Petri Honkavaara

Päätöstuloste HR-järjestelmään

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sulautettu tietotekniikka

Insinöörityö

23.5.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Petri Honkavaara Päätöstuloste HR-järjestelmään 31 sivua 23.5.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	sulautettu tietotekniikka
Ohjaajat	kehityspäällikkö Arto Ihantoja tuntiopettaja Pasi Ranne

Insinöörityössä suunniteltiin vuosilomahakemukselle päätöstulosteen toteutus ja toteutettiin se HR-järjestelmään. Päätöstuloste sisältää hakemuksesta tehdyn päätöksen oleellisimmat tiedot ja käyttäjän tulee saada se selaimesta auki PDF-muodossa. Toteutus tuli suunnitella niin, että jatkossa olisi helppo lisätä uusia päätöstulosteita eri hakemuksille. Insinöörityö tehtiin osana HR-järjestelmän kehitysprojektia CGI Suomi Oy:lle.

Ympäristö ja käytettävät tekniikat olivat ennalta määrätyt, joten niiden valintoihin ei insinöörityössä otettu kantaa. Päätöstulosteen toteuttamista suunnitellessa piti ottaa huomioon olemassa oleva ympäristö ja sijoittaa päätöstulosteen tarvitsemat osat osaksi sitä. Päätöstulosteen muodostamista varten oli valittu SQL Server Reporting Services -työkalu, jolla päätöstulosteen ulkoasu ja tarvittavan tietojoukon hakeminen toteutettiin.

Insinöörityössä vertailtiin ja testattiin eri vaihtoehtoja päätöstulosteen toteuttamiseksi, minkä perusteella toteuttamistapa valittiin. Päätöstulosteen datalähteen valinta oli yksi vertailussa oleva kohde. Koska päätöstulosteen tarvitsema tietojoukko vaihtelee hakemuskohtaisesti, täytyy sen kokoamisen olla helposti toteutettavissa usealle eri hakemukselle. Lisäksi päätöstulosteiden ulkoasun muokattavuuden tuli olla yksinkertaista ja tavoitteena oli saada sellainen rakenne, että yhteen paikkaan tehty muutos koskisi kaikkia päätöstulosteita.

Vuosilomahakemuksen päätöstuloste saatiin toteutettua ja liitettyä osaksi HR-järjestelmää. Se vastasi työtä varten saatuja määritelmiä, jonka lisäksi oltiin tyytyväisiä tapaan, jolla se oli liitetty osaksi olemassa olevaa HR-järjestelmää. Myöhemmässä vaiheessa asiakkaalta tuli pyyntö toteuttaa uusia päätöstulosteita eri hakemuksille, mikä onnistui toimivan kokonaisuuden ansiosta sujuvasti. Kaikki päätöstulosteet menivät asiakkaalle testiin, jonka seurauksena päätöstulosteille haluttiin asioita eri tavalla näkyviin ja määrittelyihin tuli muutoksia. Tämän seurauksena päätöstulosteita ruvettiin muokkaamaan asiakkaan tarpeita vastaaviksi ja ne vietiin uudelleen testattaviksi. Muutoksia tullaan tekemään niin pitkään, kunnes asiakas on varma haluamastaan lopputuloksesta ja päätöstulosteet vastaavat sitä.

Avainsanat	SSRS, .NET, ohjelmistokehitys

Author Title Number of Pages Date	Petri Honkavaara Decision printout for an HR system 31 pages 23 May 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Embedded Engineering
Instructors	Arto Ihantoja, Development Manager Pasi Ranne, Lecturer

The thesis describes the implementation of a decision printout for the annual leave application of an HR system. The decision printout contains the most important information of the decision made on the application and it has to be available for the user on a web browser in PDF format. The implementation was to be done so that in future it would be easy to add new decision printouts for different types of applications. The thesis was made for CGI Suomi Oy as a part of the development project of the HR system.

The environment and the used technology were pre-defined, so the thesis does not describe the reasons for them. While designing the implementation of the decision printout, it was necessary to be familiar with the existing environment and embed all the needed components for the decision printout to the existing environment. The SQL Server Reporting Services utility was chosen in order to deploy the layout and content of the decision printout.

The decision printout for the annual leave application was implemented successfully and got deployed as a part of an already existing environment. The decision printout fulfilled the requirements and the way it was deployed as a part of HR system was satisfying. Afterwards the customer ordered new decision printouts for different kinds of applications, which were also implemented.

Keywords	SSRS, .NET, software development

Sisällys

Lyhenteet ja termit

.NET	.NET Framework, ohjelmistokehitykseen tarkoitettu alusta, joka tarjoaa kehittäjälle mm. laajan luokkakirjaston kehitystyön avuksi.
ASP.NET	Osa .NET Frameworkin teknologioita WEB-sovellusten kehitystyön avuksi.
ASPX	Tiedostomuoto, sisältää sovelluksesta selaimeen lähetettävän HTML-muotoisen sanoman.
ESB	*Enterprise Service Bus*. Ohjelmistoarkkitehtuurin malli palvelukeskeisten ohjelmien kehittämiseen, missä itsenäiset sovellukset kommunikoivat keskenään.
HR-järjestelmä	*Human Resources* -järjestelmä. Hoitaa henkilöstön hallintaan liittyviä asioita.
Päätöstuloste	Tuloste tehdyn päätöksen tiedoista. Tämän työn yhteydessä päätöstulosteella tarkoitetaan tulostetta jonkin HR-järjestelmään tehdyn hakemuksen päätöksen oleellisimmista tiedoista.
RDL	*Report Definition Language*. Tiedosto XML-muodossa esitettynä, joka sisältää raportin tiedot.
Silverlight	Kehitystyökalu käyttöliittymien luomiseen WEB- ja mobiilisovelluksille.
SOAP	*Simple Object Access Protocol*. Tietoliikkenneprotokolla, joka mahdollistaa proseduurien etäkutsun.
SSRS	*SQL Server Reporting Services.* SQL Server -ohjelmiston osa. Serveripohjainen raportointialusta, jolla voidaan luoda, hallita ja toimittaa raportteja. Sisältää myös ohjelmointirajapinnan, jonka ansiosta kehittäjät voivat kutsua SSRS:ää omissa sovelluksissaan.
WCF	*Windows Communication Foundation*. Framework palvelukeskeisten ohjelmien kehittämiseen.

WSDL	*Web Service Desription Language*. XML-pohjainen kieli, jota käytetään palvelun toiminnollisuuden kuvaamiseen.

1 Johdanto

Insinöörityö tehdään CGI Suomi Oy:lle osana HR-järjestelmän kehitysprojektia. HR (Human Resources) -järjestelmällä hoidetaan henkilöstön hallintaan liittyviä asioita. Työn tavoitteena on suunnitella päätöstulosteen toteutus ja toteuttaa se HR-järjestelmään. Päätöstuloste on tuloste jonkin HR-järjestelmään tehdyn hakemuksen päätöksen oleellisimmista tiedoista. HR-järjestelmässä on useita kymmeniä eri hakemuksia, joita käyttäjä voi täyttää ja lähettää eteenpäin. Kun hakemus on lähetetty eteenpäin, asianomainen käsittelijä avaa sen ja tekee päätöksen. Hakemuksen tehneen esimiehen tai käsittelijän pitää pystyä antamaan hakijalle tuloste tehdystä päätöksestä hakijan niin halutessa.

Tässä vaiheessa projektia toteutetaan päätöstuloste vain yhdelle hakemukselle, vaikka eri hakemuksia on useita kymmeniä. Toteutusta suunnitellessa pitää varautua siihen, että jatkossa halutaan lisää päätöstulosteita eri hakemuksille. Uusien päätöstulosteiden lisääminen projektiin pitää sujua tulevaisuudessa mahdollisimman helposti.

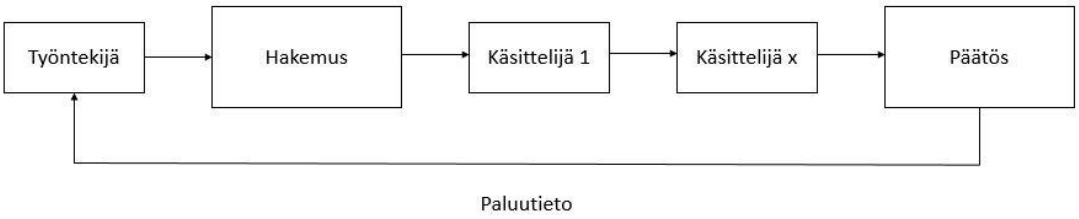
2 Alkuasetelma

2.1 Toimintaympäristön esittely

Olemassa oleva HR-järjestelmä on käytössä monilla asiakkailla ympäri Suomea, mutta järjestelmä ja sen käyttämät teknologiat kaipaavat uudistamista. Järjestelmässä on voitu käyttää sähköisiä lomakkeita hyväksi, mutta ajan kanssa on tullut esille uusia kehitystarpeita. Käynnissä olevassa projektissa uudistetaan ja yksinkertaistetaan teknisiä ratkaisuja järjestelmään sekä ennen kaikkea mahdollistetaan uudenlaisten sähköisten palvelujen tuottaminen asiakkaan erilaisiin tarpeisiin.

HR-järjestelmän eteen on kehitetty uusi sähköinen edusta, josta käytetään nimeä lomakemoottori. Sen avulla luodaan uudenlaisia palveluita, tehostetaan sähköistä asiointia ja korostetaan itsepalvelua. Asioita automatisoidaan siis niin pitkälle kuin mahdollista. Ideana on, että käyttäjä voi hoitaa omia työsuhdeasioita mahdollisimman paljon itse, esimerkiksi lomien ja poissaolojen ilmoittamiset tulevat onnistumaan helposti.

Lomakemoottori mahdollistaa itsepalvelun. Kun työntekijä haluaa hakea esimerkiksi vuosilomaa, tekee hän itse tai joku muu hänen puolestansa hakemuksen lomakemoottoriin. Hakemus lähtee käsiteltäväksi, ja sen voi käsitellä esimies tai esimiesketju riippuen hakemuksen luonteesta. Jos kyseessä on esimiesketju, alin esimies voi esimerkiksi puoltaa hakemusta ja laittaa sen eteenpäin. Tämä jatkuu niin pitkään, kunnes joku ketjusta tekee lopullisen päätöksen eli hylkää tai hyväksyy hakemuksen. Hakemusten käsittely päättyy siis aina päätökseen. Kuva 1 havainnollistaa päätöksentekoprosessin kulun.

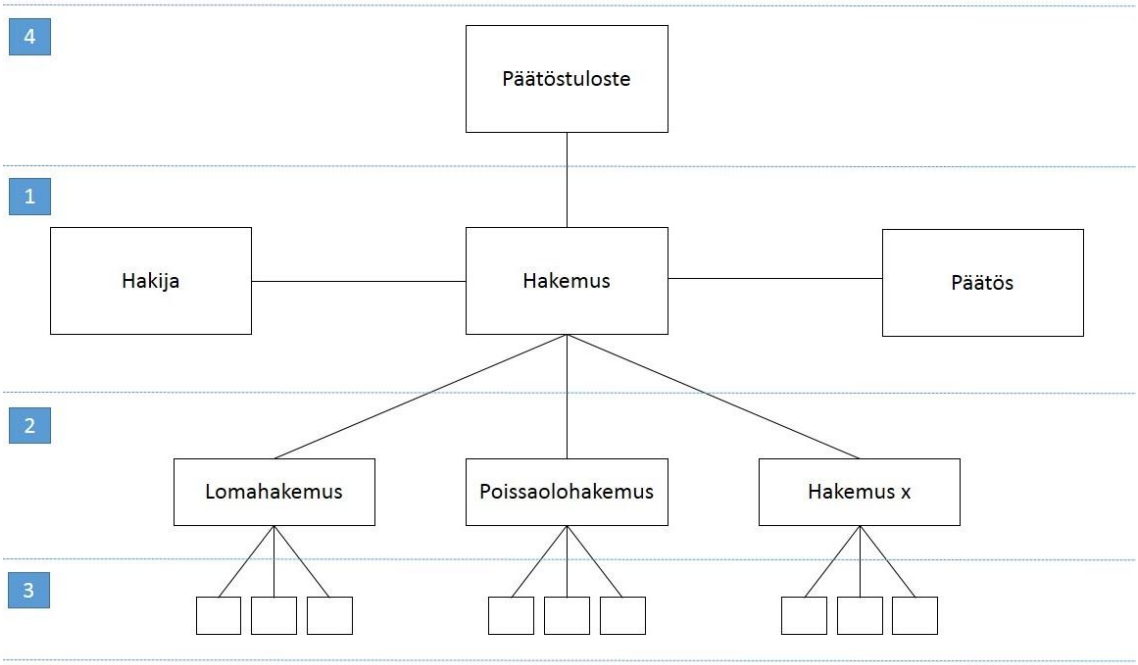


Kuva 1. Päätöksentekoprosessin kulku.

