. Sarjallistettavat olit merkitään DataContract –atribuutilla ja

olioiden jäsenmuuttujat DataMember –atribuutilla. Attribuutteja havainnollistaa esimerkki 4.1.

```
[ServiceContract(Namespace=”http://esimerkki.com/palvelu/”)]
[DataContractSerializer]
Public interface esimerkkiPalveluRajapinta {
        [OperationContract]
        Olio HaeOlio( int Id);
        [DataContract (Namespace=”http://esimerkki.com/olio”]
        Public class Olio {
                [DataMember]
                Public int Id;
        }
}
```

**Esimerkki 4.1 Sarjallistamisen merkkaavat attribuutit.**

WCF:ssä sarjallistamiseen käytettävän komponentti määritetään rajapinnan sopimuksen yhteydessä. Oletuksena jokainen protokolla käyttää sille sopivaa komponenttia, mutta tätä voidaan käsin vaihtaa tai luoda omia mukautettuja komponentteja. Sarjallistamisen toteuttavan komponentin valinta riippuu siitä, mitä formaattia käytetään, onko sarjallistettavat tietotyypit tavallisia vai mukautettuja ja halutaanko tukea teknologiariippumatonta toteutusta esimerkiksi tilanteissa jossa asiakas on toteutettu jollain muulla teknologialla kuin .Net-sovelluskehyksellä. Sarjallistamiskomponentit on kuvattu taulukossa 4.3.

**Taulukko 4.3 WCF:n sarjallistajat ja niiden ominaisuudet**

| Sarjallistaja | Viestin tyyppi | Ominaisuudet | Käyttö |
|---|---|---|---|
| DataContractJsonSerializer | JSON | JSON-tuki, sopimuksen jako | JavaScript- tai AJAX-pohjaiset Web-sovellukset |
| DataContractSerializer | XML | Sopimuksen jako | WCF-WCF-oletussarjallistaja |
| NetDataContractSerializer | XML | Sopimuksen ja-tyypin jako (type fidelity), tyyppitiedon lisäys | .Net Remoting, vain WCF-WCF |
| XMLSerializer | XML | Vanhojen tyyppien tuki, jotka ei tue DataContract-atribuuttia, mukautettavuus | Yhteensopivuus sovelluspalvelujen ja ei WCF-palveluiden kanssa |

### 4.2.3 Koodaus ja formaatit

Olio koodataan (encoding) halutulla formaatilla esimerkiksi tiedostoon ennen sen lähettämistä vastaanottajalle verkon yli, jotta tiedonsiirto kestäisi mahdollisimman vähän aikaa. WCF määrittelee koodauksen prosessiksi jossa WCF-viesti muutetaan biteiksi. Koodaaja (encoder) muuttaa sarjallistetun viestin tiettyy formaattiin. Koodaajan valinta riippuu käytettävästä protokollasta ja sitomisesta. Alla on kuvattu kolme yleisesti hyväksyttyjä tietoformaatteja .NET sovelluskehyksessä.

**XML** (eXtensible Markup Language) on ensimmäisiä yleisesti hyväksyttyjä olevia tietoformaatteja. Kieli on puurakenteista ja selkokielistä, ja mahdollistaa monipuolisen rakenteiden kuvailun. Dokumentti koostuu elementeistä, jotka voivat olla sisäkkäisiä ja elementtien järjestys määrittää dokumentin rakenteen. Esimerkki 4.2 havainnollistaa XML-dokumentin rakennetta.

```
<Person>
        <Forename>Teemu</Forename>
        <Surname>Teekkari</Surname>
        <Age>25</Age>
</Person>
```

**Esimerkki 4.2 XML**

XML-tiedostossa täytyy myös määritellä sen koodaus, jotta vastaanottaja osaa tulkita tietoa oikein. XML:n ja SOAP:n avulla voidaan määritellä tiedon formaatti ja koodaus. Näin määritelty viesti on helppo tulkita, mutta varsinkin pienillä tietomäärillä viesti saattaa sisältää ylimääräistä kuormaa (overhead), mikä hidastaa viestinvälitystä.

**JSON** (JavaScript Object Notation) on nimestään huolimatta täysin kieliriippumaton tietoformaatti. JSON:ssa tieto lähetetään avain-arvopareona, jonka vuoksi se on kevyempää kuin XML ja melko selkokielistä lukea. JSON-koodausta käytetään yleensä web-ohjelmissa ja sillä saavutetaan yhteentoimivuutta eri teknologioiden välillä. Esimerkki JSON avain-arvopareista:

```
{ ”forename”: ”Teemu”. ”surname”: ”Teekkari”, ”age”: 25 }
```

**Binäärimuotoisessa** koodauksessa viesti muutetaan binäärimuotoiseksi ennen lähettämistä. Binäärikoodaus on suorituskyvyltään optimaalinen ja sopii WCF:ssä .NET-sovelluskehyksellä toteutettujen ohjelmien tai komponenttien välille. TCP-protkolla tukee vain binäärimuotoista koodausta WCF:ssä. Binäärimuotoinen viesti ei ole selkokielistä, ja sellainen on siten vaikeampi kaapata ja hyödyntää.

### 4.2.4 Turvallisuus

Turvallisuus tiedon kuljetuksessa päätepisteiden välillä WCF:ssä jaetaan kahteen käsitteeseen: kuljetus- ja viestitkerroksen turvallisuuteen. Kuljetuskerroksen turvallisuus määrittelee tiedon suojauksen sen kulkiessa pitkin tietoverkkoa riippumatta tiedon sisällöstä. Viestikerroksen turvallisuus taas määrittää viestikohtaisen suojauksen riippumatta kuljetusmekanismista. Viestikerroksen turvallisuus on tehty varmistamaan yksittäisten viestien eheyden ja yksityisyyden esimerkiksi salauksen tai allekirjoituksen avulla, ja kaikki turvallisuuteen liittyvät asetukset kulkevat SOAP-viestin metatiedon mukana. WCF asettaa oletuksia turvallisuusasetuksiin valitun sitomisen ja protokollan mukaan.[22]

**Taulukko 4.4 Kuljetus- ja viestikerroksen turvallisuuominaisuuksia.[38]**

| | Kuljetuskerros | Viestikerros |
|---|---|---|
| Salaus | Ei valittavissa, kuljetusstandardi-riippuvainen | Ei mitään / Salaus / Salaus ja allekirjoitus |
| Salausalgoritmi | - | Useita vaihtoehtoja epäsymmentrisestä symmetriseen |
| Autentikointi | Ei mitään / Basic / NTLM / Windows / Sertifikaatti | Ei mitään / Windows / Käyttäjätunnus / Sertifikaatti / Tiketti (security token) |
| Viestien striimaus | Kyllä | Ei |
| Turvallisuuden kattavuus | Vain päätepisteestä seuraavaan pisteeseen | Päätepisteestä päätepisteeseen |

Viestin salaus ja allekirjoittaminen vaikuttaa sarjallistamisen kestoon merkittävästi. Salauksen voi tehdä ennen verkon yli lähettämistä joko tiedonsiirtokerroksessa tai viestikerroksessa. TCP-protokollan vaihtoehtoja ovat tiedonsiirtokerroksessa TSL ja SPNego ja viestikerroksessa SOAP-protkokollan teknologiat. Myös näiden yhdistelmä on olemassa. Taulukossa 4.4 on esitelty WCF:n kuljetus- ja viestikerroksen turvallisuusominaisuuksia.[22][39]

Suorituskykyyn vaikuttaa moni turvallisuuteen liittyvä asetus. Kuljetuskerroksen turvallisuuden käyttö ei poissulje viestin salausta, vaan se voidaan tehdä viestikerroksessa vaikka turvallisuusasetukseksi olisi valittu kuljetus. Salauksen tarve riippuu käyttökohteesta ja järjestelmän turvallisuusvaatimuksista. Esimerkiksi sisäverkossa toimivan järjestelmän tiedonkulun turvallisuusvaatimukset eroaa tietoverkon yli toimivasta järjestelmästä.[38]

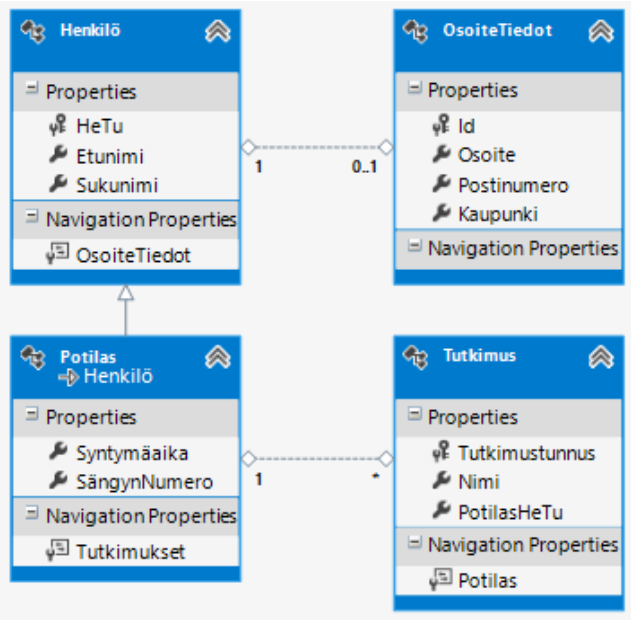
## 4.3 Tietokantaoperaatiot ja tietomalli

Tässä kohdassa kuvataan miten Entity Framework:n tarjoamilla teknologioilla käytetään tietokantaa ja millaisia hakuja entiteettimallia vasten voi tehdä. Esitellään valittujen teknologioiden aiheuttamat muutokset järjestelmän toteutuksessa ja optimointimahdollisuudet kyselyissä. Entiteettejä välitettäessä verkon yli kyselyjen merkitys korostuu, koska kyselyn tulos vaikuttaa suoraan siihen kuinka paljon tietoa ja assosiaatioita entiteetissä on mukana. Kyselyn palauttama entiteettigraafin käsittely ennen verkon yli lähettämistä voi olla tarpeen, sillä entiteetit saattavat sisältää metatietoja tai ylimääräistä lisäkuormaa. Kaiken lähtökohtana on hyvin suunniteltu tietokannan käsitteellinen malli, entiteettimalli.

