TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JARKKO TUIKKA

AJASTETTUJA TEHTÄVIÄ SUORITTAVAN JÄRJESTELMÄ-YLLÄPIDOLLISEN TYÖKALUN MÄÄRITTELY JA TOTEUTUS

Diplomityö

Tarkastaja: Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-neuvoston kokouksessa 9. joulu-kuuta 2015

TIIVISTELMÄ

JARKKO TUIKKA: Ajastettuja tehtäviä suorittavan järjestelmäylläpidollisen työkalun määrittely ja toteutus
Tampereen teknillinen yliopisto
Diplomityö, 45 sivua
Tammikuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Sulautetut järjestelmät
Tarkastaja: professori Tommi Mikkonen

Avainsanat: .NET, Windows Workflow Foundation, AppFabric, ajastetut tehtävät, järjestelmäylläpito

Asiakkalla oli tuotantojärjestelmä, johon oli asiakkaan toimesta lisätty skriptejä, ohjelmia ja palveluita, jotka hoitivat suhteellisen yksinkertaisia tehtäviä ajastetusti. Ongelmana tässä järjestelmässä oli se, että tehtäviä oli toteutettu erilaisin tekniikoin ja ajastuksia oli lisätty Windows Task Scheduler:lle usealle koneelle. Tästä syystä järjestelmä oli sekava ja vaati hallintatyökalua. Ongelmia analysoitaessa päätettiin, että toteutetaan uusi järjestelmä, jossa kaikki hoidetaan keskitetysti, hyläten vanhat toteutukset.

Ongelmien analysoinnista päästiin uuden järjestelmän määrittelyyn. Järjestelmän rajoitukseksi asetettiin, että se toimisi Microsoft Windows- ympäristössä .NET- sovelluskehyksellä. Määrittelyssä päätettiin, että järjestelmä sisältää yhden käyttöliittymän jossa tehtäviä muokataan graafisesti Windows Workflow Foundation (WF)- tekniikalla ja näiden ajastukset määritetään samasta käyttöliittymästä. Arkkitehtuuria määritellessä päätettiin, että käyttöliittymässä luodut tehtävät tallennettaisiin WF- palveluina Windows AppFabric:iin, joka on Internet Information Services (IIS)- palvelinohjelmiston lisäys. Näitä palveluita voitaisiin sitten käynnistää mistä tahansa webservice- kutsuina. Käyttöliittymän lisäksi järjestelmään tarvittaisiin myös taustapalvelu, jota käyttöliittymä voisi käyttää webservice-palvelun kautta. Tämä taustapalvelu ylläpitäisi ajastuksia, ja pystyisi käynnistämään halutun WF- palvelun ajastuksen lauetessa, mutta myös tallentaisi tiedot pysyvästi tietokantaan. Käyttöliittymä, taustapalvelu ja tehtävät muodostaisivat lokia toiminnastaan, jota voitaisiin käyttää monitorointiin.

Määrittelyn jälkeen siirryttiin toteutukseen. Toteutuksessa käytettiin sellaisia tekniikoita ja työkaluja kuin WF, Windows Communication Foundation (WCF), Windows Presentation Foundation (WPF), IIS, AppFabric, Web Deploy, Quartz.NET, Common.Logging, Log4net ja Topshelf. Käyttöliittymän toteutuksen pohjana käytettiin Microsoft:n referenssitoteutusta WF- editorikäyttöliittymään, joka sijoitti luodut WF-palvelut ajettavaksi AppFabric:iin. Taustapalvelu toteutettiin Windows-palveluna, joka käytti Quartz.NET-ohjelmakirjastoa vuoronnuksien toteutukseen persistentillä tavalla. Sekä käyttöliittymä että taustapalvelu lokittivat viestejä tietokantaan käyttäen Common.Logging- lokiabstraktiota ja Log4net lokitoteutusta.

Lopputuloksena saatiin määritelmän mukainen ohjelma, joka saatiin onnistuneesti vietyä tuotantokäyttöön asiakkaalle. Määrittelyssä oli esitetty lisätoimintoina ohjelman osien välisten kommunikointien salaus, käyttäjien todennus, luokittelu ja tilastointi käyttöliittymässä sekä suoritettujen tehtävien historian ja yksittäisen suorituskerran graafinen esitys käyttöliittymässä, jotka jätettiin jatkokehitysideoiksi.