Lomakemoottorin yksi sisällöllinen osa-alue liittyy henkilöiden vuosilomiin. Myös näiden hakemusten käsittelyn lopussa syntyy päätös. Vuosilomahakemukselle tehdystä päätöksestä taas pitäisi saada muodostettua päätöstuloste, mikä on tämän insinöörityön aihe.

2.2 Hakemus ja päätöstuloste

Kuten todettu, järjestelmässä on useita kymmeniä yksittäisiä hakemuksia. Hakemukset ovat dynaamisia lomakkeita, joita hakija voi täyttää. Jokaisella hakemuksella on yksilöllinen lomake, tietosisältö, käsittelytapa ja käsittelysäännöt. Esimerkkinä käsittelysäännöstä on hakemuksen päätöksentekoon tarvittava hyväksyntäketju. Tämä tarkoittaa määrää, kuinka monen esimiehen kautta hakemuksen on mentävä, jotta vastaan tulee tarpeeksi korkealla tasolla oleva esimies, jolla on valtuudet tehdä kyseisestä hakemuksesta päätös. Jokaisesta konkreettisesta hakemuksesta, josta on tehty päätös, voidaan tehdä yksilöllinen päätöstuloste. Käydään kuvan 2 avulla läpi päätöstulosteen ja hakemuksen sijoittumista kokonaisuuteen, keskittyen erityisesti vuosilomahakemukseen ja vuosilomapäätöstulosteeseen.



Kuva 2. Käsitemalli.

Kuvassa 2 on avattu rakennetta, miten eri käsitteet liittyvät toisiinsa ja miten ne ovat toisistaan riippuvaisia. Käydään kuvaa läpi taso kerrallaan:

1. Hakija täyttää jonkin hakemuksen. Hakemuksesta riippuen käydään läpi tarpeellinen hyväksyntäkierros, jonka jälkeen tehdään hakemukselle päätös. Lopullinen päätös hakemukselle on aina muotoa hylätty tai hyväksytty.

2. Hakemukset on lajiteltu hakemustyypeittäin. Tällä hetkellä järjestelmässä on lomahakemuksia ja poissaolohakemuksia. ”Hakemus x” kuvaa sitä, että järjestelmään voi tulla lisäksi uusia hakemustyyppejä.

3. Hakemustyyppien alle tulevat yksittäiset hakemukset, joita on yhteensä noin neljäkymmentä kappaletta. Lomahakemuksia on noin kymmenen kappaletta ja poissaolohakemuksia on noin kolmekymmentä kappaletta. Esimerkkinä yksittäisestä hakemuksesta on vuosilomahakemus, jonka hakemustyyppi on lomahakemus. Olemassa olevien hakemuksien lisäksi voidaan osaksi järjestelmää tuoda kokonaan uusia hakemuksia sijoittamalla ne oikean hakemustyypin alle tai luomalla kokonaan uusi hakemustyyppi.

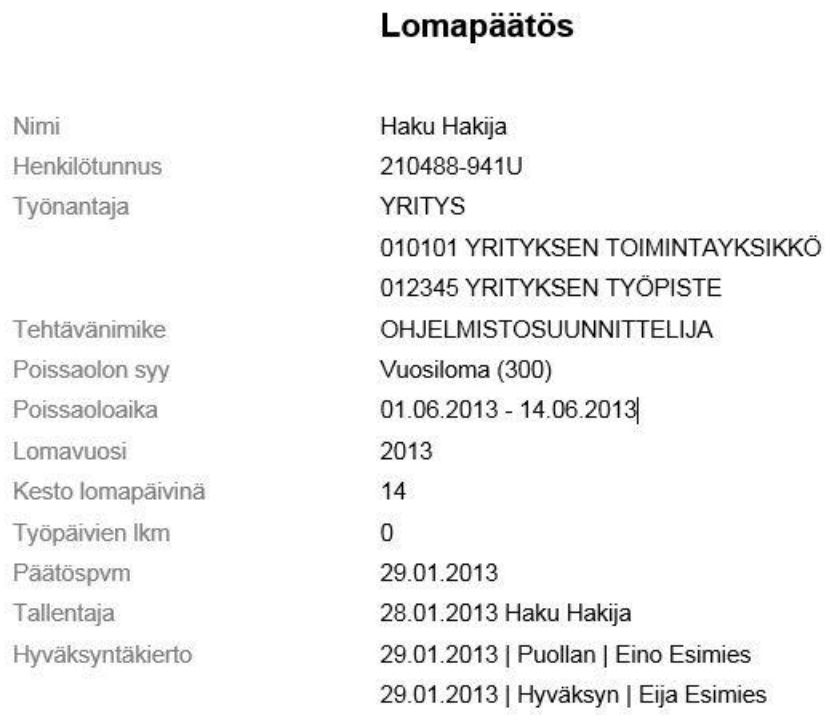
4. Kolmen edellisen kohdan perusteella pystytään käsittämään päätökseen tarvittavan prosessin kulku ja hakemusten yksilöllisyys. Päätöstuloste voidaan tehdä mille tahansa hakemukselle, kunhan hakemuksesta on tehty päätös. Hakemuksesta tarvittava tietojoukko päätöstulosteelle on siis saataville siitä hetkestä lähtien kun päätös on tehty. Jokaisella yksittäisellä hakemuksella tarvittava tietojoukko on erilainen ja se pitää ottaa huomioon päätöstulostetta luodessa.

2.3 Päätöstulosteen tarve

Päätöstuloste voi olla virallinen asiakirja, jolla hakija voi esimerkiksi todistaa vuosilomansa ajankohdan. Järjestelmässä tehtyjen päätöksien suuren määrän vuoksi päätöksenteon yhteydessä ei kuitenkaan automaattisesti tehdä päätöstulostetta ja tallenneta sitä arkistoon. Sen sijaan on määritelty, että käyttäjä voi halutessaan käydä erikseen tilaamassa päätöstulosteen käyttöliittymältä. Käyttöliittymältä tilattava tuloste on PDF-formaatissa. Käyttäjä voi avata päätöstulosteen suoraan PDF-lukuohjelmaan, tai tallentaa sen.

Järjestelmässä on tällä hetkellä noin 40 yksittäistä hakemusta, joille voidaan tehdä päätös. Tässä vaiheessa projektia toteutetaan päätöstuloste kuitenkin vain yhdelle hakemukselle, joka on vuosilomalomahakemus. Tulevaisuudessa tullaan todennäköisesti toteuttamaan päätöstulosteita useammille eri hakemuksille, mutta tässä työssä keskitytään yhden päätöstulosteen toteutukseen. Toteutusta suunnitellessa otetaan kuitenkin huomioon mahdollisuus lisätä myöhemmin uusia päätöstulosteita ratkaisuun mahdollisimman helposti.

Vuosilomahakemus on yksittäinen hakemus, joka on hakemustyypiltään lomahakemus. Erilaisia lomahakemuksia on yhteensä noin kymmenen kappaletta ja kaikilla niiden päätöstulosteilla on joukko yhteisiä kenttiä, jonka lisäksi hakemuksesta riippuen myös joukko yksilöllisiä kenttiä. Kuvasta 2 näkee miten yksittäiset hakemukset sijoittuvat osaksi kokonaisuutta. Lomatyypistä riippuen päätöstulosteen tietosisältö ja sen määrä siis vaihtelee.



Lomapäätös

Nimi	Haku Hakija
Henkilötunnus	210488-941U
Työnantaja	YRITYS
	010101 YRITYKSEN TOIMINTAYKSIKKÖ
	012345 YRITYKSEN TYÖPISTE
Tehtävänimike	OHJELMISTOSUUNNITTELIJA
Poissaolon syy	Vuosiloma (300)
Poissaoloaika	01.06.2013 - 14.06.2013
Lomavuosi	2013
Kesto lomapäivinä	14
Työpäivien lkm	0
Päätöspvm	29.01.2013
Tallentaja	28.01.2013 Haku Hakija
Hyväksyntäkierto	29.01.2013 | Puollan | Eino Esimies
	29.01.2013 | Hyväksyn | Eija Esimies

Kuva 3. Päätöstuloste-esimerkkinä lomapäätös, jossa lomatyyppinä vuosiloma.

Päätöstulosteen määrittelyjen mukana saatiin asiakkaalta esimerkkinä vuosilomahakemuksen päätöstulosteen ulkoasu, joka on kuvassa 3. Päätöstulosteen ulkoasu on erittäin yksinkertainen, eikä se sisällä mitään grafiikkaa. Päätöstulosteella on ylhäällä otsikko, vasemmassa sarakkeessa kenttäkohtaiset selitteet ja oikeassa sarakkeessa itse tieto. Lomatyyppi näkyy tulosteella kohdassa poissaolon syy. Esimerkissä on lomatyyppinä vuosiloma ja sen lomatyyppikohtaiset kentät ovat ”Lomavuosi”, ”Kesto lomapäivinä” ja ”Työpäivien lkm”. Lomatyypistä riippumatta kaikki muut kentät tulisivat lomahakemuksien päätöstulosteilla olemaan yhteisiä. Jokaisen eri lomatyypin hakemuskohtaiset kentät tulisivat samaan väliin kuin vuosilomalla ja niitä olisi kahdesta kolmeen. Samantapaisella logiikalla sijoittuisivat myös kaikkien poissaolohakemuksien kentät päätöstulosteille.

2.4 Ratkaisun ohjelmistoarkkitehtuurin elementit

Ratkaisun ohjelmistoarkkitehtuuri on jo valmiiksi valittu ja käytössä. Valinnat eivät siis kaipaa perusteluja eikä niihin voi enää vaikuttaa. Ohjelmistoarkkitehtuuri koostuu seuraavista elementeistä:

- .NET 4.0, ASP.NET 4.0 ja Silverlight 5
- Microsoftin käyttöjärjestelmät Windows 7 ja Windows Server 2008 R2
- Silverlight 5 Tools for Visual Studio 2010 (kehitysväline)
- SQL Server 2008 R2
- Visual Studio 2010 ja Visual Studio 2008 (kehitysväline)
- Visual SVN (kehitysväline).

SQL Server -ohjelmiston mukana on asennettu myös SSRS (SQL Server Reporting Services). SSRS on palvelinpohjainen raportointialusta, jolla voidaan luoda, hallita ja toimittaa raportteja. Se sisältää myös ohjelmointirajapinnan, jonka ansiosta kehittäjät voivat kutsua SSRS:ää omissa sovelluksissaan. SSRS:llä tehtyyn raporttiin liittyy aina datalähde sekä erilaiset kontrollit, joiden avulla dataa haetaan, näytetään ja ryhmitellään. [1.] Tämä kaikki tieto löytyy RDL-loppuisesta tiedostosta (Report Definition Language), johon se on XML-muodossa tallennettu. Itse RDL-tiedostoa ei kuitenkaan tarvitse rakentaa XML:nä vaan sitä voidaan kehittää esimerkiksi Visual Studiossa, joka luo taustalle XML-kuvauksen. [2.] RDL-tiedostojen kehitys tulee tehdä käyttäen Visual Studion versiota 2008, jotta se olisi yhteensopiva SQL Serverin version kanssa.