Entiteettimallia ja sen suunnitteluratkaisuja, kuten vierasavaimia ja periyttämistä käsitellään alakohdassa 4.3.1, jo. Alakohdassa 4.3.2 esitellään entiteettimallin kyselyä tukevat kielet ja niiden ominaisuuksia. Alakohdassa 4.3.3 esitetään ObjectService -komponentin tarjoamia palveluja kuten muutosten jäljittämistä, laiskaa lataamista ja vertaillaan kahden kontekstin käyttöä, joiden sisällä tietokannan operaatiot tapahtuvat Entity Framework:ssä. Alakohdat 4.3.1 – 4.3.3 perustuvat lähteeseen [37], ellei toisin mainita.

### 4.3.1 Entiteettimalli

Entiteettimalli on asikaspuolen tietomalli ja Entity Framework:n sydän. Entiteettimallissa yhdistää asiakas- ja palvelukerroksen tieto-oliot (busines objects) ja tietokannan taulut yhteiseksi koko järjestelmän kattavaksi tietomalliksi. Esimerkki entiteettimallista graafisella työkalulla tehtynä on kuvassa 4.7.




**Kuva 4.7 Esimerkki entiteettimallista.**

Entiteettimalli tallentuu kolmeen XML-tiedostoon. Käsitteellistä mallia varten on oma XML-pohjainen kieli, jolla kuvataan entiteetit, assosiaatiot ja toiminnot. Entity Framework käyttää tiedoston metatietoja kuvaamaan entiteetit ja niiden assosiaatiot tietolähteeseen. Myös fyysistä mallia varten on oma XML-kuvaus tietokannan rakennetta varten. Kolmas tiedosto yhdistää mallit toisiinsa kuvauksella, jonka metatietoja Entity Framework käyttää ajon aikaisesti kääntämään käsitteelliseen malliin kohdistuneita kyselyitä tietokantakohtaisiksi käskyiksi.

Periyttäminen on Entity Framework:ssa mahdollista. Sen avulla entiteetit voidaan jakaa järkeviin osiin. Kantaluokka sisältää kaikille yhteiset ominaisuudet ja lapsiluokat niille erityiset ominaisuudet. Luokan periytymisen perusteella voidaan olioista helposti tunnistaa ketkä ovat potilaita ja ketkä esimerkiksi henkilökuntaa. Tehokkuuden kannalta on tärkeää, että voidaan hakea tietty potilas potilaista, eikä kaikkia tietokannan henkilöitä tarvitse käydä läpi.

Periytettyjä olioita ei voi kuitenkaan poistaa poistamatta kantaluokan oliota, mikä esimerkkimallissa voisi joskus olla tarpeellista. Periytettyä ja kantaluokkaa ei siis voi erottaa toisistaan.[37] Periytetyt luokat pitää esitellä sarjallistamista varten, sillä WCF tunnistaa oletuksella vain kantaluokat. Jos asiakassovellus pyytää palvelulta henkilöitä, ei WCF vastaanottaessa viestiä asiakaspäässä osaa sarjallistaa periytettyjä luokkia. Tätä varten lapsiluokat voidaan merkitä KnowType-atribuutilla kantaluokan entiteetin kuvaksessa.[40]

Entiteettimalli esittelee vierasavain-assosiaation (navigation property) rinnalle vieras avain-ominaisuuden (foreing key property). Tavallinen assosiaatio on viite toiseen tauluun, kun taas vieras avain ominaisuus pitää tallessa toisen taulun pääavaimen. Esimerkkimallissa Tutkimus-taulun PotilasHeTu on vierasavain ominaisuus. Ominaisuus on kätevä tilanteissa, jossa esimerkiksi listattaisiin potilaille tehtyjä tutkimuksia vain henkilötunnuksen perusteella. Tällöin tietokannasta tarvittaisiin hakea vain tutkimustaulut, sillä se sisältäisi jo tiedon potilaan henkilötunnuksesta.

### 4.3.2 Kyselykielet

Tehokkuuden kannalta on tärkeää hakea vain oleellinen tieto tietokannasta. Kuvassa 4.7 esitellyn entiteettimallin mukaisen tiedon hakuun tarvitaan kysely, joka sisältää paljon sisäisiä ja ulkoisia liitoksia, jos haluttaisiin potilaan tietoihin mukaan siihen liittyvät lähetteet. Potilaan tietojen hakua varten entiteettimalliin voidaan tehdä kyselyjä kolmella Entity Framework:n tukemalla kyselykielellä. Seuraavaksi esitellään käytettävissä olevat kyselykielet, niiden syntaksia ja ominaisuuksia.[37]

**SQL** tai tavallinen SQL (plain SQL) on kyselykielistä yksinkertaisin ja matalatasoisin. SQL:n syntaksi on tuttua mutta työlästä kirjoittaa. Tavallinen SQL-kysely on suoritus-

kykyisin verrattuna muihin kyselykieliin, sillä Entity Framework ei joudu muokkaamaan kyselyä sen ollessa valmiiksi SQL-muodossa. Tietokannan tai tietomallin muuttuessa kyselyjä joudutaan päivittämään paljon käsin. Kyselykieli on myös altis näppäinvirheille ja injektiohyökkäyksille.[35]

Esimerkissä 4.3 on SQL-kielellä toteutettu kysely, joka hakisi kaikki potilaat ja niihin liittyvät osoitetiedot kuvan 4.7 mukaiseen tietokantaan, olettaen että taulujen yhdistävä tekijä tietokannassa olisi Henkilö-taulun pääavain.

```
string kysely ”SELECT     Potilas.*, OsoiteTiedot.*, Henkilö.*
                  FROM              Henkilö
                  INNER JOIN OsoiteTiedot ON Henkilö.HeTu = Osoitetiedot.HeTu
                        INNER JOIN Potilas ON Henkilö.HeTu = Potilas.Hetu”
```

**Esimerkki 4.3 SQL-kysely.**

Entity Framework mahdollista tavallisten SQL-kyselyjen tekemisen kontekstin sisällä:
`context.Henkilö.SqlQuery(kysely);`

**Entity SQL** käyttä SQL-tapaista syntaksia kyselyihin. Kyselykieli tukee entiteettimallia ja sen ominaisuuksia, kuten periyttämistä ja assosiaatioita. Entity SQL käyttää Object services –komponenttia muuttamaan kyselyn parametrisoiduksi SQL-kyselyksi ja kyselyn tuloksen takaisin ObjectQuery-paluuarvoisiksi entiteeteiksi. Kyselykieltä voidaan käyttää ympäri .NET-sovelluskehyksen toteutuskielestä riippumatta. Esimerkissä 4.4 on kuvattu edellisen kyselyn toteutus Entity SQL -kielellä.

```
string kysely = “SELECT h, h.LisäTiedot FROM OFTYPE(Henkilö, Malli.Potilas) as h”;
ObjectQuery<Potilas> potilaat = context.CreateQuery<Potilas>(kysely);
```

**Esimerkki 4.4 Entity SQL –kyselyn toteutus.**

**LINQ to Entities** kyselyiden syntaksi samantapaista, mutta helpompaa oppia kuin Entity SQL:ssä. Kyselyt voidaan kohdistaa entiteettimallin käsitteelliseen malliin. LINQ to Entities käyttää Object services –komponenttia muuttamaan kyselyn parametrisoiduksi SQL-kyselyksi ja kyselyn tuloksen takaisin entiteeteiksi. LINQ to Entities –kyselykieltä voi käyttää vain Entity Framework:n kanssa. Seuraavaa kyselyä on helppo ylläpitää, sillä käsitteelliseen malliin kohdistunut muutos muuttaa myös kyselyä. Esimerkkikyselyn toteutus LINQ to Entities -kielellä:

```
var potilaat = from h in context.Henkilo.OfType<Potilas>().Include(it => it.Lisätiedot) select h;
```

### 4.3.3 Entiteettimallin palvelut ja kontekstit

ObjectService-komponentin tehtävät voidaan jakaa seitsemään kategoriaan: Kyselyjen käsittely, olioiden ilmentäminen, olioiden hallinta, olioiden suhteiden hallinta, olioiden tilan hallinta, muutosten päivittäminen tietokantaan, sekä sarjallistaminen ja tiedon sitominen. Olioiden hallintaan liittyy niiden lataaminen välimuistiin kontekstin sisällä ja

olioihin tapahtuvien muutosten jäljittäminen välimuistin ja pysyvän muistin (tietokannan) välillä. Tietokantahaun yhteydessä konteksti tallentaa välimuistiin entiteetin alkuperäiset arvot ja tilan ObjectStateEntry-luokkana.