ABSTRACT

JARKKO TUIKKA: Specification and implementation of a scheduled task running system-maintenance tool
Tampere University of Technology
Master of Science Thesis, 45 pages
January 2016
Master's Degree Programme in Information Technology
Major: Embedded systems
Examiner: Professor Tommi Mikkonen

Keywords: .NET, Windows Workflow Foundation, AppFabric, Scheduled tasks, System maintenance

One of our clients had a production system, where they had implemented scripts, programs and services to handle scheduled tasks. The problem in their approach was that they had used different techniques to implement the tasks and the schedules were saved to Windows Task Scheduler's on different computer. Thus the system was incoherent and hard to maintain and need for a maintenance tool was clear. After we analyzed the problems, we decided that we should discard the old methods and create a new system, where everything was done in a coherent way.

After analyzing the problems, we started the specification of the new system. The limitations for the new system was that it had to work in Microsoft Windows environment using .NET software framework. We specified that the new system would contain one graphical user interface (GUI), where the users could define the tasks using Windows Workflow Foundation (WF) and create the schedules for running these tasks. In architectural design, we specified that the workflows created in GUI would be deployed into Internet Information Services (IIS) add-on Windows AppFabric as WF- services. The tasks could then be started using web service- calls to these services. In addition to GUI and AppFabric, the system would also need a background service which the GUI could use using a web service. This background service would maintain the task metadata and the task schedules, and could start the tasks according to the schedule. GUI, background service and the tasks should write log about their actions for monitoring purposes.

After specification we moved on to implementation. In the implementation phase we used techniques such as WF, Windows Communication Foundation (WCF), Windows Presentation Foundation (WPF), IIS, AppFabric, Web Deploy, Quartz.NET, Common.Logging, Log4net and Topshelf. A Microsoft's reference implementation for WF editing in WPF application was used for a basis for the GUI. Background service was implemented as a standalone Windows Service, which used Quartz.NET toolkit to schedule the tasks in a persistent way. Both GUI and background service logged messages using Common.Logging- logging abstraction and Log4net logging implementation.

As a result, we created a working program, which was successfully deployed to the customer's production system. Thus the project was successful. As further development ideas, we should implement secured data transfer methods between the modules, user authentication, authorization and statistics in GUI, and showing execution history of the tasks and graphical presentation of a specific task execution in the GUI.

ALKUSANAT

Tämä on Tampereen teknillisen yliopiston tietotekniikan laitokselle tehty diplomityö. Haluan kiittää tämän työn mahdollistamisesta ja valmistumisesta työn tarkastajaa prof. Tommi Mikkosta, Eatech Oy:tä sekä avovaimoani Annea.

Tampereella, 16.12.2015

Jarkko Tuikka

SISÄLLYSLUETTELO

1. JOHDANTO

Erään asiakkaamme tuotantojärjestelmässä oli suorituksessa useita ajastettuja tehtäviä, jotka hoitivat esimerkiksi tietynlaisten aineistojen muodostuksia ja siirtoja muihin palveluihin. Tehtävät olivat kertaluonteisia, eli ne toteuttivat jonkin, yleensä yksinkertaisen, toiminnon ja sen jälkeen niiden suoritus loppuu. Tehtävien ajastuksia oli monenlaisia. Yleisin ajastus-tyyppi oli tehtävän kerran päivässä johonkin kellonaikaan käynnistävä ajastus. Muunlaisia ajastustyyppejä olivat esimerkiksi tehtävän tunnin välein toimistoaikoina tai vaikkapa kerran kuun ensimmäisenä päivänä käynnistävä ajastus. Tehtävä ja ajastus voidaan tässä asiayhteydessä määritellä seuraavalla tavalla:

- Tehtävä: Joukko ohjelmallisia askeleita joilla suoritetaan jokin kokonaisuus. Tehtävä suorituksen kesto on äärellinen ja yleensä lyhyt. Tehtävät eivät ole yleensä monimutkaisia tai kooltaan eli ohjelmakoodin määrältään suuria.
- Ajastus: Määrittää ajankohdan tai -kohdat milloin tehtävän suoritus käynnistetään. Ajastus voi määrittää tehtävän suoritettavaksi heti tai joskus tulevaisuudessa, kerran tai toisteisesti. Ajastukset ovat yleensä suhteellisen yksinkertaisia, esimerkiksi käynnistäen tehtävän kerran päivässä johonkin aikaan, mutta monenlaisia variaatioita on olemassa

Näitä tehtäviä ja ajastuksia varten ei ollut olemassa minkäänlaista hallintatyökalua, joten järjestelmän ylläpitäjät olivat saaneet luoda jokaisen tehtävän, sekä niiden ajastukset, helpoimmaksi näkemällään tavalla, jolloin tehtävien määrän kasvettua alkoi muodostua ongelmia niiden dokumentaatiosta ja ylläpidettävyydestä. Tässä diplomityössä esitellään nuo ongelmat ratkaisevan järjestelmän määrittely, toteutus ja arviointi. Tällä uudella järjestelmällä voidaan luoda, tallentaa, katselmoida ja muokata monimutkaisiakin, mutta kertaluonteisia tehtäviä, ja suorittaa niitä ajastetusti ja vikasietoisesti. Järjestelmä toteutettiin Microsoft Windows -ympäristöön [1] .NET -sovelluskehyksellä [2] ja C# -ohjelmointikielellä [3]. Lukijan oletetaan tuntevan perusteet Windows -ympäristössä ohjelmoinnista, .NET -sovelluskehyksestä ja C# -ohjelmointikielestä.

Työn aluksi luvussa 2 esitellään luotavassa järjestelmässä käytettävät tekniikat ja työkalut. Tekniikat ja työkalut esitellään sillä tasolla, että lukijalla on peruskäsitys niiden käyttötavoista. Tämän jälkeen luvussa 3 esitellään motivaatio tämän järjestelmän luomiseen ja luvussa 4 järjestelmän määrittely. Määrittelyssä esitellään järjestelmälle asetetut vaatimukset sekä rajoitukset ja arkkitehtuuri. Luvussa 5 esitetään järjestelmän toteutus, käyttäen apuna luokkakaavioita, koodiesimerkkejä sekä havainnollistavia kuvia. Luvussa 6 arvioidaan järjestelmän määrittelyn ja toteutuksen onnistumista, ongelmakohtia sekä esitellään jatkokehitysideoita. Luvussa 7 vedetään yhteenveto esitellyistä asioista.

2. TEKNIIKAT

Tässä luvussa esitellään ne tämän sovelluksen tekemiseen käytetyt tekniikat tai työkalut, jotka ovat lukijoille luultavasti ennestään tuntemattomia, tai niiden esittely on nähty muuten tarpeelliseksi. Tämän luvun ei ole tarkoitus toimia kattavana manuaalina esiteltäviin tekniikoihin, vaan paremminkin johdantona näihin tekniikoihin, siten että lukijoilla olisi jonkinlainen käsitys näiden tekniikoiden peruskäytöstä. Esiteltävät tekniikat ovat Windows Workflow Foundation (WF) [4], Internet Information Services (IIS) [5], AppFabric [6], Web Deploy [7], Quartz.NET [8], Common.Logging [9], log4net [10] ja Topshelf [11]. Tässä luvussa ei esitellä niitä tässä työssä käytettyjä tekniikoita, jotka oletetaan ainakin jollain tasolla tunnetuksi tai niiden esittely on katsottu aiheen kannalta tarpeettomaksi. Tällaisia tekniikoita ovat .NET Framework -ohjelmistokehys, C# -ohjelmointikieli, Common Language Runtime (CLR) -ajoympäristö [12], Windows Communication Foundation (WCF) [13], Windows Presentation Foundation (WPF) [14], Windows Server [15] sekä Microsoft SQL Server [16].