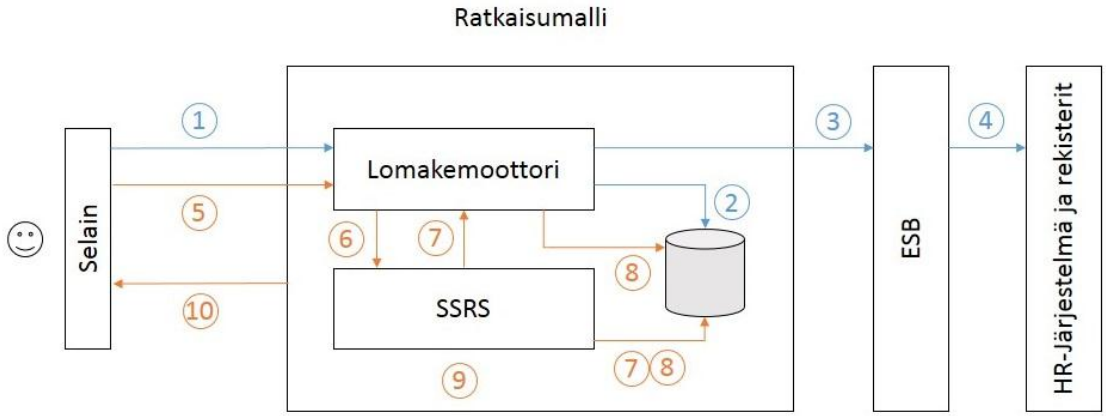
Sovelluksen kehitysympäristönä on Visual Studio 2010, jossa itse ohjelmointi toteutetaan. Visual Studioon on asennettu Silverlight 5 Tools for Visual Studio 2010 -lisäosa, jotta käyttöliittymää kehittäessä saadaan käyttöön Silverlightin tarjoamat edut. Sovelluksen taustalla on .NET 4.0 Framework, joka antaa kehittäjän käyttöön valmiita luokkakirjastoja. Projektin kehityksessä on käytössä Visual SVN -versionhallinta, joka mahdollistaa koodimuutosten sitomisen tehtäviin ja pitää kehitystyön järjestelmällisenä. Kehittäjien ja sovelluksen käyttäjien käyttöjärjestelminä on Windows 7. Lomakemoottori, SSRS ja tietokanta ovat palvelimilla, joiden käyttöjärjestelminä on Windows Server 2008 R2.

3 Suunnittelu

3.1 Päätöksenteko ja päätöstulosteen tilaus

Raamit työtä varten on annettu valmiina, eikä niihin voi vaikuttaa. Tiedon kulun tulee tapahtua HR-järjestelmään ja sen rekistereihin ESB-väylän kautta. ESB (Enterprise Service Bus) on ohjelmistoarkkitehtuurin malli palvelukeskeisten ohjelmien kehittämiseen, jossa itsenäiset sovellukset kommunikoivat keskenään. Tässä tapauksessa lomakemoottori ja HR-järjestelmä kommunikoivat keskenään ESB:n avulla. Olemassa olevaan raamiin kuuluu HR-järjestelmä ja sen rekisterit sekä lomakemoottori ja sen käyttämä tietokanta. SSRS ja muut tarpeelliset osat tulee sovittaa osaksi olemassa olevaa raamia.

Kuvassa 4 havainnollistetaan päätöksentekoprosessia ja tarkastellaan päätöstulosteen sovittamista järjestelmään ohjelmistoratkaisun näkökulmasta. Kuvassa oleva SSRS-niminen laatikko tulee uutena osana olemassa olevaan raamiin. Sinisellä merkityt tapahtumat kuvaavat päätöksentekoprosessia ja oranssilla merkityt tapahtumat päätöstulosteen muodostamisprosessia.



Kuva 4. Päätöstuloste ohjelmistoratkaisun näkökulmasta.

Käyttötapausesimerkissä käyttäjä on tehnyt vuosilomahakemuksen. Päättäjä on käynyt hyväksymässä vuosilomahakemuksen. Käyttäjä haluaa tulosteen päätöksestä ja käy tilaamassa käyttöliittymältä päätöstulosteen.

Käydään tilanne läpi ohjelmistoratkaisun näkökulmasta, kuinka prosessi toimii, kun hakemus on syötetty järjestelmään:

Päätöksenteko

1. Päättäjä käsittelee hakemuksen ja tekee päätöksen.
2. Lomakemoottori tallentaa päätöksen omaan tietokantaansa.
3. Lomakemoottori kutsuu hakemukselle osoitettua ESB-palvelua, esimerkiksi tallenna-palvelua.
4. ESB kutsuu HR-järjestelmän palveluita päätöksen viemiseksi HR-järjestelmään.

Päätöstulosteen tilaus

5. Tarvittaessa käyttäjä pyytää päätöstulosteen lomakemoottorilta.
6. Lomakemoottori delegoi pyynnön SSRS:lle.
7. SSRS kutsuu lomakemoottoria saadakseen tarvitsemansa tietojoukon päätöstulosteen muodostamista varten. Vaihtoehtoisesti SSRS voi itse hakea tarvitsemansa tietojoukon suoraan tietokannasta, jolloin siirrytään suoraan vaiheeseen yhdeksän. Näitä vaihtoehtoja tullaan vertailemaan myöhemmin.
8. Lomakemoottori käy hakemassa tietokannasta päätökseen liittyvän tietojoukon.
9. SSRS käyttää saatua tietojoukkoa hyväkseen ja hoitaa päätöstulosteen visuaalisen muotoilun
10. Lomakemoottori palauttaa päätöstulosteen käyttäjän selaimeen PDF-dokumenttina.

Järjestelmän ympäristö on jo olemassa, ja se on havainnollistettu kuvan 4 avulla. Päätöstuloste tulee sijoittaa osaksi olemassa olevaa kokonaisuutta mahdollisimman sulavasti. On määritelty, että päätöstulosteet toteutetaan SSRS:llä, joten valinta raportointijärjestelmän suhteen selvä. SSRS:ssä on mahdollista julkaista raportteja suoraan Report Serveriltä käyttäjille selaimeen aukeaviksi sekä määritellä eri raporteille ja käyttäjille käyttöoikeuksia. Tietoturvasyistä johtuen on kuitenkin määritelty, että HR-järjestelmässä kaiken selaimen ja ohjelman välisen tiedonsiirron tulee tapahtua lomakemoottorin kautta, mikä poissulkee juuri esitellyn tavan kutsua SSRS:ää suoraan selaimesta. Päätöstulosteen tuominen selaimeen tullaan siis hoitamaan

lomakemoottorin kautta, kuten on määritelty. Päätöstuloste tuodaan käyttöliittymälle jommallakummalla kuvassa 4 esitetyistä tavoista.

3.2 Datalähde

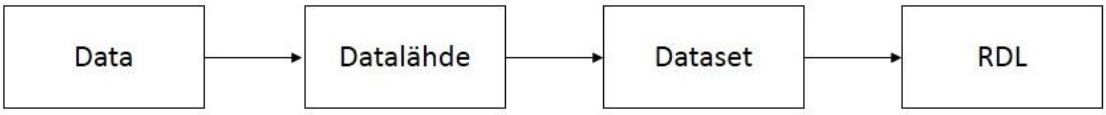
Päätöstulosteelle tarvitaan luonnollisesti päätökseen liittyvää dataa. Tämä tarkoittaa sitä, että päätöstulosteen RDL-tiedostolle tarvitaan päätökseen liittyvää dataa. Päätökseen liittyvä muuttuva data sijoitetaan kuvan 3 mukaisesti oikean puolimmaiseen sarakkeeseen. Lisäksi RDL-tiedostolle tarvitaan pääotsikko sekä kaikki vasemman sarakkeen otsikot. Päätöstulosteelle tarvitaan siis joukko hakemuskohtaisia vakiotekstejä otsikoita varten, sekä hakemuksen sisällöstä riippuvat muuttuvat arvot, jotka muodostavat tulosteen datasisällön. Täytyy muistaa, että päätöstulosteen rivien määrä sekä sisältö vaihtelevat hakemuksesta riippuen. Järjestelmään voi tulla toteutettavaksi noin kymmenen erilaista lomapäätöstulostetta ja noin 30 poissaolopäätöstulostetta, jotka kaikki pitäisi pystyä mahdollisimman helposti muodostamaan.

Hakemuksesta aletaan tallentaa kaikkia siihen liittyviä oleellisia tietoja tietokantaan siitä hetkestä alkaen, kun hakemus on luotu. Tietokannassa on tallessa kaikki päätöstulosteelle tarvittava tieto siitä hetkestä alkaen, kun hakemuksesta on tehty päätös. Kaikki päätöstulosteelle tarvittava datasisältö on siis tallessa, mutta ei vielä RDL-tiedoston käytettävissä.

Tietokannassa on eräässä taulussa jokaista tehtyä hakemusta kohden yksi rivi. Taulussa on yksi kenttä, johon on tallennettu hakemuksesta kaikki perustiedot sekä historia päätöksenteon kulusta. Tietokannan kentän sisältö on XML-muodossa ja se sisältää yli sata yksittäistä tietoa. Kentästä saadaan kaikki päätöstulosteen tarvitsema tietosisältö. Vuosilomahakemuksen päätöstulosteelle tulee 14 rivin verran dataa. Voidaan siis tehdä johtopäätös, että XML:ssä on todella paljon päätöstulosteelle turhaa tietoa.

RDL-tiedostolle voidaan välittää sen tarvitsema tietosisältö muutamalla eri tavalla, joista kaksi päätöstulosteelle sopivinta valittiin vertailtavaksi. Ensimmäinen vaihtoehto on hakea päätöstulosteen tarvitsema tietojoukko XML-muodossa suoraan tietokannasta RDL:lle, jossa siitä voi parsia Element Path -syntaksia (avataan luvussa

3.2.1) käyttämällä haluttuja tietoja RDL:än käyttöön. Toinen tapa on hakea lomakemoottorin koodissa XML-muotoisesta datasta ainoastaan päätöstulosteen tarvitsemat kentät ja lähettää ne listana RDL:lle, joka ottaa käyttöön ne kaikki.



Kuva 5. RDL-tiedostolle datan tuomiseen tarvittavat elementit.

RDL-tiedostolle datan tuomiseen tarvittavat elementit on esitelty kuvassa 5. Datalla tarkoitetaan lähes mitä tahansa dataa, joka on ohjelmallisesti saatavilla. Se voi olla esimerkiksi kovalevyllä oleva XML-tiedosto tai jollain palvelimella sijaitseva tietokanta. SSRS-projekti tarvitsee normaalisti toimiakseen kolme osaa: datalähteen, datasetin ja RDL-tiedoston. Nämä osat ovat kuvan kolme oikeimman puoleista laatikkoa ja ne ovat yhteydessä toisiinsa kuvan osoittamassa järjestyksessä. Datalähde osoittaa mistäpäin, millä tekniikalla ja millaisessa muodossa data haetaan. Datasetin avulla valitaan mitä tietoja datalähteen tarjoamasta datasta valitaan. Lopulta RDL-tiedostolle voidaan sijoittaa datasetiltä valittuja tietoja ja hoitaa datan visuaalinen muotoilu ja esittäminen. Esimerkkinä näiden kolmen osan yhteistyöstä voisi olla kokonaisuus, jossa datana on tietokanta jollakin palvelimella ja datalähde kertoo tietokannan nimen ja sijainnin. Datasetillä määritetään tietokantakysely, joka ottaa muutaman kentän tulokseksi. Lopuksi RDL-tiedostolle sijoitetaan datasetin arvot.

3.2.1 Tietokanta

Kun tietokanta on datalähde, voidaan hakea hakemuksen päätöstulostetta varten tarvittava XML-muotoinen tietojoukko suoraan tietokannasta RDL-tiedostolle. Tätä varten SSRS-projektiin tulee luoda datalähde, joka kertoo tietokannan nimen ja sijainnin sekä käyttäjätunnukset tietokannan käyttöä varten. Datasetin avulla luodaan kysely datalähteen osoittamaan tietokantaan, josta tarpeellisten parametrien avulla saadaan oikea rivi ja kenttä, josta XML-muotoinen sisältö saadaan purettavaksi. Tämän jälkeen Element Path -syntaksin avulla tulee purkaa päätöstulosteelle

tarvittavat kentät ulos XML:stä. Element Path -syntaksi on nimensä mukaisesti oma syntaksi XML-muotoisen tiedon käsittelyä varten. Esimerkkikoodissa 1 on esimerkki syntaksin käytöstä. Esimerkissä saadaan haettua vain osa päätöstulosteelle tarvittavasta tietojoukosta, joten vastaavia kyselyjä tulisi tehdä enemmän, jotta kaikki tarvittava tieto olisi saatavilla.

```
<Query>
        <ElementPath>
        Document{}/Informer{}/Person
        </ElementPath>
</Query>
```

Esimerkkikoodi 1. Esimerkki Element Path -syntaksista, jossa XML-tiedostosta haetaan tietoja hakemuksen tekijästä.