Tilan hallinta verkon yli kommunikoivissa järjestelmissä aiheuttaa haasteita, sillä ObjectService -komponentin konteksti loppuu palvelun ulkopuolelle. ObjectContext-konteksti, ObjectStateEntry-luokka ja olioiden hallintaluokka eivät ole sarjallistettavia. Ilman mukautetun ohjelmakoodin kirjoitusta ei muutosten jäljitystä ja entiteetin tilaa voi sarjallistaa ja kuljettaa verkon yli. Aiemmin kuvatuilla POCO- ja self-tracking entites -entiteettityypeillä tilan sarjallistaminen on kuitenkin mahdollista toteuttaa. Jos entiteettiin tai entiteettipuuhun kohdistuu muutoksia asiakassovelluksessa, vaihtuu entiteetin tilaksi muutettu. Muutos vaikuttaa myös kaikkiin entiteetin assosiaatioiden päissä oleviin entiteetteihin.

Kontekstien sisällä entiteettejä ja niihin liittyviä entiteettejä voidaan ladata eri tavoilla. Tavan valinta riippuu operaation tarkoituksesta ja tehokkuusvaatimuksista. Konteksti tarjoaa seuraavat tavat:

- Projektio – ladataan vain tietyt osat entiteeteistä. Tämä parantaa suorituskykyä verrattuna koko entiteetin lataukseen. Projektion joutuu käsin tallentamaan takaisin.
- Innokas lataus (eager loading) – ladataan entiteettiin liittyvät valitut assosiaatiot kyselyn yhteydessä esimerkiksi Include-käskyllä.
- Laiska lataus – ladataan entiteettiin liittyvät assosiaatiot vasta kun assosiaatiota käytetään.
- Eksplisiittinen lataus – ladataan entiteettiin liittyvät assosiaatiot vasta kun ne eksplisiittisesti haetaan esimerkiksi Load-käskyllä.

Laiskan ja eksplisiittisen latauksen kohdalla tietokantahakujen määrä nousee, mitä useampi entiteetti liittyy suoritettaan operaatioon. Toisaalta usean Include-käskyn kanssa innokas lataus saattaa olla hidasta monimutkaisten tietokantojen kanssa.

Konteksteja on Entity Framework:n versiosta 4.1 eteenpäin kaksi. Uusi `DbContext` eroaa vanhasta ObjectContext-luokasta taulukon 4.5 mukaisesti. `DbContext` ei tue self-tracking entities -tyyppisiä entiteettejä, mikä hankaloittaa kerrosarkkitehtuurillisten järjestelmien muutosten jäljitystä. Palvelun operaatioita suunnitellessa tämä voidaan kuitenkin kiertää esimerkiksi suunnittelemalla entiteettien päivitys-, lisäys- ja poisto-operaatioille omat funktionsa.[41]

Entiteettien tallentamiselle tietokantaan kontekstit tarjoavat SaveChanges()-käskyn. SaveChanges tekee muutokset annetussa kontekstissa yhtenä transaktiona, mutta ei takaa operaatioiden tiettyä järjestystä. Tämä tulee ottaa huomioon suunnitellessa operaatioita, joissa voi olla samassa transaktiossa lisäys-, päivitys- tai poisto-operaatioita.[41]

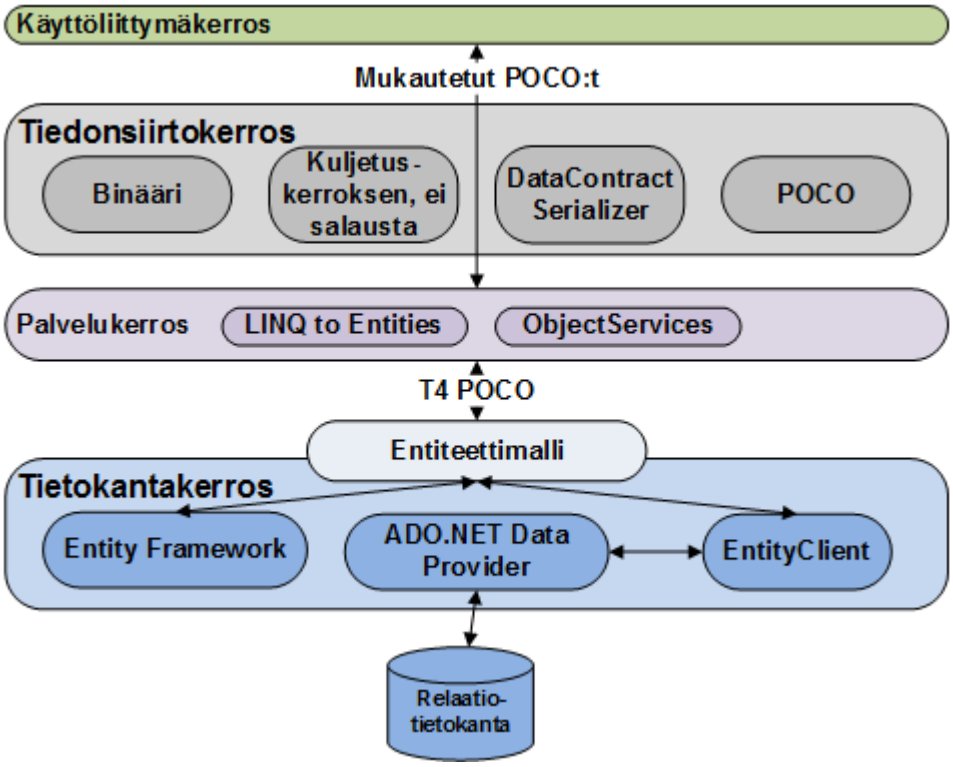
**Taulukko 4.5 Kontekstien eroja.[41]**

| DbContextin ominaisuus | EF 4.0 vastaava ominaisuus | Tarkoitus | DbContextin hyöty |
|---|---|---|---|
| DbContext | ObjectContext | Kuvaa session tietokannan kanssa. Tarjoaa kysely, muutosten jäljitys ja tallennus -operaatiot | Yksinkertaistaa useita ominaisuuksi |
| DbSet | ObjectSet | lisäys-, poisto- ja liittämisoperaatiot | Yksinkertaistaa ObjectSet:n tarjoamia ominaisuuksia |
| DbQuery | ObjectQuery | kyselyt tietokantaan | Toiminnallisuus DbSet-luokassa, joten voidaan kutsua entiteettejä suoraan. |
| Change Tracker API | ObjectStateManager | Muutostenjäljitysoperaatiot | Yksinkertaistaa ohjelmointirajapintaa |
| Validation API | ei saatavilla | Tarjoaa automaattista tiedon validointia tietokantakerroksessa | Uusi ominaisuus |

## 4.4 Valitut ratkaisut

Valittuihin suunnitteluratkaisuihin vaikutti suorituskyvyn ohella myös helppokäyttöisyys ja muunneltavuus, sillä toteutettavaa järjestelmään tehdään ketterästi ja järjestelmän määrittely saattaa muuttua läpi kehitystyön. Tästä syystä esimerkiksi tietokantamuutokset on tärkeää voida toteuttaa niin, että mahdollisimman vähän ohjelmakoodia jouduttaisiin refaktoroimaan. Valitut suunnitteluratkaisut on esitelty kuvassa 4.8.