Päätöstulosteelle tarvittava tietojoukko saadaan, kun kyselyjä tehdään tarpeeksi, mutta päätöstulosteen otsikot puuttuvat yhä. Visual Studiossa on RDL-tiedostoa suunniteltaessa käytössä joukko erilaisia kontrolleja, joista tämän työn kannalta tärkein on taulukko. Taulukkoon voi valita halutun määrän rivejä ja sarakkeita tai niiden määrän voi laittaa mukautumaan muuttuvan datan määrään. Vuosilomahakemuksen päätöstulostetta suunniteltaessa voidaan laittaa 14-rivinen ja 2-sarakkeinen taulukko, jonka vasemman puoleisiin sarakkeisiin asetetaan vakio-otsikot määrittelyn mukaan ja oikean puoleisiin sarakkeisiin asetetaan XML-muotoisesta sisällöstä puretut arvot oikeille paikoilleen.

Lomahakemuksien päätöstulosteita on noin kymmenen erilaista, joten taulukon vasemmassa sarakkeessa olevat vakio-otsikot yhdellä RDL-tiedostolla eivät tulisi toimimaan. RDL:lle pitää kehittää logiikka, jonka perusteella se valitsee näytettävät otsikot lomahakemuksen tyypistä riippuen. Tämä vaatii ehtojen kirjoittamista itse RDL-tiedostoon, mutta kaikille lomapäätöksille saadaan silloin yksi yhteinen RDL-tiedosto. Vaihtoehtoisesti jokaiselle lomahakemukselle voi tehdä oman RDL-tiedostonsa ja sijoittaa niille omat taulukot vakio-otsikoilla sellaisenaan ilman mitään logiikkaa.

3.2.2 WCF-palvelu

Kun WCF-palvelu on datalähde, XML-tiedosto haetaan kannasta ja puretaan lomakemoottorilla sijaitsevassa koodissa päätöstulosteelle sopivaan muotoon. WCF (Windows Communication Foundation) on framework palvelukeskeisten ohjelmien kehittämiseen. Lomakemoottorilla on käytössä C#-kieli ja .NET-luokkakirjastot, joita käytetään XML-tiedoston purkamiseen. Lisäksi koodissa on valmiiksi olemassa yleisiä metodeja hakemuksen XML-tiedoston purkuun ja käsittelyyn. Näiden työkalujen avulla XML-sisällön purku päätöstulosteelle sopivaan muotoon sujuu helposti.

Päätöstuloste koostuu kahdesta sarakkeesta, joista toinen on aina vakioteksti ja toinen muuttuva arvo. Koodissa voidaan siis luoda valmiita olioita, joilla on otsikko ja sen sisältämä arvo. Näitä olioita kasataan listaan ja palautetaan lista RDL-tiedostolle. Koodissa voidaan luoda erilaisia listoja hakemuksen syystä riippuen, jotta jokainen hakemus saisi omat kenttänsä oikein päätöstulosteelle. Näin RDL-tiedostolle asti ei tarvitse tuoda logiikkaa, vaan se pysyy lomakemoottorilla keskitettynä.

Ratkaisua varten täytyy julkaista WCF-palvelu, jotta koodissa luotu lista saadaan välitettyä RDL-tiedostolle. WCF-palveluiden idea on, että muut ohjelmat voivat kutsua niiden metodeja, jolloin WCF-palvelu suorittaa metodinsa aina samalla tavalla riippumatta, mistä sitä on kutsuttu. WCF-palvelua kutsuvan ohjelman ei tarvitse tietää WCF-palvelun logiikkaa. Riittää, että kutsuva ohjelma tietää, miten WCF-palvelua kutsutaan, mitä tietoja pitää lähettää parametreina ja mitä tietoja WCF-palvelulta saadaan takaisin. Nämä tiedot WCF-palvelusta saadaan selville sen metadatasta, joka julkaistaan WSDL-muodossa (Web Service Description Language). [3, s. 3.] WCF-palvelun osoite pitää tietää, jotta sitä voidaan kutsua. Osoite voidaan laittaa selaimeen, jolloin WCF-palvelun rajapintaa, eli metadataa WSDL-muodossa voidaan tutkia. Rajapinnasta näkee esimerkiksi julkaistut metodit ja niiden SOAP-kutsun tekemiseen tarvittavat tiedot. SOAP (Simple Object Access Protocol) on tietoliikenneprotokolla metodien etäkutsua varten. SOAP-kutsulla haetaan päätöstulosteen tarvitsema tietojoukko WCF-palvelussa julkaistulta metodilta. Esimerkkikoodin 2 esimerkissä kutsutaan WCF-palvelua, joka vastaanottaa kolme parametria.

```
<Query>
        <Method Name="HaePaatostuloste" Namespace="http://lomakemoottori.fi/tulosteet">
                <Parameters>
                        <Parameter Name="hakemustyyppi" Type="String"></Parameter>
                        <Parameter Name="hakemuskohtainenTunniste" Type="String"></Parameter>
                        <Parameter Name="kieli" Type="String"></Parameter>
                </Parameters>
        </Method>
        <SoapAction>
        http://lomakemoottori.fi/tulosteet/IPaatostuloste/HaePaatostuloste
        </SoapAction>
</Query>
```

Esimerkkikoodi 2. Esimerkki SOAP-kutsusta, jossa kutsutaan kuvitteellisen WCF-palvelun metodia, joka ottaa vastaan kolme parametria.

Tätä ratkaisua varten tulee SSRS-projektiin määrittää datalähde, joka kertoo WCF-palvelun sijainnin ja tarvittavat käyttäjätunnukset WCF-palvelun käyttämistä varten. Datasetissä määritetään metodi, jota datalähteen määrittämästä WCF-palvelusta halutaan kutsua ja tarvittavat parametrit. Esimerkkikoodi 2 on esimerkki datasetin kyselystä. Datasetistä voidaan taas valita halutut kentät ja sijoittaa ne RDL-tiedostolle.

Kun koodi on tehty ja WCF-palvelu on käynnissä, saadaan RDL-tiedostolle päätöstulosteen tarvitsema tietojoukko sitä kautta. WCF-palvelulta kutsuttavalle metodille kerrotaan parametreissa hakemuksen tyyppi, hakemuskohtainen tunniste sekä käytettävä kieli, jotta WCF-palvelu osaa hakea tietokannasta oikean hakemuksen tiedot ja palauttaa oikeannäköisen listan oikeaa sisältöä päätöstulosteelle. Tällä tyylillä RDL-tiedosto voi olla hakemuksen syystä riippumatta yksi ja sama. RDL-tiedostolle voidaan sijoittaa päätöstulosteen otsikko sekä yksi taulukko, jolla on kaksi saraketta, muuta ei tarvita. Taulukon vasemman puoleiseen sarakkeeseen sijoitetaan WCF-palvelulta saadun listan olion otsikko-ominaisuus ja oikean puolen sarakkeeseen sijoitetaan olion arvo-ominaisuus. Taulukon ominaisuuksista voi valita, että se luo niin monta riviä kuin datasetistä siihen on tavaraa tulossa. Tällä tavalla voidaan tuoda samalle RDL-tiedostolle 14-rivisen päätöstulosteen tiedot tai vaikka 3-rivisen päätöstulosteen tiedot, ja se toimii molemmissa tapauksissa.

3.3 Yhdenmukaisuus ja hallittavuus

Kuten johdannossa kerrottiin, tätä työtä tehdessä pitää varautua siihen, että tulevaisuudessa projektiin halutaan uusia päätöstulosteita. On myös otettava huomioon, että toteutettavat päätöstulosteet tulee olla jatkossa helposti hallittavissa. Jos päätöstulosteille haluttaisiin jatkossa esimerkiksi sijoittaa logo yläkulmaan, pitää se olla helposti lisättävissä kaikkiin päätöstulosteisiin.

Monissa asioissa halutaan, että runko on yhtäläinen, mutta sisältö muuttuu. Otetaan esimerkkinä WWW-sivusto, jonka sisältö muuttuu navigoimalla sivustoa ympäri. Ympärillä kuitenkin pysyy sama ulkoasu, joka viestii väreillään ja logoillaan käyttäjälle kyseessä olevan yhä sama sivusto. Tällainen raami on yksinkertainen toteuttaa WWW-sivustolle esimerkiksi ASP.NET-teknologiaa käyttämällä. Siinä määritetään MasterPage, josta tehdään juuri sen näköinen kuin halutaan. MasterPagen sisälle luodaan määritettyjä paikkoja, joihin voi sijoittaa itse sisällön. Sisältö voi muuttua käyttötarkoituksen mukaan miksikä vain, mutta sisällöstä riippumatta sivun ulkoasu pysyy koko ajan yhtenäisenä ympärillä olevan MasterPagen avulla. [4.] Mikäli SSRS:ssä olisi vastaavaa MasterPage olemassa, olisi päätöstulosteiden yhdenmukaisuus ja hallittavuus helposti hoidettavissa. Asia ei kuitenkaan näin ole, vaan joudutaan etsimään vaihtoehtoista ratkaisua, jolla voitaisiin pitää eri tulosteiden yhteisten osien kontrolloitavuus yhdessä paikkaa. Emmehän halua lisätä uutta logoa sadalle RDL-tiedostolle yksi kerrallaan.

Report Template

Report Template, eli mallipohja on tavallinen RDL-tiedosto, joka suunnitellaan ja luodaan samalla tavalla kuin mikä tahansa RDL-tiedosto. Ainoa huomio on, että se pitää viedä tiettyyn kansioon SSRS:n versiosta riippuen, jotta Visual Studio osaa jatkossa ehdottaa sen käyttöä uutta RDL-tiedostoa luodessa. [5.]

Kun raporttien yhteiset ulkoasutekijät ovat tiedossa, voidaan niiden pohjalta luoda Report Template, jossa raporteille yhteiset osat ovat valmiiksi tehtynä. Report Template toimii samalla tavalla kuin esimerkiksi Word-template, eli uutta RDL-tiedostoa tehdessä voidaan se luoda haluttuun Report Templateen pohjautuen. Näin jokaiselle projektiin luodulle RDL-tiedostolle saadaan perittyä yhteneväinen ulkoasu. Report

Templatesta luotujen raporttien sisältöä voidaan kuitenkin myöhemmin muokata vastaamaan raporttikohtaisia määrityksiä.

Kun raportti on perustettu Report Templatesta, ei raportilla ja Report Templatella ole tämän jälkeen enää mitään yhteyttä toisiinsa. Report Templateen voidaan tehdä muutoksia tai se voidaan jopa poistaa kokonaan, eikä itse raportille tapahdu mitään. Sama toimii myös toisin päin, eli raportille voidaan tehdä muutoksia tai poistaa ja Report Template pysyy ennallaan.

Subreport

Subreport on nimensä mukaan raportilla oleva raportti eli RDL-tiedosto RDL-tiedoston sisällä. Visual Studiossa pystyy lisäämään suunnitteilla olevalle raportille Subreport-kontrollin, jonka sijainnin ja koon pystyy määrittämään. Subreportiksi pitää valita jokin olemassa oleva RDL-tiedosto, ja sille voi antaa parametreja. Kun käyttäjä haluaa nähdä raportin, suoritetaan sekä itse raportti että sen Subreport, mutta käyttäjälle ei välity mitään tietoa, että näin tapahtuu. Raportti tulostuu ihan normaalisti näytölle. [6.]

Subreportia voi käyttää hyväksi esimerkiksi luomalla omat RDL-tiedostot kaikille projektin raporteille yhteisestä ylätunnisteesta ja alatunnisteesta. Kun kaikki projektin RDL-tiedostot käyttävät yhtä yhteistä ylätunnistetta ja yhtä yhteistä alatunnistetta, on raporttien hallittavuus yhdessä paikassa. Kun Subreportia muokkaa, päivittyvät muutokset myös saman tien sitä käyttäville raporteille. Jos projektissa olisi esimerkiksi sata erillistä raporttia ja ylätunnisteessa oleva logo haluttaisiin muuttaa, tarvitsisi muutos tehdä vain yhteen paikkaan projektissa ja logo vaihtuisi kaikille raporteille.

Report Part

Report Part on RDL-tiedoston osa, joka julkaistaan Report Serverille. Tämä osa voi olla esimerkiksi taulukko, joka halutaan usealle raportille samanlaisena. Jokainen raportti voi muokata lokaalisti Report Partia oman näköisekseen ilman, että tekee muutoksia serverille julkaistuun Report Partiin. Report Partiin voi tehdä muutoksia myös niin, että ne julkaistaan serverille. Tämän jälkeen avattaessa RDL-tiedosto, joka käyttää Report Partia, kysytään käyttäjältä haluaako hän päivittää sen uuteen versioon. Tässä vaiheessa käyttäjä voi siis joko pitää Report Partin juuri sille raportille muokattuna tai hakea uusimman version serveriltä. [7.]

Yksi ainoa RDL-tiedosto

Mikäli projektissa on ainoastaan yksi RDL-tiedosto, jonka tietosisältöä muuttamalla saadaan kaikki tarvittavat raportit muodostettua, pysyy raporttien hallinta yksinkertaisena jo sellaisenaan. Kun kaikki eri raportit pohjautuvat yhteen RDL-tiedostoon, muokkaamalla pelkästään sitä ulottuvat muutokset luonnollisesti jokaiselle eri raportille. Tämä ratkaisu vaatii tosin sen, että kaikki raportit noudattavat samaa perusrakennetta, joka RDL-tiedostolle on luotu.

3.4 Valitut tekniikat

Esiteltyihin tekniikkoihin valittiin työn kannalta soveltuvimmat. Muitakin tekniikoita haluttujen ominaisuuksien toteuttamiseen olisi ollut, mutta ne eivät olleet tämän työn luonteeseen sopivia. Kun valitaan tapa toteutukselle, täytyy miettiä sen soveltuvuutta projektin jo olemassa oleviin osiin. Valitulla tavalla pitää tietysti saada myös asetetut tavoitteet täytettyä.

3.4.1 Datalähde

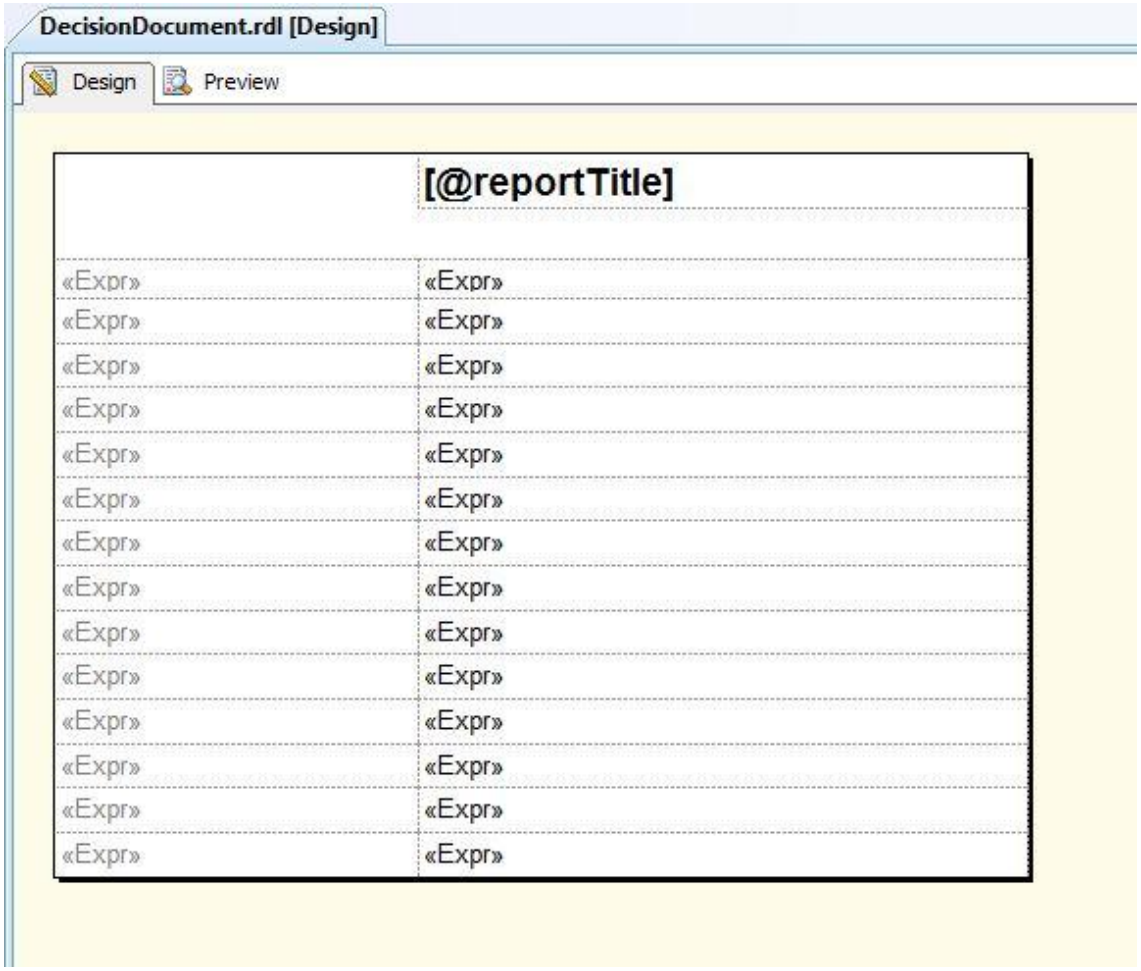
Päätöstulosteiden datalähteiden vertailuun valittiin vain kaksi tapaa. Molempien tapojen toimivuutta projektissa mietittiin etukäteen, jonka jälkeen niitä testattiin käytännössä. Tehtyjen testien perusteella valittiin paremmalta vaikuttava tapa.

Kun tietokanta on datalähteenä, päätöstulosteen tarvitsema XML-tiedosto tulee käsiteltäväksi sellaisenaan, jolloin lähes kaiken päätöstulosteisiin liittyvän logiikan pystyy sijoittamaan RDL-tiedostolle. Näin tulosteisiin liittyviä muutoksia voitaisiin hallita selkeästi yhdestä paikkaa. Mikäli kaikille tulosteille haluttaisiin XML-tiedostosta samat kentät, olisi tämä yksinkertainen tapa toteutukselle. Hakemuksesta riippuen XML-tiedostosta halutaan kuitenkin eri kenttiä näkyviin tulosteelle. Logiikka tämän toiminnon toteutukseen RDL-tiedostolla menee monimutkaiseksi toteuttaa. XML-tiedosto, josta päätöstulosteelle tuleva tieto poimitaan sisältää paljon eri tasoja, eikä Element Path -syntaksilla saa kaikkia haluttuja kenttiä poimittua kerralla. Tämä tarkoittaa käytännössä sitä, että esimerkiksi hakemuksen tekijän tiedot pitää hakea omalla kyselyllä ja hakemuksen tekijän yrityksen tiedot omalla. Vastaavia kyselyitä pitää tehdä monia jo pelkästään yhden hakemuksen päätöstulostetta varten. Ottaen huomioon

tulevaisuuden ja mahdollisen tarpeen uusille päätöstulosteille, olisi tämä työläs tapa toteuttaa ne kaikki.

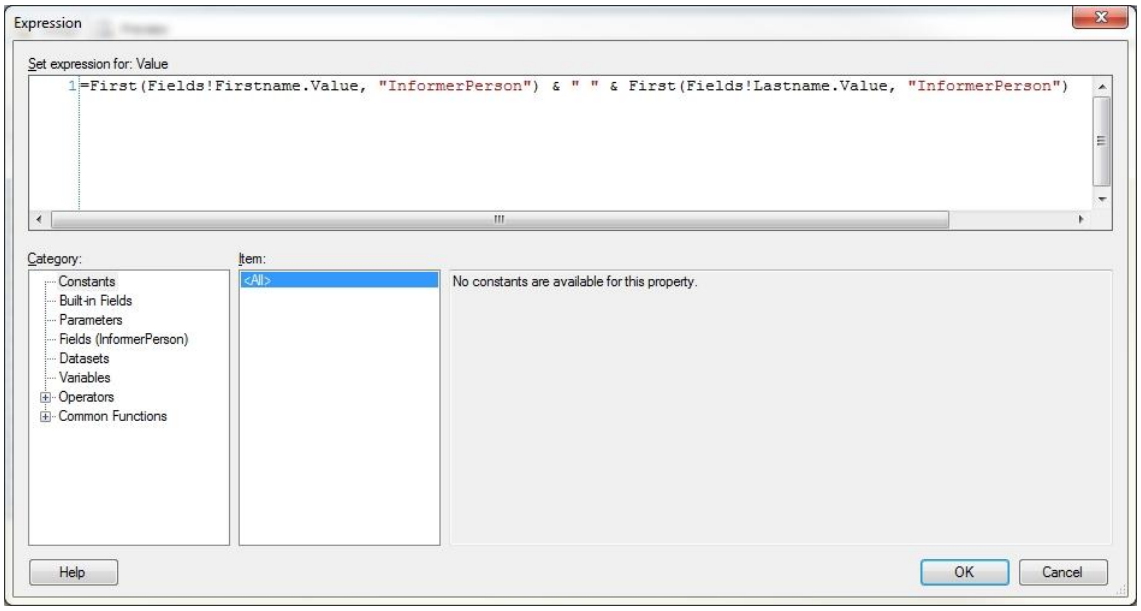
Kun WCF-palvelu on datalähteenä, pystytään pitämään RDL-tiedosto yksinkertaisena. Kaikki logiikka datan käsittelyyn liittyen menee WCF-palvelun koodiin lomakemoottorille. Koodissa on helppo käsitellä dataa, luoda sääntöjä eri hakemuksia varten ja lopulta lähettää RDL-tiedostolle hakemuksesta riippuen tietty joukko dataa. Koodissa saadaan XML-tiedostolta hakemustyyppi selville, jonka perusteella datan käsittely siirtyy hakemustyypin perusteella valittuun metodiin, joka luo hakemuskohtaisesti päätöstulosteen tarvitseman tietojoukon lähetettäväksi RDL-tiedostolle.

Datalähteen valinnassa pidettiin tärkeänä RDL-tiedoston selkeyttä. Vaikka RDL-tiedostoon pystyy sisällyttämään logiikkaa ja koodia, on sen pääasiallinen tarkoitus kuitenkin datan visuaalinen muotoilu. Kuvassa 6 on RDL-tiedosto, jonka datalähteenä on tietokanta ja XML-tiedoston käsittely on suoritettu kokonaan RDL-tiedostolla.



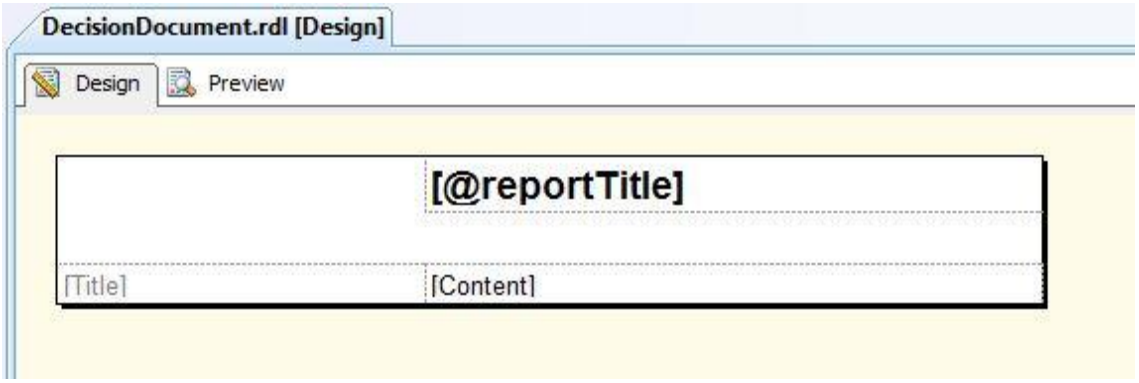
Kuva 6. RDL-tiedosto, jonka datalähteenä on tietokanta.