Entity Framework:n päälle kasattu tietokantakerros käyttää ADO.NET Data Provider, EntityClient, ja ObjectServices –komponentteja tietokannan käsittelyyn. Järjestelmän tietomalli suunnitellaan käsitteellisen mallin avulla ja siitä luodaan olio-relaatiokuvauksella entiteettimalli. Entiteettimallista luodaan T4-mallin (T4 Template) avulla automaattisesti POCO-tyyppisiä tiedonhakuolioita. Näitä entiteettejä käytetään koko järjestelmän tasolla: Entiteettimalliin kohdistuneet kyselyt palauttavat POCO-entiteettejä, ja entiteettimallin mukaiset entiteetit näkyvät palvelu- ja käyttöliittymäkerroksille.[42]




**Kuva 4.8 Valitut suunnitteluratkaisut.**

Palvelukerroksessa ObjectServices-komponentti muuttaa LINQ to Entities -kyselykielellä toteutettavat tietokantakyselyt entiteeteiksi. Kontekstiksi valittiin DbContext sen yksinkertaisuuden vuoksi. DbContext-kontekstia käyttämällä myös POCO-entiteettien luonti ja kysely on vaivattomampaa. Luku operaatiot toteutetaan ilman muutosten jäljitystä, jos mahdollista, sillä muutosten jäljitys tekee operaatiosta hieman raskaamman, eikä haku-tyyppisten funktioiden kohdalla sitä tarvita. Tietokantahaussa käytetään innokasta hakua vähentämään hakujen määrää. Yleensä palvelun hakufunktiot palauttavat tiedon käyttöliittymälle heti haun jälkeen, jolloin innokas lataus on tehokas tapa.[42]

Tiedonsiirtokerroksessa tiedonsiirto-olioina toimivat samat POCO-entiteetit, joita voidaan mukauttaa. Joskus on tarpeellista liittää entiteettiin jotain käyttöliittymäkerroksen tarvitsemaa tietoa, mutta tarpeellisen tiedon liittäminen assosiaationa kasvattaisi liikaa hyötykuormaa. Tätä varten entiteettejä voidaan mukauttaa laajennos-luokilla. Samoin entiteettejä voidaan pienentää projektion tavoin luomalla laajennos-luokkaan karsittu versio entiteetistä tiedonsiirtoa varten. Entiteetit sarjallistetaan WCF:n DataContractSerializer-luokalla ja koodataan binäärimuotoon TCP-protokollalla kuljetettavaksi. Koska toteutettava järjestelmä toimii PET-keskuksen sisäverkossa ja tehokkuus on yksi tärkeimmistä vaatimuksista, päätettiin käyttää TCP-protokollalle suotuisaa kuljetuskerroksen turvallisuutta ja jättää salaus ja allekirjoitus pois WCF-viestistä.[37]

# 5 TULOKSET JA ARVIOINTI

Tässä luvussa arvioidaan ja vertaillaan asetettujen kriteerien ja toteutettavan järjestelmän vaatimusten pohjalta valittuja ratkaisuvaihtoehtoja muihin esiteltyihin ratkaisuihin. Arviointi etenee konkreettisista suunnitteluratkaisuista ja komponenteista abstrakteihin ohjelmistoarkkitehtuureihin ja teknologiavalintoihin. Työn pääaiheena on tiedon kuvaus ja sarjallistaminen, joten arvioinnin täsmällinen osuus kohdistuu juuri tiedonsiirto- ja tietokantakerroksiin. Käyttöliittymä- ja palvelukerroksen suunnitteluratkaisuja ja järjestelmän arkkitehtuuria vertaillaan suurpiirteisemmin. Vertailua vaihtoehtojen kesken tehdään pääosin toteutettavan järjestelmän kannalta.

Kohdassa 5.1 kuvataan arviointikriteerit tarkemmin ja esitellään testausympäristö. Kohdassa 5.2 esitellään toteutettuun PET ERP -järjestelmään kohdistettujen testien tuloksia. Kohdassa 5.3 arvioidaan ensin tiedon sarjallistamis- ja kuvausratkaisuja testeihin ja kirjallisuuteen perustuen ja seuraavaksi arvioidaan järjestelmän arkkitehtuurivaihtoehtoja. Lopuksi arvioidaan kuinka valitut ratkaisut vastaavat järjestelmälle asetettuihin vaatimuksiin. Kohdassa 5.4 tarjotaan lukijalle jatkokehitysideoita ja esitellään mahdollisia vaihtoehtoja työn ratkaisuille. Lisäksi kerrotaan työn aihepiirin ulkopuolelle rajattuja näkökohtia, joita tulee ottaa huomioon samankaltaisten järjestelmien toteutuksessa.

## 5.1 Arviointikriteerit ja testiympäristö

Arviointikriteerit perustuvat järjestelmälle asetettuihin vaatimuksiin ja arviointiin toteutuksen yleiskäyttöisyydestä palvelupohjaisissa järjestelmissä. Seuraavat arviointikriteerit ovat työn keskeisiä, ja niitä käytetään arvioimaan niin korkean tason teknologioita ja arkkitehtuureita kuin yksittäisiä, matalan tason suunnitteluratkaisujakin.

- **Suorituskyky** on tärkein arviointikriteeri ja yksi järjestelmän haasteellisimmista vaatimuksista. Tietokantahakujen pitää olla nopeita ja sarjallistamisen tehokasta sadan yhtäaikaisen asiakassovelluksen käytössä.
- **Muunneltavuus** vaikuttaa teknologian tai suunnitteluratkaisun kykyyn sopeutua määrittelyn tai toteutuksen muutoksiin tai uusiin ominaisuuksiin. Tarkoittaa esimerkiksi muutostarvetta tietokantakyselyyn tietomallin luokkaan kohdistuneen muutoksen jälkeen.
- **Yleiskäyttöisyys** vastaa järjestelmän vaatimuksissa integroitumista muihin järjestelmiin ja käyttäjien autentikointia. Järjestelmään pitää pystyä integroimaan erilaisia laitteita ja järjestelmiä ja sen pitää tukea erilaisia käyttäjän autentikointitapoja. Yleiskäyttöisyys tarkoittaa myös toteutuksen tai komponentin yleiskäyt-

töisyyttä, jos se esimerkiksi siirretään toisella teknologialla toteutettuun järjestelmään tai komponenttiin.
- **Toteutuksen nopeus ja helppous** vaikuttavat kehittäjän työhön. Määrittää teknologioiden opetteluun ja ohjelmointiin kuluvaa aikaa, sekä ohjelmakoodin määrää ja helppolukuisuutta.

Testausympäristönä toimi järjestelmän kehitysympäristö. Testit toteutettiin testausta varten tehdyllä konsolisovelluksella, joka toimi asiakassovelluksen roolissa. Konsolisovelluksella voitiin luoda monta instanssia asiakassovelluksesta, jotka kirjautuivat kuuntelemaan palvelimen PER ERP -palvelua. Käynnistyessään asiakassovellus kutsui PET ERP -palvelun valittua operaatiota annetun lukumäärän verran. Palvelinta ajettiin samalla koneella IIS:n isännöimänä ja tietokantakutsut suoritettiin Entity Framework 5.0 -ohjelmistokehyksen version entiteettimallia vasten.

Tietokantaan luotiin vuoden aikataulu siten, että vuoden jokaiselle päivälle oli aikataulutettu 72 tutkimusta (aikavälille 08:00-16:00, joka tunnille 9 tutkimusta). Tällöin aikataulun sisältö tietokannassa on noin 35 000 kuvaustutkimusta ja 300 000 kuvaukseen ja lääkeainetuotantoon liittyvää vaihetta. Aikataulusta tehtiin kohdejärjestelmän normaalikäyttöä ruuhkaisempi versio.

## 5.2 Testien tuloksia

Konsolisovellus loi käynnistyessään kymmenen asiakassovellusta, jotka kutsuivat palvelun yhden päivän tutkimuksia hakevaa operaatiota viisi sataa kertaa 100 – 2000 millisekunnin välein. Kyselyistä mitattiin kestot tietokantakyselyn ja koko operaation osalta, josta voitiin laskea myös sarjallistamiseen ja tiedon kulkuun mennyt aika erotuksena. Mittaukset toistettiin vaihtamalla asetuksia, ja näin saatiin viiden tuhannen mittauksen keskiarvo jokaiselle tutkittavalle ratkaisuvaihtoehdolle. Ensin testattiin erilaisia kyselyjä tietokantaan WCF-asetusten ollessa: *TCP-protokolla kuljetuskerroksen turvallisuudella ilman salausta ja binäärisarjallistamisella*. Testit 1-4 toteutettiin DbContext-luokkaa käyttäen ja testi 5 ObjectContext-luokkaa käyttäen.