Taulukossa on 14 riviä, jotta kaikki vuosilomahakemuksen päätöstulosteen tiedot saadaan paikoilleen. Jokaisessa kentässä lukee ”<<Expr>>”, joka tarkoittaa sitä, että kentässä on logiikka, jolla sen arvo päätellään. Esimerkkinä kuva 7, jossa avataan, mitä ”<<Expr>>”-kentän sisällä oleva logiikka voisi olla.



Kuva 7. Kentän sisällöksi, jolla on tämä ”<<Expr>>” tulee XML-tiedostosta haetun hakemuksen tekijän etunimi ja sukunimi, jotka ovat eroteltuina välilyönnillä.

Kuvassa 8 on RDL-tiedosto, jonka datalähteenä on WCF-palvelu ja päätöstulosteen tarvitsema tietojoukko on saatu valmiina listana. Vuosilomahakemuksen päätöstulostetta varten tarvitsemia tietoja on 14 riviä. RDL-tiedosto tarvitse kuitenkin vain yhden rivin taulukkoon, jonka vasemman puoleiseen sarakkeeseen sijoitetaan rivin otsikko ja oikean puoleiseen sarakkeeseen itse arvo. Taulukko generoi uusia rivejä niin monta, kuin WCF-palvelun palauttamassa listassa on olioita. Vertaamalla kuvia 6 ja 8 huomataan selkeä ero RDL-tiedostojen yksinkertaisuudessa.



Kuva 8. RDL-tiedosto, jonka datalähteenä on WCF-palvelu.

Olemassa olevassa projektissa on lomakemoottorissa valmiina metodeja XML-tiedoston läpi käymiseen. Näitä metodeja hyväksi käyttämällä, saadaan säästettyä

työtä ja data saadaan päätöstulosteen RDL-tiedostolle WCF-palvelusta valmiiksi pilkottuna. Datalähteen ollessa suoraan tietokannasta haettu XML-tiedosto, tulee yhdelle RDL-tiedostolle useita kyselyitä XML-tiedostoon sekä logiikkaa eri kenttien näyttämissääntöjä varten. Molempia tapoja koetettiin käytännössä, ja molemmilla tavoilla päätöstuloste saatiin muodostettua. Kun tietokanta oli datalähteenä, tuli toteutusta tehdessä vastaan Element Path -syntaksin kömpelyys ja RDL-tiedoston monimutkaisuus. Kun taas datalähteenä oli WCF-palvelu, saatiin RDL-tiedostosta niin yksinkertainen kuin mahdollista. Datalähteen toteutustavaksi valittiin WCF-palvelu, koska sen avulla RDL-tiedosto saadaan pidettyä yksinkertaisena ja logiikka sijoitettua koodiin, jossa sitä on helppo hallita.

3.4.2 Yhdenmukaisuus ja hallittavuus

Yhdenmukaisuuteen ja hallittavuuteen ei löytynyt tapaa, jolla voitaisiin varmistaa tulevaisuudessa minkä tahansa tyyppisen tulosteen sijoittaminen helposti osaksi projektia ja säilyttää silti yhdenmukaisuus ja hallittavuus. Valinnassa mietittiin erityisesti tämän hetken tulostetarvetta ja oletettiin, että jatkossa tulevat uudet päätöstulosteet noudattavat samaa ulkoasua.

Koska päätöstulosteen ulkoasu on yksinkertainen ja RDL-tiedostolle saadaan tuleva data valmiiksi pilkottuna, valittiin toteutustavaksi yksi ainut RDL-tiedosto. Sillä on määritelty erikseen paikka päätöstulosteen otsikolle, kentän otsikolle ja kentän arvolle. Niihin sijoitetaan koodista saadut arvot, joten ulkoasu pysyy koko ajan samana, ainoastaan sisältö muuttuu. Samaan rakenteeseen voi sijoittaa minkä tahansa päätöksen tiedot, eikä RDL-tiedostoon tarvitse koskeakaan uuden päätöstulosteen luomisen yhteydessä. Jatkossa voidaan uutta päätöstulostetta varten luoda koodissa uusi metodi, joka kasaa tarvittavan tietojoukon ja lähettää sen RDL-tiedostolle vastaanotettavaksi.

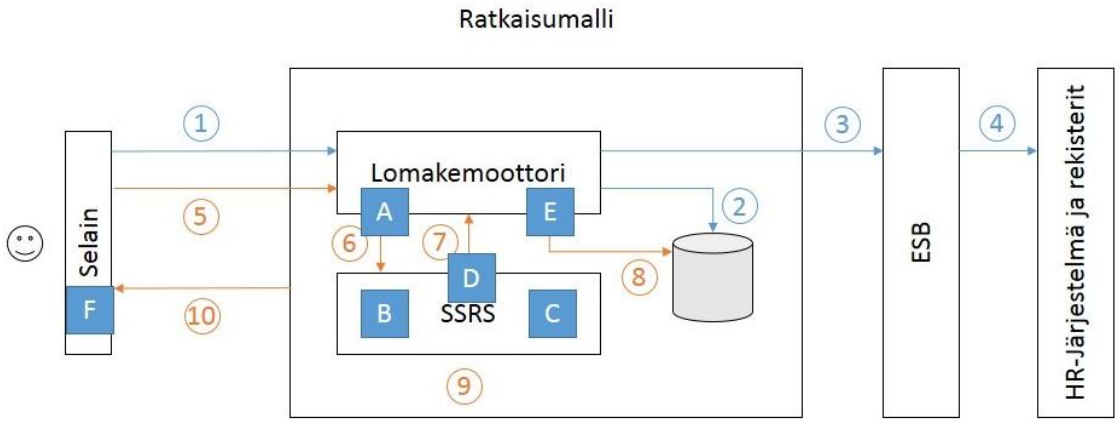
Kaikilla esitellyillä tekniikoilla on omat hyvät ja huonot puolensa. Kaikki ne sopisivat hyvin johonkin käyttötarkoitukseen. Toteutustapa valittiin kuitenkin nimenomaan tämän projektin tarpeita vastaavaksi. Report Template hylättiin saman tien, koska siitä ei ole minkäänlaista apua yhdenmukaisuuden hallinnassa, jos olemassa oleviin RDL-tiedostoihin haluttaisiin tehdä muutoksia. Report Part ja Subreport olivat kovassa vertailussa, vaikka kumpikaan ei vaikuttanut täysin toimivalta vaihtoehdolta. Työn kannalta hallittavuuden ratkaisuja arvioidessa kävi kuitenkin niin, että datalähteen

valinnasta johtuen tuli mahdollisuus yhden RDL-tiedoston käyttöön. Alun perin tällaista vaihtoehtoa ei ollut edes harkinnassa, mutta päätöstulosteiden hahmottelemisen myötä näinkin yksinkertainen ratkaisu nousi esille. Se on kaikista yksinkertaisin mahdollinen tapa hallita päätöstulosteiden ulkoasua, koska siinä on ainoastaan yksi tiedosto, johon muutokset tulee tehdä. Toteutustavaksi valittiin siis yksi anoa RDL-tiedosto.

4 Toteutus

4.1 Toteutuksen yleiskuva

Kuvassa 4 käytiin aiemmin läpi päätöstulosteen sijoittumista osaksi olemassa olevaa raamia. Tätä sijoittumista avataan vielä tarkemmin ja käydään läpi yksityiskohtaisemmin, mitä tapahtuu ja missä. Tässä vaiheessa on jo tehty päätös siitä, että SSRS ei hae päätöstulosteen tarvitsemaa tietojoukkoa suoraan tietokannasta, vaan se haetaan lomakemoottorilla sijaitsevan WCF-palvelun kautta. SSRS-laatikosta suoraan tietokantaan menevä nuoli on selkeyden vuoksi poistettu kuvasta 9. Kuvaan 9 on lisäksi sijoitettu viisi laatikkoa aiemman kuvan sisään. Ne ovat osat, jotka kommunikoivat toistensa kanssa keskenään.



Kuva 9. Päätöstuloste ohjelmistoratkaisun näkökulmasta tarkemmin avattuna.

A. Report.aspx-sivu (aspx avataan luvussa 4.4) vastaanottaa HTTP-pyynnön selaimelta (Anna päätöstuloste PDF:nä). Tämä osio muodostaa päätöstulostepyynnön ja hakee päätöstulosteen, jonka se palauttaa PDF:nä selaimelle. Tämä osio on toteutettu, koska käyttäjän tietoliikennettä ei haluta päästää suoraan raportointipalvelimelle johtuen tietoturvavaatimuksista. Toteutustapana on käytetty ASP.NET-teknologiaa ja C#-ohjelmointikieltä.

B. Päätöstuloste, eli RDL-tiedosto hakee datasetin (laatikko-C) ja datalähteen (laatikko-D) kautta tietojoukon ja vastaa tulosteen ulkoasun muotoilusta. Toteutustapana on käytetty SSRS:ää.

C. Dataset määrittelee käytetyn metodin datalähteen osoittamalta WCF-palvelulta sekä metodin tarvitsemat parametrit. Toteutustapana on käytetty SSRS:ää.

D. Datalähde määrittelee tietojoukon hakupaikan eli WCF-palvelun sijainnin. Toteutustapana on käytetty SSRS:ää.

E. Päätöstuloste-WCF-palvelu vastaa päätöstulosteen tarvitsemien tietojen hakemisesta sekä tuottaa päätöstulosteen RDL-tiedoston tarvitseman tietosisällön. Päätöstuloste-WCF-palvelu palvelee kaikkien hakemustyyppien kaikkia yksittäisiä hakemuksia. Päätöstuloste-WCF-palvelu on tehty sen takia, että, jokaisen hakemuksen yksilöllinen tietosisältö saadaan muodostettua yhdessä keskitetyssä paikassa ja samalla saadaan yksinkertaistettua varsinaisen päätöstulosteen käsittely. RDL-tiedosto saadaan siis pidettyä mahdollisimman yksinkertaisena. Toteutustapana on käytetty WCF-teknologiaa, C#-ohjelmointikieltä ja .NET-frameworkin LINQ to SQL -komponenttia, jolla on toteutettu tietokantakyselyt.

F. Käyttäjällä auki oleva selain vastaanottaa päätöstulosteen PDF-muodossa. Käyttäjä voi tarvittaessa tulostaa päätöstulosteen paperille, tallentaa sen, tai avata PDF-lukuohjelmalla.

4.2 Datalähteen toteutus

Kuvan 9 laatikko-D eli päätöstulosteen datalähde päätettiin toteuttaa WCF-palveluna. Datalähteen toteutuksessa RDL-tiedostoon ei kohdistunut paljon töitä. Sen sijaan itse WCF-palvelun toteuttaminen oli luonnollisesti suuressa roolissa.

WCF-palvelu päätöstulosteita varten luotiin osaksi olemassa olevaa lomakemoottorin koodiprojektia. WCF-palvelu on kirjoitettu C#-kielellä, ja se käyttää hyväkseen .NET-Frameworkia sekä koodiprojektin valmiita luokkia ja metodeja, jotka auttavat hakemuksen tietoja sisältävän XML-tiedoston käsittelyssä. Lisäksi koodiin täytyi rakentaa omaa logiikkaa ja luokkia hakemusten päätöstulosteita varten.