1. Innokas lataus, LINQ kysely muotoa:

```
List<ProjectBlock> blocks = db.Block.OfType<ProjectBlock>().Where(it => it.Time >=
        startTime && it.Time <= endTime).Include(it => it.Project).Include(it
        => it.Device).Include(it => it.Reservation).ToList();
```

2. Eksplisiittinen lataus, LINQ-kysely muotoa:

```
var blocks = db.Block.OfType<ProjectBlock>()
        .Where(it => it.Time >= startTime && it.Time <= endTime);
foreach (ProjectBlock block in blocks)
        { db.Entry(block).Reference(it => it.Project).Load(); }
```

3. Innokas lataus, paloiteltu kysely kahteen osaan, LINQ-kysely muotoa:

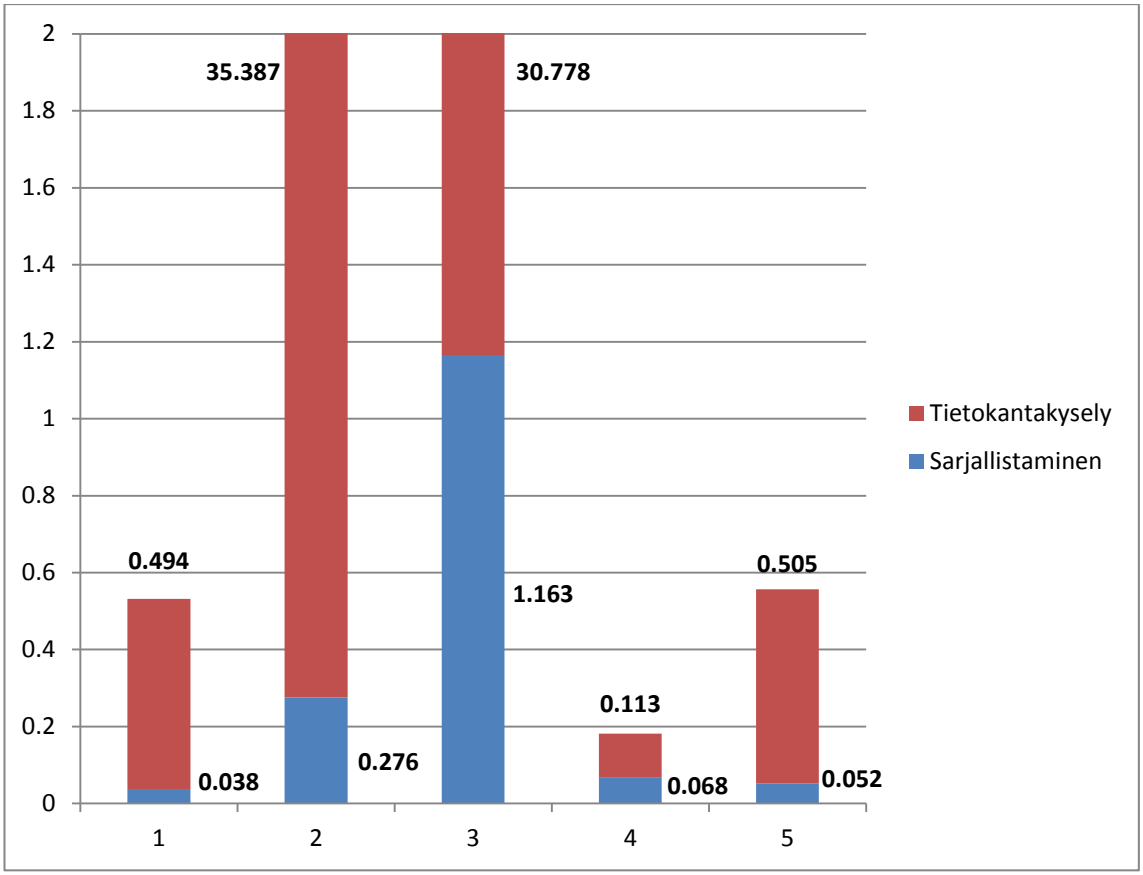
```
var blocks = db.Block.OfType<ProjectBlock>().Where(it => it.Time >= startTime &&
        it.Time <= endTime).Include(it => it.Device).Include(it => it.Reservation);
foreach (ProjectBlock block in blocks)
        { block.Project = db.Project.Find(block.ProjectId); }
```

4. Innokas lataus, muutosten jäljitys pois käytöstä, LINQ-kysely muotoa:

```
List<ProjectBlock> blocks = db.Block.AsNoTracking().OfType<ProjectBlock>()
        .Where(it => it.Time >= startTime && it.Time <= endTime).Include(it =>
        it.Project).Include(it => it.Device).Include(it => it.Reservation).ToList();
```

5. Innokas lataus, kontekstina ObjectContext, kysely osaksi Entity SQL-kyselykieltä:

```
string kysely = "SELECT VALUE p FROM OFTYPE(Block, PetErpDataModel.ProjectBlock)
        AS p WHERE p.Time >= @start && p.Time <= @end";
var objctx = (db as IObjectContextAdapter).ObjectContext;
var blocks = objctx.CreateQuery<ProjectBlock>(kysely, new ObjectParameter("start",
        startTime), new ObjectParameter("end", endTime)).Include(it =>
        it.Project).Include(it => it.Device).Include(it => it.Reservation).ToList();
```



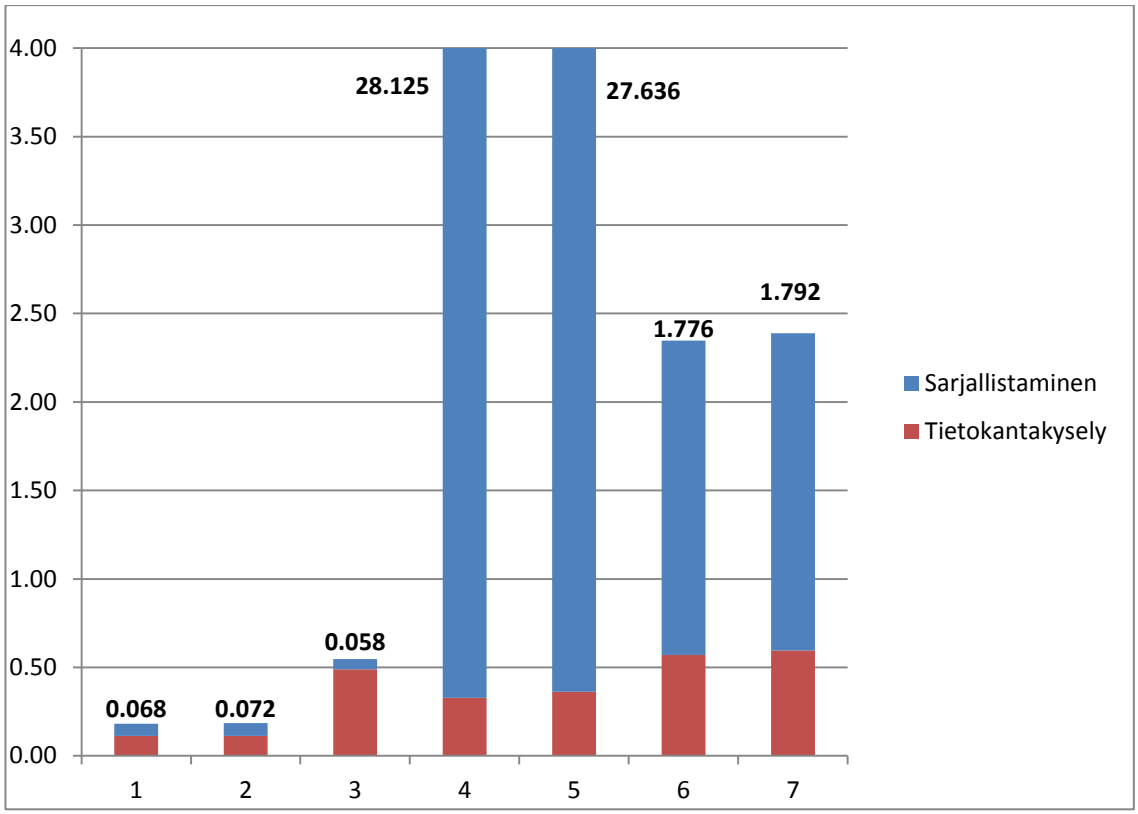

**Kuva 5.1 Erilaisten tietokantakyselyjen kestot ja vaikutukset sarjallistamiseen.**

Erilaisten tietokantakyselyjen kestot ja vaikutukset sarjallistamiseen on esitelty kuvassa 5.1. Vaaka-akselilla on numeroidut tekniikat ja pystyakselilla aika sekunteina. Kuvasta nähdään kuinka suurella tietomäärällä eksplisiittinen lataus (2) ja paloiteltu kysely (3) hidastavat kyselyn käyttökelvottomaksi. Miltei yhtäaikaiset kutsut muilta asiakassovelluksilta hidastavat toistensa tietokantakyselyjä, mikä näkyy eniten testeissä 2 ja 3. Kyselykielten LINQ to Entities ja Entity SQL erot ovat hyvin pieniä, mikä toisaalta voi johtua myös siitä, että Entity SQL-kyselyn innokas lataus tehdään LINQ-metodien avulla. Lyhin kesto kyselyyn saavutetaan, kun muutosten jäljitys otetaan pois käytöstä. Tämä kuitenkin näyttää hidastavan hieman sarjallistamista.

Seuraavaksi testattiin WCF:n turvallisuusasetusten vaikutusta sarjallistamisen kestoon. Edellisistä testeistä valittiin 1-kohdan LINQ to Entities -kysely innokkaalla latauksella. TCP-protokollan viesti- ja kuljetuskerroksen turvallisuuksia testattiin salauksen kanssa ja ilman, sekä erilaisilla salausalgoritmeilla seuraavasti:

1. Kuljetuskerroksen turvallisuus, ei salausta, ei muutosten jäljitystä.
2. Kuljetuskerroksen turvallisuus, salaus ja allekirjoitus, ei muutosten jäljitystä.
3. Kuljetuskerroksen turvallisuus, salaus ja allekirjoitus, muutosten jäljitys päällä.
4. Viestikerroksen turvallisuus, salausalgoritmina Basic128, kuljetuskerroksen salaus ja allekirjoitus, ei muutosten jäljitystä.
5. Viestikerroksen turvallisuus, salausalgoritmina Basic128, ei kuljetuskerroksen salausta, ei muutosten jäljitystä.
6. Viestikerroksen turvallisuus, salausalgoritmina Basic256, ei kuljetuskerroksen salausta, muutosten jäljitys päällä.
7. Viestikerroksen turvallisuus, salausalgoritmina Basic128, ei kuljetuskerroksen salausta, muutosten jäljitys päällä.

Sitomisen turvallisuusasetusten vaikutukset sarjallistamiseen ja tietokantakyselyihin on esitelty kuvassa 5.2. Vaaka-akselilla on numeroidut tekniikat ja pystyakselilla aika sekunteina. Sarjallistaminen kuljetuskerroksen turvallisuudella on monin kerroin viestikerroksen turvallisuutta tehokkaampaa. Muutosten jäljityksen poistaminen hidastaa viestikerroksen sarjallistamista merkittävästi. Myös kuljetuskerroksessa muutosten jäljityksen poistaminen hidastaa hieman sarjallistamista, mutta nopeuttaa tiedonkulun kokonaiskestoa huomattavasti. Salauksen ja allekirjoituksen merkitys kuljetuskerroksen turvallisuusasetuksilla näyttää hidastavan sarjallistamista vain noin neljä millisekuntia. Ero ei ole merkittävä näin suuren oliograafin hakemisessa kerralla, mutta saattaa vaikuttaa pienempien tietokantakyselyjen kestoon suhteessa enemmän. Salausalgoritmeista Basic256 osoittautui hieman yllättäen nopeammaksi kuin Basic128.




**Kuva 5.2 WCF:n turvallisuusasetusten vaikutus sarjallistamisen kestoon.**

## 5.3 Ratkaisujen arviointi

Tehtyjen testien perusteella löydettiin hyvä ratkaisu PET ERP -järjestelmän kuvausaikataulun päiväkohtaiselle hakemiselle. TCP-protokolla binäärimuotoisella koodauksella ja sarjallistamisella, kuljetuskerroksen turvallisuudella ja ilman salausta osoittautui nopeimmaksi tavaksi. Tietokantakysely toteutettiin LINQ to Entities –kyselykielellä ilman muutosten jäljitystä, ja näin koko operaation kestoksi asiakassovellukselta tietokantaan ja takaisin saatiin keskimäärin 0,181 sekuntia. Operaation kesto on järjestelmän tietokannalla varsin hyvää tasoa, sillä operaatio on myös yksi PET ERP -järjestelmä raskaimmista operaatioista.

Tiedon sarjallistamis- ja kuvausvaihtoehtoja on arvioitu kriteerien pohjalta taulukossa 5.1. Vaihtoehdot on esitelty järjestyksessä matalan tason vaihtoehdoista korkean tason ratkaisuihin. Jokainen teknologia, kyselykieli tai suunnitteluratkaisu on arvioitu kriteerien pohjalta asteikolla: hyvä, normaali, huono. Tähdellä (*) merkityt tarkoittavat tilannetta, jossa entiteettien luodaan automaattisesti mallin avulla.

WCF:n sitomisen ja protokollan valinnassa netTcpBinding on luotu juuri PET ERP -järjestelmän kaltaisille sisäverkossa toimiville hajautetuille WCF-ohjelmille.[23] HTTP-protokollan suorituskyky ja eri sitomisvaihtoehdot eivät pärjää vertailussa netTcpBinding-sitomiselle. Myös kaksisuuntaisuuden ansiosta TCP täyttää vaatimuksen reaaliaikaisesta muutoksista tiedottamisesta.[43]

**Taulukko 5.1 Tiedon sarjallistamis- ja kuvausvaihtoehtojen arviointi.[28]**

| Teknologia /suunnitteluratkaisu | Suorituskyky | Muunneltavuus | Yleiskäyttöisyys | Toteutuksen nopeus ja helppous |
|---|---|---|---|---|
| LINQ to Entities | Normaali | Hyvä | Huono | Hyvä |
| Entity SQL | Normaali | Normaali | Normaali | Huono |
| SQL | Hyvä | Huono | Hyvä | Huono |
| EntityObject | Huono | Huono | Huono | Hyvä |
| POCO | Hyvä | Normaali | Normaali | Hyvä * |
| POCO proxy | Normaali | Huono | Huono | Hyvä * |
| Self-tracking Entities | Normaali | Huono | Huono | Normaali * |
| ADO.NET | Hyvä | Huono | Normaali | Huono |
| LINQ to SQL | Normaali | Normaali | Normaali | Normaali |
| Entity Framework | Huono | Hyvä | Hyvä | Hyvä |

Olio-relaatio-kuvausteknologioissa suorituskyvykkäin kyselyjen suhteen on ADO.NET, mutta sen olio-relaatio-kuvauksessa on puutteita ja se on muunneltavuudeltaan ja yleiskäyttöisyydeltään huonoin. Tehtyjen testien perusteella Entity Framework osoittautui riittävän suorituskykyiseksi, ja varaa optimointiin vielä jäi esimerkiksi puhtaiden SQL-kyselyjen, ennalta luotujen näkymien ja automaattisesti käännettyjen kyselyjen muodossa. Myös LINQ to SQL -komponenttikirjasto kärsii puutteellisesta olio-relaatiokuvauksesta, eikä pärjää muunneltavuudessa tai yleiskäyttöisyydessä Entity Framework -ohjelmistokehykselle.[28][42]

Entity Frameworkin omasta EntityObject –luokasta periytetyt entiteetit sisältävät paljon metatietoa, jonka takia POCO:jen lähettäminen verkon yli on tehokkaampaa. Myöskään sarjallistaminen Entitiy Framework:lla ei ole välityspalvelimien (proxy) kanssa kannattavaa tässä järjestelmässä, sillä tällöin verkon yli kulkee paljon ylimääräistä tietoa, mikä hidastaa järjestelmän toimintaa. Tämän takia EntityObject-, self-trackin Entities ja POCO proxy -luokat karsiutuivat valituista ratkaisuista entiteettien tyypeiksi. [35]

Seuraavaksi arvioidaan järjestelmän arkkitehtuuri- ja teknologiavaihtoehtoja, jotka esiteltiin luvussa 3. Esitellyt vaihtoehdot on arvioitu taulukossa 5.2 samaan tapaan kuin taulukossa 5.1. Ensin luetellaan tiedonsiirtokerroksen vaihtoehdot, seuraavaksi palvelukerroksen vaihtoehdot ja lopuksi käyttöliittymäkerroksen vaihtoehdot.

**Taulukko 5.2. Järjestelmän teknologioiden ja suunnitteluratkaisujen arviointi.**

| Teknologia /suunnitteluratkaisu | Suorituskyky | Muunneltavuus | Yleiskäyttöisyys | Toteutuksen nopeus ja helppous |
|---|---|---|---|---|
| WCF | Hyvä | Hyvä | Hyvä | Huono |
| SignalR | Normaali | Huono | Normaali | Hyvä |
| Viestinvälitys | Normaali | Normaali | Hyvä | Normaali |
| ASP.NET Web API | Normaali | Normaali | Hyvä | Huono |
| WCF | Hyvä | Hyvä | Normaali | Huono |
| ASP.NET MVC | - | Hyvä | Hyvä | Huono |
| ASP.NET Web Forms | - | Huono | Normaali | Hyvä |
| WPF | - | Normaali | Normaali | Hyvä |
| WPF ja MVVM | - | Hyvä | Hyvä | Normaali |

Tiedonsiirtokerroksen teknologiaksi valittiin WCF. Suorituskyvyssä WCF peittoaa kaikki vanhemmat hajautettujen järjestelmien tekoon tarkoitetut .NET-teknologiat. SignalR on hyvä vaihtoehto kaksisuuntaisen liikenteen toteutukselle, muttei tue TCP-protokollaa. Sisäverkossa toteutettavalle järjestelmälle TCP-protokolla on osoittautunut nopeimmaksi ratkaisuksi, joka oli yksi syy WCF:n valintaan. Viestinvälitys on paluuarvoton suunnitteluratkaisu, jolloin esimerkiksi testeissä testatun kuvausaikataulun nouto ei onnistuisi yhdellä operaatiolla. Tästä syntyy turhaa kuormitusta verrattuna WCF:n kaksisuuntaiseen tiedonvälitykseen. WCF-sovelluskehys vaatii toteuttajalta perehtymistä, missä viestinvälitys ja SignalR ovat helpommin opeteltavia teknologioita.[44]