WCF-palvelu palauttaa RDL-tiedostolle listan olioita, jotka ovat muodoltaan nimi-arvo-pareja. Nimi-arvo-parit ovat olioita, joista nimi-ominaisuus sisältää päätöstulosteen rivin otsikon ja se sijoitetaan päätöstulosteen taulukon vasemman puoleiseen sarakkeeseen. Arvo-ominaisuus sisältää nimensä mukaisesti muuttuvan arvon, joka sijoitetaan päätöstulosteen taulukon oikean puoleiseen sarakkeeseen. Kun listaa palautettavista olioista aletaan muodostaa, tarkistetaan ensiksi, mistä hakemustyypistä on kyse, ja ohjataan listan muodostus sitä hakemustyyppiä varten luotuun metodiin, esimerkki hakemustyypistä on lomahakemus. Tässä metodissa lisätään listaan kaikki kyseiselle hakemustyypille yhteiset nimi-arvo-parit, jonka jälkeen tarkistetaan hakemuksen syy. Koska päätöstulosteen sisältö vaihtelee hakemuksen syystä

riippuen, on koodiin tehty logiikkaa, jossa listaan lisätään vielä hakemussyystä riippuvat nimi-arvo-parit, esimerkki hakemuksen syystä on vuosilomahakemus. Näin saadaan jokaiselle eri hakemussyylle luotua uniikit listat, jotka palautetaan RDL-tiedostolle. Lisäksi koodi on toteutettu niin, että siihen on helppo lisätä uusien hakemustyyppien ja hakemusten päätöstulosteiden tarvitsemien tietojoukkojen muodostus, jotta jatkokehitys olisi mahdollisimman sujuvaa.

WCF-palvelussa on julkaistu hakemusten päätöstulosteita varten metodi, joka vastaanottaa seuraavat parametrit:

- Tulostetyyppi kertoo, mistä hakemuksen tyypistä on kyse. Tämän avulla voidaan jatkossa erotella hakemusten eri tyypit koodissa. Tällä hetkellä parametrin arvona on pelkästään koodi, joka ilmoittaa kyseessä olevan tulostetyypin olevan lomapäätöstuloste.

- Hakemuksen tunniste toimii yksilöivänä avaimena ja kertoo mistä hakemuksesta on kyse. Tämän avulla osataan hakea kannasta oikea XML-tiedosto, josta päätöstulosteen sisältö muodostetaan.

- Sisältökieli kertoo, mitä kieltä päätöstulosteella halutaan käytettävän. Tällä hetkellä parametrin arvona on vakiona FI ja päätöstuloste tehdään aina suomeksi. Tämä parametri on tehty jatkoa ajatellen, jotta uusien kielien lisääminen olisi helppoa.

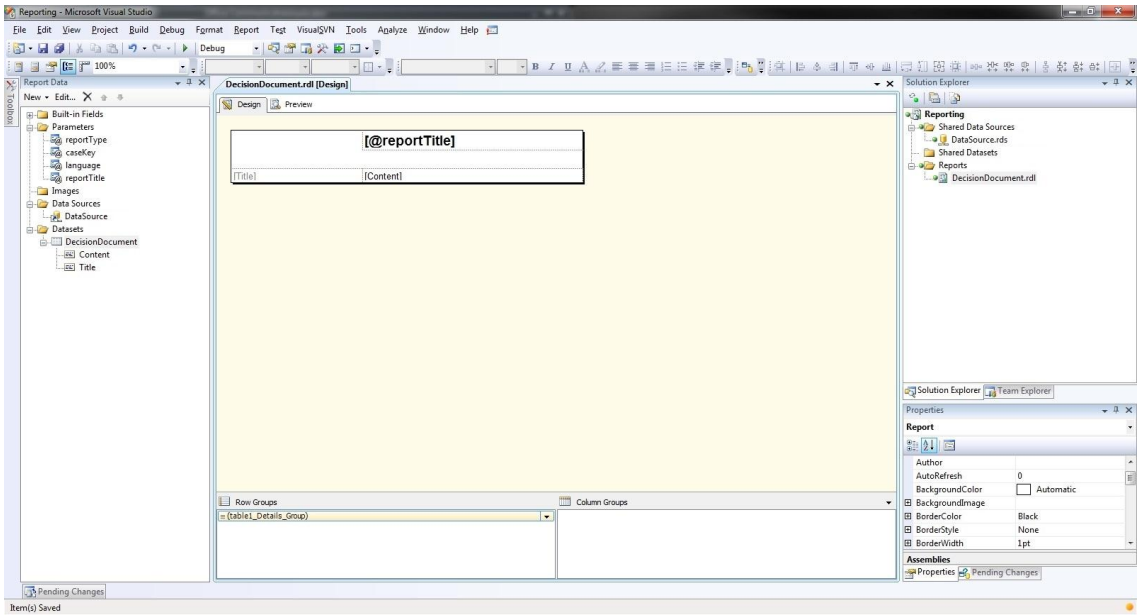
Tätä metodia RDL-tiedosto kutsuu datalähteensä ja datasetinsä avulla saadakseen tarvittavan tietojoukon päätöstulosteen luomiseksi. WCF-palvelu palauttaa RDL-tiedostolle listan nimi-arvo-pareja, jotka sijoitetaan päätöstulosteen RDL-tiedoston taulukkoon, jolloin päätöstulosteen tietosisältö saadaan muodostettua.

Toiminnollisuuden testausta varten on olemassa palvelin, jolla pyörii SSRS sekä lomakemoottori. Palvelimella on siis myös päätöstulosteen luomista varten tarvittava WCF-palvelu käynnissä, koska se on osa lomakemoottoria. SSRS-projektissa tulee kertoa, missä WCF-palvelu sijaitsee, jonka lisäksi tulee määrittää SOAP-kutsu, jossa kerrotaan kutsuttavan metodin nimi ja lähetettävät parametrit. Kuten todettu, datalähde kertoo WCF-palvelun sijainnin ja datasetin SOAP-kutsu kertoo kutsuttavan metodin nimen ja tarvittavat parametrit. Kun datalähde, dataset ja RDL-tiedosto ovat kunnossa, voidaan kokonaisuuden toimivuutta testata kutsumalla WCF-palvelua testiparametreilla. Jos SOAP-kutsu ja WCF-palvelun osoite on määritelty oikein ja annetuilla parametreilla löytyy tietokannasta XML-tiedosto, palauttaa WCF-palvelu RDL-tiedostolle päätöstulostetta varten luodun listan nimi-arvo-pareja. Datasetistä

sijoitetaan RDL-tiedostolla olevan taulun vasemman puoleiseen sarakkeeseen nimi-arvo-pareista nimi ja oikean puoleiseen sarakkeeseen arvo. Kokonaisuuden toimivuutta voidaan testata esimerkiksi Visual Studion tarjoaman Preview-ominaisuuden avulla. Siinä annetaan parametreille arvot, jonka jälkeen napsautetaan View Report -painiketta. Se käy suorittamassa WCF-palvelun sisältämän koodin ja sijoittaa saadut arvot RDL-tiedostolle muodostaen valmiin päätöstulosteen.

4.3 Yhdenmukaisuuden ja hallittavuuden toteutus

Kuvassa 10 nähdään, miltä SSRS-projekti näyttää, kun se on auki Visual Studiossa, jossa on auki kehityksen kannalta oleellisimmat ikkunat. Kuvan projekti on jo toteutettu, ja se sisältää kaikki tarvittavat osat toimiakseen. Kuvan vasemmassa laidassa on Report Data -niminen osio, jonka avulla voidaan hallita RDL-tiedoston käyttämää dataa. Keskellä näyttöä on itse RDL-tiedosto auki Design-moodissa, jonka avulla voidaan luoda päätöstulosteen ulkoasu ja sijoittaa tarvittavat tiedot oikeille paikoilleen. Oikealla puolella näyttöä on Solution Explorer -ikkuna auki, jonka avulla voidaan tarkastella, miltä projektin rakenne näyttää ja mitä eri osia se sisältää.



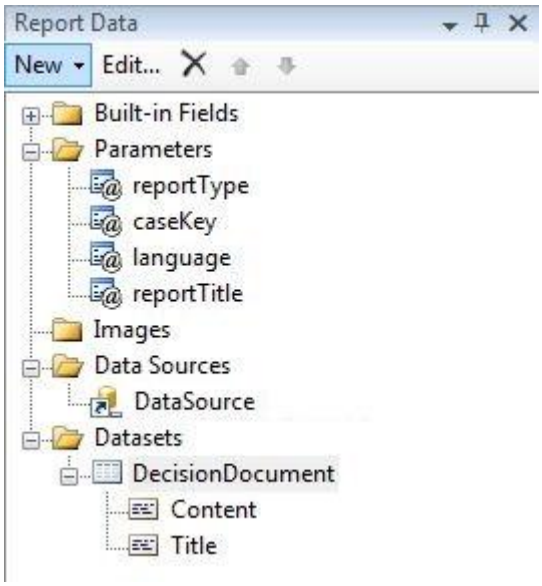
Kuva 10. Yleiskuva Visual Studiosta, jossa on auki SSRS-projekti.

Yhdenmukaisuuden ja hallittavuuden toteutuksen tavaksi valittiin yksi ainoa RDL-tiedosto. Tämän tavan toteuttamisessa ei tarvittu mitään erikoisia työkaluja, vaan luotiin

uusi SSRS-projekti, johon kuului yksi RDL-tiedosto, yksi datalähde, yksi dataset ja neljä parametria. Datalähteen, datasetin ja RDL-tiedoston toteutus tapahtui limittäin, koska kaikkien osien kehitys vaikutti toisiinsa. Solution Explorer -ikkunasta nähdään kuinka yksinkertaisena SSRS-projekti pysyy. Projektiin kuuluu vain yksi RDL-tiedosto ja yksi datalähde, kuten kuvasta 11 nähdään.

Kuva 11. Solution Explorer -ikkunasta voidaan tarkastella projektin rakennetta ja sen sisältämiä tiedostoja.

RDL-tiedostolle tuli muuttaa joitain asetuksia, jotta sen käyttäminen olisi helpompaa. Esimerkiksi mittayksiköksi valittiin senttimetrit ja päätöstulosteen kooksi A4. RDL-tiedostolle määritettiin WCF-kutsua varten tarvittavat kolme parametria. Lisäksi myöhempää käyttöä varten lisättiin päätöstulosteen otsikkoa varten oma parametri. Parametrit näkyvät Visual Studiossa Report Data -nimisessä osiossa Parameters-hakemistossa, kuten kuvasta 12 ilmenee. Lisäksi kuvassa 12 näkyy DataSets-hakemistossa RDL-tiedoston käyttämä datasetti. DecisionDocument-nimisellä datasetillä on kaksi kenttää, jotka se ottaa vastaan WCF-palvelulta kutsuttavalta metodilta. Nämä kentät ovat nimi-arvo-pareja, jotka sijoitetaan RDL-tiedoston taulukkoon. Datasetti ottaa vastaan WCF-palvelun metodilta listan nimi-arvo-pareja, joten niiden määrä voi vaihdella ja nimi-arvo-parien määrästä riippuen RDL-tiedoston taulukko generoi tarpeellisen määrän rivejä, joihin arvot sijoitetaan.



Kuva 12. Report Data -näkymä, jossa Parameters-hakemistossa näkyy RDL-tiedostolle määritetyt parametrit. DataSets-hakemistosta löytyy DecisionDocument-niminen datasetti, joka sisältää WCF-palvelun metodin palauttamat nimi-arvo-parit.

RDL-tiedostolle lisättiin tekstikenttä päätöstulosteen otsikkoa varten ja kaksi-sarakkeinen taulukko nimi-arvo-pareja varten. Tekstikentälle ja taulukon molemmille sarakkeille asetettiin fontit vastaaman kuvan 3 määrityksiä. Datalähteen valinnassa nostettiin esille RDL-tiedoston yksinkertaisuuden tärkeys, ja nyt se tosiaan on yksinkertainen. RDL-tiedostolle ei tarvitse muita kontrolleja lisätä. Kun datalähde ja RDL-tiedosto ovat valmiina, pitää ne ladata Report Serverille, jotta niitä voidaan kutsua sieltä jatkossa mistä vain. Datsetti generoituu RDL-tiedoston XML-kuvauksen mukaan, joten sitä ei tarvitse erikseen viedä. Report Serverille lataaminen (käytetään myös termiä deploy) tapahtuu määrittämällä käytössä olevan Report Serverin tiedot SSRS-projektin TargetURL -kohtaan, jonka jälkeen napsautetaan lataa-toimintoa, jolloin datalähde ja RDL-tiedosto ladataan TargetURL:issa määritettyyn osoitteeseen.