Palvelukerroksen vaihtoehdoista WCF-sovelluskehyksellä voidaan palvella työpöytä- ja web-sovelluksia, kun taas ASP.NET Web API on sovelluspalveluiden toteutukseen. Molemmat teknologiat tukevat hyvin erilaisia turvallisuusratkaisuja. Web API on hieman yleiskäyttöisempi ja kevyempi, sillä tavallisen HTTP-paketin lisäkuorma (overhead) on yleensä WCF:n SOAP-viestiä pienempi. ASP.NET Web API tukee vain HTTP-protokollaa, jonka takia se häviää TCP-protokollaa tukevalle WCF-sovelluskehykselle suorituskyvyssä. Molemmat soveltuvat palvelupohjaisten järjestelmien palvelukerroksen toteutukseen, WCF paremmin sisäverkon järjestelmiin ja Web API internetin sovelluspalveluiden tekoon.[45][46]

Käyttöliittymäkerroksen ratkaisuvaihtoehdoissa suorituskyvyssä ei ole työn kannalta merkitystä arvioinnissa, sillä tiedonkulku asiakassovelluksen ja tietokannan välillä on ratkaisevaa. PET ERP -järjestelmän kokoisen web-sovelluksen tekoon soveltuisi ASP.NET MVC paremmin kuin ASP.NET Web Forms. MVC on muunneltavuudeltaan ja yleiskäyttöisyydeltään parempi, sillä Web Forms on tiukasti sidottu sivuihin, kun taas

MVC:llä voidaan toteutusta eriyttää ja käyttää yhteisiä käsittelijöitä. Työpöytäsovelluksessa MVVM-suunnitteluratkaisun opetteluun kuluu aikaa, mutta sillä voidaan kätevästi eriyttää tiedon käsittely ja näkymien hallinta omaksi kerroksekseen, joka lisää toteutuksen yleiskäyttöisyyttä ja muunneltavuutta pelkän graafisen WPF-ohjelmistokomponentin käyttöön verrattuna.

## 5.4 Jatkokehitysideoita

Olio-relaatio-kuvaus tuo ongelmia asiakas-palvelin-ajatteluun, sillä relaatiollinen palvelin on erillään asiakassovelluksesta. Miten hallita olioiden tilaa ja identiteettiä asiakassovelluksessa ja palvelimella? Asiakas saa palvelulta vastauksena vain tietyn hetkisen tietokannan tiedon (snapshot) olion muodossa. Tämä eroaa relationaalisesta ajattelusta, jossa kaikki tieto on vain arvoja. Usean asiakassovelluksen järjestelmissä täytyy myös ottaa huomioon rinnakkaisuusongelmat: samaa oliota ei saisi päästä muokkaamaan yhtä aikaa, eikä asiakassovelluksilla saisi olla eri tiloissa olevia olioita. Näihin ongelmiin on tarjolla ratkaisuja välityspalvelimien, kopioiden, lukitusten ja takaisinkutsujen muodossa. Rinnakkaisuus on rajattu pois työn aihepiiristä, koska se ei liity suoraan tutkimusongelmaan, mutta tulee ottaa huomioon palvelupohjaisia järjestelmiä toteutettaessa. [30]

Palvelun funktioiden ja tietokantakyselyjen toteutusta voidaan mahdollisesti optimoida hakemalla vain käyttöliittymän kannalta pakollinen tieto, ja ladata loput tiedot taustalla päivittäen käyttöliittymää hetki hetkeltä. WCF tukee asynkronista tiedonsiirtoa ja .NET-sovelluskehys tukee taustakäsittelijöitä, joissa voidaan suorittaa operaatioita eri säikeessä estämättä käyttöliittymän toimintaa. Palvelukutsun suorittamisen jälkeen käyttöliittymä voidaan päivittää.[42]

Tietokantakerroksen toteutuksessa olio-relaatio-kuvausteknologiaksi on valittu Entity Framework:n versio 5.0. Entity Framework:n versio 6 tuo parannuksia kontekstien ja kyselyjen suorituskykyyn. Tähän siirtymistä kannattaa ainakin harkita, jos .NET-sovelluskehyksen version päivittäminen uudempaan järjestelmän ympäristössä onnistuu. Myös tietokannan indeksointi kannattaa ottaa huomioon tietokantakerrosta optimoitaessa. [42]

# 6 JOHTOPÄÄTÖKSET

Työn tutkimusongelmana olivat PET ERP -järjestelmälle asetettujen suorituskykyvaatimusten, reaaliaikaisen muutoksista tiedottamisen ja usean yhtäaikaisen käyttäjän haasteet. Tässä työssä selvitettiin palvelupohjaisen järjestelmän ratkaisuvaihtoehtoja suorituskykyisen tiedonkulun toteuttamiselle asiakassovellukselta tietokantaan ja takaisin. Järjestelmä voidaan jakaa kokonaisuuksiin kerrosten pohjalta. Käyttöliittymäkerroksen kattava asiakassovellus on hajautettu palvelu- ja tietokantakerroksesta tiedonsiirtokerroksen avulla ympäri Turun PET-keskuksen sisäverkkoa.

Työssä ratkaisuksi tarjottiin teknologia- ja suunnitteluratkaisukokonaisuutta, jonka muodostavat .NET-sovelluskehyksen tarjoamat komponentit. Käyttöliittymäkerros toteutettiin graafisella rajapintakerroksella WPF ja MVVM-suunnitteluratkaisun avulla. Palvelu- ja tiedonsiirtokerros toteutettiin WCF-ohjelmistokehyksen avulla ja tietokantakerros Entity Framework -sovelluskehyksellä. Tiedonsiirto perustui päätepisteisiin ja kaksisuuntaiseen tiedonkulkuun suorituskykyistä TCP-protokollaa käyttäen. Olio-relaatio-kuvauksella luotiin tietokannasta eriytetty tietomalli. Tietokantakyselyt kohdistettiin entiteettimalliin ja tiedonhakuolioina ympäri järjestelmää toimivat POCO-tyyppiset entiteetit.

Suoritettujen testien perusteella löydettiin suorituskykyinen ratkaisu PET ERP –järjestelmän päiväkohtaisen kuvantamisaikataulun noutavalle operaatiolle. Binäärisarjallistuksella ja LINQ to Entities -kyselykielellä optimoidun operaation keskimääräiseksi kestoksi saatiin 0,181 sekuntia. Tietomäärältään suuren testitietokannan ja yhtä aikaa palvelua kutsuvien testisovellusten huomioon ottaen tulosta voidaan pitää varsin onnistuneena. Testien simuloima tilanne oikean järjestelmän toiminnasta oli realistinen ja tavallista käyttöä vastaavaa kuvantamisaikataulua hieman raskaampi.

Työssä tarjottu ratkaisukokonaisuus sopii testien ja arvioinnin pohjalta hyvin PET ERP -järjestelmän toteutukseen. Ratkaisua voidaan käyttää myös yleisesti palvelupohjaisten tietojärjestelmien toteutuksen perustana. Käytetyt sovelluskehykset ja komponentit ovat muunneltavia ja yleiskäyttöisiä. WCF-sovelluskehys tarjoaa monipuolisia vaihtoehtoja vaatimuksissa mainittujen muiden järjestelmien ja laitteiden integraatioille PET ERP -järjestelmään. Myös työpöytäsovelluksesta web-sovellukseen siirtyminen on tehty vaivattomaksi eriyttämällä käyttöliittymäkerros ja palvelukerros toisistaan WCF:llä toteutetun tiedonsiirtokerroksen ansiosta. Uuden käyttöliittymäkerroksen tarvitsee vain to-

teuttaa palvelun rajapinta. Näkymien kehitystyössä voidaan käyttää Entity Frameworkin tarjoamaa entiteettimallia tiedon esityksen suunnittelulle.

Työssä onnistuttiin kattamaan kaikki keskeiset PER ERP -järjestelmän kaltaisen palvelupohjaisen tietojärjestelmän suunnitteluratkaisut .NET-sovelluskehyksellä. Tiedon kuvauksen ja sarjallistamisen tarkastelu toteutettiin riittävän yksityiskohtaisesti ja vaihtoehtoja arvioitiin suhteellisen laajasti. Suorituskyvyn optimoinnille jätettiin työssä vielä paljon varaa, eikä kaikkia sarjallistamisen tai tietokantahakujen optimointiin tarkoitettuja suunnitteluratkaisuja tai tekniikoita käyty läpi. Myös itse tietokannan optimointi jätettiin jatkokehitykseen.

Yleisesti .NET-ohjelmistokehys ja tarjottu ratkaisukokonaisuus sopii palvelukeskeisten järjestelmien suunnitteluun ja toteutukseen. Ratkaisu skaalautuu hyvin pieniin ja keskisuuriin järjestelmiin, mutta todella isojen järjestelmien toteutuksessa suorituskyky saattaa aiheuttaa haasteita. Tulevaisuudessa .NET-sovelluskehys tarjoaa enemmän mahdollisuuksia ja tukea useammalle alustalle sen siirryttyä avoimen lähdekoodin teknologiaksi.[47]

# LÄHTEET

[1] Davenport, T.H. Putting the Enterprise into the Enterprise System. Harvard Business Review, 1998, 76(3). Sivut.121–131.)

[2] Turun PET-keskus. Tietojärjestelmän hankinta PET-keskuksen toiminnanohjaukseen, ERP-vaatimusmäärittely. Ei julkisesti saatavilla.

[3] Stefan Dumbrava, Doru Panescu and Mihaela Costin. A Three-tier Software Architecture for Manufacturing Activity Control in ERP Concept [WWW]. International Conference on Computer Systems and Technologies - CompSysTech' 2005. 2005, pp 3-1 – 3-6. [viitattu 9.5.2014] saatavissa:
http://www.researchgate.net/publication/228947994_A_Threetier_Software_Architecture_for_Manufacturing_Activity_Control_in_ERP_Concept

[4] Medbit Oy. Arkkitehtuuriperiaatteet versio 0.1. Ei julksesti saatavilla.

[5] Varsinais-Suomen Sairaanhoitopiiri. PET-keskus [WWW]. [Viitattu 30.10.2014] saatavissa:
http://www.vsshp.fi/fi/toimipaikat/tyks/osastot-ja-poliklinikat/Sivut/pet-keskus.aspx

[6] TurkuCRC. Hyvän kliinisen tutkimustavan (GCP) periaatteet [WWW]. [viitattu 15.5.2014] saatavissa: http://www.turkucrc.fi/index.phtml?s=43

[7] Microsoft. Sharepoint [WWW]. [viitattu 14.12.2014] saatavissa:
http://office.microsoft.com/fi-fi/sharepoint/

[8] Tanenbaum, A. Steen, M. 2002. Distributed Systems, Principles and Paradigms. Prentice-Hall, ISBN: 0-13-088893-1.

[9] Solis, S. 2009. Illustrated WPF. APress, ISBN: 978-1-4302-1911-8

[10] Mircrosoft Developer Network. The MVVM Pattern [WWW]. [viitattu 2.10.2014] saatavissa:
http://msdn.microsoft.com/en-us/library/hh848246.aspx

[11] Hoang, L., Thuan L. T. 2002. .NET Framework Essentials, O'Reilly, ISBN: 0-596-00302-1.

[12] MacDonald, M., Freeman, A., Szpuszta, M. 2010. Pro ASP.NET 4 in C# 2010, Apress, ISBN: 978-4302-2529-4.

[13] Ghani, Imram Abdul. Difference betweeen ASP.NET WebForms and ASP.NET MVC [WWW]. [viitattu 16.9.2014] saatavissa:

[14] Hall, G.M. 2010. Pro WPF and Silverlight MVVM: Effective Application Development with Model-View-ViewModel. Apress, ISBN: 978-1-4302-3162-2

[15] Sharp, J. 2010. Windows Communication Foundation 4 Step by Step. Microsoft Press, ISBN: 978-0-735-64556-1

[16] Gordon, I. 2011. Essential Software Architecture. Springer, ISBN: 978-3-642-19175-6

[17] Block, G., Cibraro, P., Felix, P., Dierking, H., Miller, D. 2014. Designing Evolvable Web APIs with ASP.NET. O'Reilly, ISBN: 978-1-449-33771-1

[18] Harrington, J.L. 2009. Relational Database Design and Implementation. Morgan Kaufmann Publishers, ISBN: 978-0-12-374730-3

[19] Harrington, J.L. 2002. Relational Database Design Clearly Explained. Morgan Kaufmann Publishers, ISBN: 978-1-55860-820-7

[20] Rotem-Gal-Oz, A. 2012. SOA Patterns. Manning, ISBN: 9781933988269

[21] Koskimies, K., Mikkonen, T. 2005. Ohjelmistoarkkitehtuurit. Helsinki, Talentum, ISBN: 9789521408625

[22] Resnick, S., Crane, R., Bowen, C. 2008. Essential Windows Communication Foundation (WCF). Addison-Wesley Professional, ISBN: 0-321-44006-4

[23] Cibraro, P., Clayes, K., Cozzolino, F., Grabner, J. 2010. Professional WCF 4: Windows Communication Foundation with .NET 4. Wiley Publishing Inc, ISBN: 978-0-470-56314-4

[24] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. 1996. Pattern-Oriented Software Architecture, Volume 1, A System of Patterns. Wiley.
http://www.webdevelopmenthelp.net/2013/10/difference-between-asp-net-webform-and-asp-net-mvc.html

[25] Mircrosoft Developer Network. Message Queuing (MSMQ) [WWW]. [viitattu 3.10.2014] saatavissa:
http://msdn.microsoft.com/en-us/library/ms711472(v=vs.85).aspx

[26] Aguilar, J.M. 2014. SignalR Programming in Microsoft ASP.NET. Microsoft Press, ISBN: 978-0-7356-8388-4

[27] Pathak, N. 2011. Pro WCF 4: Practical Microsoft SOA Implementation. Apress, ISBN: 978-1430233688

[28] Avani, D., Sudhanshu, H. 2010. Microsoft Data Access Technologies SWOT Analysis. White paper, Infosys.

[29] Mircrosoft Developer Network. Entity Framework Overview [WWW]. [viitattu 17.10.2014] saatavissa:
http://msdn.microsoft.com/en-us/library/vstudio/bb399567.aspx

[30] Fussell, M. L. 1997. Foundations of Object Relational Mapping. White paper, ChiMu Corp.

[31] Jennings, R. 2009. Professional ADO.NET 3.5 with LINQ and the Entity Framework. Wiley Publishing, ISBN: 978-0-470-18261-1

[32] Adya, A., Blakeley, J. A., Melnik, S., Muralidhar, S. Anatomy of the ADO.NET Entity Framework [WWW]. Proceedings of the ACM SIGMOD International Conference on Management of Data. 2007, pp 877-888.

[33] Patel, A. WCF Serialization Part 1: Day 6 [WWW]. [Viitattu 30.10.2014] saatavissa: http://www.c-sharpcorner.com/UploadFile/db2972/wcf-serialization-part-1-day-6/

[34] Mehta, V., P. 2008. Pro LINQ Object Relational Mapping with C# 2008. Apress, ISBN: 978-1-59059-965-5.

[35] Tenny, L., Hirani, Z. 2010. Entity Framework 4.0 Recipes A Problem-Solution Approach. Apress, ISBN: 978-1-4302-2703-8

[36] Fowler, M. Data Transfer Object [WWW]. [viitattu 16.11.2014] saatavissa: http://martinfowler.com/eaaCatalog/dataTransferObject.html

[37] Lerman, J. 2010. Programming Entity Framework. O'Reilly, ISBN: 978-0-596-80726-9.

[38] Mircrosoft Developer Network. Chapter 4: WCF Security Fundamentals [WWW]. [viitattu 26.11.2014] saatavissa: http://msdn.microsoft.com/en-us/library/ff650862.aspx

[39] Mircrosoft Developer Network. WCF-NetTcp Transport Properties Dialog Box, Receive, Security Tab [WWW]. [viitattu 26.11.2014] saatavissa: http://msdn.microsoft.com/en-us/library/bb246097.aspx

[40] Mircrosoft Developer Network. Data Contract Known Types [WWW]. [viitattu 3.12.2014] saatavissa:
http://msdn.microsoft.com/en-us/library/ms730167%28v=vs.110%29.aspx

[41] Lerman, J., Miller, R. 2012. Programming Entity Framerwork: DbContext. O'Reilly, ISBN: 978-1-449-31296-1

[42] Microsoft Developer Network. Performance Considerations for Entity Framework 4, 5, and 6 [WWW]. [viitattu 12.12.2014] saatavissa:
http://msdn.microsoft.com/en-us/data/hh949853.aspx#5

[43] Terrance, A., S. High Performance WCF Services: netTcpBinding [WWW]. [viitattu 14.12.2014] saatavissa: http://blog.shutupandcode.net/?p=1085

[44] Microsoft Developer Network. A Performance Comparison of Windows Communication Foundation (WCF) with Existing Distributed Communication Technologies [WWW]. [viitattu 14.12.2014] saatavissa:
http://msdn.microsoft.com/en-us/library/bb310550.aspx

[45] Microsoft Developer Network. WCF and ASP.NET Web API [WWW]. [viitattu 15.12.2014] saatavissa:
http://msdn.microsoft.com/en-us/library/jj823172(v=vs.110).aspx

[46] Nein, J. ASP.NET Web API vs. Windows Communication Foundation (WCF) [WWW]. [viitattu 15.12.2014] saatavussa: http://captechconsulting.com/blog/john-nein/aspnet-web-api-vs-windows-communication-foundation-wcf

[47] Microsoft Developer Network. .NET Core is Open Source [WWW].[viitattu 16.12.2014] saatavissa:
http://blogs.msdn.com/b/dotnet/archive/2014/11/12/net-core-is-open-source.aspx