Koska Lomakemoottori-projektissa on käytössä versionhallinta, liitetään SSRS-projekti osaksi sitä. Näin päätöstulosteiden ulkoasuun tehdyt muutokset pysyvät versioituina ja hallinnassa. Päätöstulosteeseen tehdyt muutokset pitää viedä versionhallintaan, jonka lisäksi ne pitää ladata Report Serverille, jotta oikea versio päätöstulosteesta on SSRS:än käytössä.

4.4 Päätöstulosteen tuominen selaimeen teknisesti

WCF-palvelu on toteutettu ja se on käynnissä palvelimella. Lisäksi SSRS on käynnissä palvelimella ja Report Serverille on ladattu datalähde sekä RDL-tiedosto päätöstulosteen muodostamista varten. Enää on jäljellä tämän kokonaisuuden yhdistäminen prosessiksi, jolla saadaan käyttäjälle päätöstuloste selaimeen PDF-muodossa painiketta napsauttamalla.

Painike, jolla päätöstuloste tilataan, sijoitetaan käyttöliittymälle osaksi hakemuksen esikatselunäyttöä. Esikatselunäytöllä on tiedossa kyseisen hakemuksen perustiedot. Päätöstulosteen tilaus -painikkeelle lisätään napsautuksesta laukeava tapahtuma, jolloin päästään koodiin käsittelemään käyttäjän päätöstulosteen tilauspyyntöä.

Koodissa on tehty oma Report.aspx-sivu päätöstulostetta varten. Aspx-sivu on osa ASP.NET-frameworkia, joka taas on osa .NET-frameworkia. Aspx-sivu toimii kuin HTML-sivu, mutta lisäksi siihen voi sijoittaa koodia jota suoritetaan, luoden siitä dynaamisen. Aspx-sivut muodostuvat kahdesta osasta: aspx-päätteisestä tiedostosta, jossa määritetään sivun ulkoasu, sekä aspx.cs-päätteisestä tiedostosta (mikäli ohjelmointikielenä on C#), jota kutsutaan myös code behind -tiedostoksi, jossa itse koodi sijaitsee. [8.] Report.aspx-sivulla käytetään NET-frameworkin ReportViewer-luokkaa hyväksi. ReportViewer-luokka on tehty raporttien näyttämistä ja kutsumista varten. Siitä luodulla ilmentymällä voidaan kutsua Report Serverille ladattua RDL-tiedostoa. ReportViewerille pitää alustaa Report Serverin osoite, kutsuttavan RDL-tiedoston polku sekä voimassa olevat Report Serverin kirjautumistiedot, joilla saadaan yhteys Report Serveriin muodostettua. Nämä tiedot ovat sen tyyppisiä, että ne halutaan pitää jatkossa helposti muokattavassa paikassa, mikäli esimerkiksi Report Server siirretään eri palvelimelle ja sen osoite muuttuu. Tiedot tallennetaan web.config-tiedostoon, josta ne haetaan koodissa. Web.config-tiedoston arvoja pystyy muokkaamaan suorituksen aikana ilman, että koko projektista tarvitsee viedä uutta versiota palvelimelle.

Report.aspx.cs-luokassa on rakennettu logiikka niin, että sivun URL-osoitteen yhteydessä voidaan ottaa vastaan kaksi parametria. Näiden parametrien avulla saadaan tieto hakemuksen tyypistä sekä hakemuksen uniikista tunnisteesta. Hakemuksen esikatselunäytöllä napsautetaan tilaa päätöstuloste -painiketta, jolloin napsautuksen aiheuttamassa tapahtumassa saadaan selville hakemuksen tyyppi sekä

hakemuksen uniikki tunniste, koska hakemuksen esikatselunäytöllä on tiedossa kaikki hakemuksen perustiedot. Report.aspx-sivua kutsutaan esikatselunäytön code behind -tiedostossa rakentamalla Report.aspx-sivun URL-osoite. Osaksi URL-osoitetta lisätään parametrit hakemuksen tyyppi sekä hakemuksen uniikki tunniste. Esimerkki kasaan parsitun URL-osoitteen loppuosasta on muotoa ”…/Report.aspx?CaseKey=LOMA223432&ReportType=vacations”. Aspx-sivuille voidaan lähettää parametreja URL-osoitteen yhteydessä lisäämällä ne kysymysmerkin perään. Esimerkin URL-osoitteessa CaseKey on parametrin nimi ja yhtäsuuruus-merkin jälkeinen arvo on parametrin sisältämä arvo eli ”LOMA223432”. Parametri kertoo hakemuksen uniikin tunnisteen. ReportType-parametri toimii samalla tavalla, mutta kertoo hakemuksen tyypin, joka tässä tapauksessa on ”vacations.” eli lomahakemus. Koodissa ohjataan käyttäjä esimerkki-URL-osoitteen osoittamalle sivulle, jolloin päästään kiinni Report.aspx.cs-sivun PageLoad-tapahtumaan. Tapahtumassa luodaan ilmentymä ReportViewer-luokasta ja alustetaan se kuntoon aikaisemmin mainitulla tavalla. Kun ReportViewer-olio on kunnossa, voidaan sille asettaa RDL-tiedoston tarvitsemat parametrit, jotka saadaan selvitettyä URL-osoitteen mukana lähetetyistä parametreista. Lisäksi asetetaan Language-parametrin arvoksi tällä hetkellä suoraan FI. Jos tulevaisuudessa tulee tarvetta, voidaan rakentaa logiikka päätöstulosteiden kielen valitsemiseksi. Myös päätöstulosteen otsikko selvitetään tässä kohtaa tulostetyypin perusteella ja se asetetaan paikoilleen.

ReportViewer-oliolla on metodi, jolla voidaan kutsua Report Serveriltä haluttua RDL-tiedostoa. Kun sitä kutsutaan, menevät parametrit kutsun mukana paikoilleen ja RDL-tiedosto kutsuu taas edelleen datalähteensä ja datasetinsä avulla WCF-palvelua samoilla parametreilla. WCF-palvelu suorittaa oman koodinsa ja palauttaa RDL-tiedostolle listan nimi-arvo-pareja, jotka asettuvat RDL-tiedostolla omille määritetyille paikoilleen, eli kaksisarakkeiseen taulukkoon. Päätöstulosteen otsikon arvoksi sijoitetaan suoraan parametrina Report.aspx.cs-sivulta saatu päätöstulosteen otsikko-parametri. Tämä juuri muodostettu päätöstuloste palautetaan Report.aspx.cs-sivulle, jossa se otetaan vastaan binäärimuodossa. Binäärimuotoinen data sijoitetaan Report.aspx-sivun HTTPResponseen ja kerrotaan sille, että se halutaan lähettää selaimeen PDF-muodossa. Käyttäjä ei missään vaiheessa siirry pois hakemuksen esikatselu-sivulta, vaan Report.aspx-sivu pyörii taustalla tehden oman osuutensa kokonaisuudesta palauttaen käyttäjän selaimeen pelkästään päätöstulosteen PDF-muodossa.

Nyt voidaan palata kuvaan 9 ja todeta, että siinä kuvattu reitti on käyty läpi ja ollaan tilanteessa, jossa käyttäjän selaimeen tarjotaan hakemuksen päätöstulostetta PDF-muodossa. Käyttäjä pysyy samalla sivulla kuin päätöstulosteen tilauspainiketta napsauttaessaan oli ja selaimeen aukeaa kuvan 13 mukainen kyselyikkuna, jossa käyttäjältä kysytään, haluaako hän peruuttaa toiminnon, avata tiedoston vai tallentaa tiedoston.

Kuva 13. Vuosilomahakemuksen päätöstulostetta tarjotaan käyttäjälle PDF-muodossa.

5 Yhteenveto

Insinöörityön tavoitteena oli toteuttaa päätöstuloste osaksi HR-järjestelmää. Työssä käytettävät työkalut ja raamit olivat ennalta annettuja, joten niihin ei voitu vaikuttaa. Sen sijaan osat piti sijoittaa paikoilleen ja saada kommunikoimaan keskenään niin, että saatiin toimiva väylä aikaiseksi päätöstulosteen tuomiseksi käyttäjälle selaimeen PDF-muodossa oikealla tietosisällöllä ja ulkoasulla.

Työn tavoitteeksi oli asetettu vain yhden päätöstulosteen muodostaminen, koska se kuitenkin pakotti luomaan toimivan väylän jatkossa tuleville päätöstulosteille, jolloin niitä olisi helppo lisätä osaksi projektia. Toteutus suunniteltiin silmälläpitäen jatkossa tulevia uusia päätöstulosteita ja siinä tehdyt ratkaisut tehtiin aina varautuen päätöstulosteiden tulevaan suureen määrään. Väylä saatiin toimivaksi ja vuosilomahakemuksen päätöstuloste selaimelle PDF-muodossa. Vuosilomahakemuksen päätöstulosteelle saatiin poimittua oikea tietosisältö tietokannasta, ja se vastasi määrittelyjä ulkoasun ja sisällön puolesta.

Ratkaisua tehtäessä tuli kuitenkin pyyntö toteuttaa lopuillekin noin 40 hakemukselle omat päätöstulosteet. Päätöstulosteen käyttämä RDL-tiedosto oli suunniteltu niin, että siitä tuli mahdollisimman yksinkertainen ja se vain sijoittaa vastaanottamansa tietojoukon taulukkoonsa, tietojoukon koosta riippumatta. WCF-palvelu, joka tuo tietosisällön RDL-tiedostolle, oli toteutettu niin, että siihen on helppo lisätä uusien yksittäisten tietojoukkojen luominen hakemuksesta riippuen. WCF-palvelun koodia laajennettiin, ja siihen lisättiin käsittely pyydetyille hakemuksille. RDL-tiedostoon, eikä mihinkään muualle tarvinnut koskea ollenkaan, vaan WCF-palvelun muokkaamisen ja hyvin suunnitellun väylän ansiosta saatiin noin 40 hakemukselle muodostettua päätöstulosteet.

Kaikki päätöstulosteet menivät asiakkaalle testiin, jonka seurauksena päätöstulosteille haluttiin asioita eri tavalla näkyviin ja määrittelyihin tuli muutoksia. Tämän seurauksena päätöstulosteita ruvettiin muokkaamaan asiakkaan tarpeita vastaaviksi ja ne vietiin uudelleen testattaviksi. Muutoksia tullaan tekemään niin pitkään, kunnes asiakas on varma haluamastaan lopputuloksesta ja päätöstulosteet vastaavat sitä.

Lähteet

1 SQL Server Reporting Services. 2013. Verkkodokumentti. Microsoft. http://msdn.microsoft.com/fi-fi/library/ms159106(v=sql.105).aspx. Luettu 10.2.2013.

2 Report Definition Language Reference. 2013. Verkkodokumentti. Microsoft. http://msdn.microsoft.com/en-us/library/ms155062(v=sql.105).aspx. Luettu 10.2.2013.

3 Löwy, Juval. 2010. Programming WCF Services.

4 ASP.NET Master Pages. 2013. Verkkodokumentti. Microsoft. http://msdn.microsoft.com/en-us/library/wtxbf3hh(v=vs.100).aspx. Luettu 20.3.2013.

5 Custom Report Template in SSRS. 2012. Verkkodokumentti. Bhushan Shah. http://bhushan.extreme-advice.com/custom-report-template-in-ssrs/. Luettu 27.3.2013.

6 How to: Add a Subreport and Parameters (Report Builder 3.0 and SSRS). 2013. Verkkodokumentti. Microsoft. http://msdn.microsoft.com/en-us/library/dd220581(v=sql.105).aspx. Luettu 27.3.2013.

7 Report Parts in Report Designer (SSRS). 2013. Verkkodokumentti. Microsoft. http://msdn.microsoft.com/en-us/library/ee635721(v=sql.105).aspx. Luettu 27.3.2013.

8 ASP.NET Web Pages Overview. 2013. Verkkodokumentti. Microsoft. http://msdn.microsoft.com/en-us/library/428509ah(v=vs.100).aspx. Luettu 10.4.2013.