2.1 Windows Workflow Foundation

WF on teknologia, joka tarjoaa ohjelmointirajapinnan työnkulkujen muodostamiseen, käyttöliittymä-kontrollin työnkulkujen graafiseen editointiin, sekä infrastruktuurin työnkulkujen suorituksessa syntyvän datan tallennukseen. WF mahdollistaa niin yksinkertaisten prosessien kuin vaativampien palveluidenkin luomisen, mutta ei ole ollut yhtä suosittu tekniikka kuin esimerkiksi imperatiivinen C# -ohjelmointi. Uusin versio WF:sta julkaistiin 15.8.2012 .NET 4.5: n mukana johon tässä luvussa keskitytään. Vanhemmat versiot sisältävät vähemmän ominaisuuksia. WF:lla tehdyt ohjelmat ovat *aktiviteettien* (engl. *Activity*) rakenteisia kokoelmia, jotka mallintavat suoritettavaa prosessia. Kaikki varsinainen toiminnallisuus suoritetaan aktiviteeteissa, ja työnkulku esittää näiden aktiviteettien väliset suhteet. Työnkulussa voi lisäksi määritellä muuttujia, joita voi esimerkiksi asettaa parametreiksi aktiviteeteille. Seuraavissa kappaleisssa esitellään työnkulkujen tyypit, aktiviteetin ja pysyvyyden käsitteet sekä työnkulkujen seuranta-, määrittely- ja suoritustavat. [4]

Työnkulku on tyypiltään joko sekventiaalinen tai tilakonemainen. Sekventiaalinen työnkulku esittää aktivitettien väliset suhteet tiukkana hierarkiana, jossa työnkulku suoritetaan ylhäältä alas. Tällaisessa työnkulussa voi kuitenkin olla esimerkiksi ehtolauseita, silmukoita ja rinnakkaisia suorituksia. Tilakone-työnkulussa aktiviteettien väliset suhteet on määritetty ja sillä on alku mutta ei välttämättä loppua. Tilakoneen suoritus etenee tilasta toiseen näiden välisten tilasiirtymien ehtojen perusteella. Tilakoneita ja sekvenssejä voi kuitenkin yhdistellä esimerkiksi siten, että tilakoneessa jokin tila suorittaa tietyn sekvenssin. Tässä työssä tilakoneita ei käytetä. [4]

Aktiviteetit ovat työnkulkujen rakennuspalasia, jotka sisältävät mahdollisesti parametreja ja paluuarvoja. .NET -sovelluskehys sisältää runsaan joukon valmiita aktiviteetteja, mutta aktiviteetteja voi luoda myös itse, joko yhdistelemällä olemassa olevia, tai luomalla täysin uuden. Valmiit aktiviteetit sisältävät toimintoja työnkulun kontrollointiin, virheiden ja muuttujien käsittelyihin sekä muihin normaaleihin ohjelmointikielten ominaisuuksiin. Käytännössä nämä valmiit aktiviteetit mahdollistavat melkein minkälaisen vain ohjelman tekemisen. Itse tehtävillä aktiviteeteilla saadaan aikaan kaikenlainen sovellukseen liittyvä toiminta. Hyvä esimerkki tällaisesta olisi vaikkapa sähköpostia lähettävä aktiviteetti. Itse tehtävää aktiviteettia varten täytyy luoda aktiviteetti-luokka, joka periytetään WF:n aktiviteettien abstraktista kantaluokasta. Lisäksi aktiviteetille täytyy luoda näkymä, jotta sitä voidaan käyttää graafisessa *workflow designer* -käyttöliittymässä. Näkymiä muokataan WPF -tekniikkassakin käytettävällä Extensible Application Markup Language (XAML) -tekniikalla [17]. Itse tehdyt aktiviteetit käännetään Dynamic Link Library (DLL) -ohjelmakirjastoiksi [18], jolloin niitä voidaan käyttää workflow designer -näkymässä kuten valmiitakin aktiviteetteja. [4]

Pysyvyys (engl. *Persistence*) on WF:n lisäominaisuus, jolla voidaan tallentaa työnkulkujen tila pysyväistalteen ja jatkaa siitä tilasta myöhemmin. WF tukee valmiiksi SQL Server tietokannan käyttöä tallennuspaikkana, mutta jos tarvetta toisten tallennuspaikkojen käyttämiselle on, niin näitä varten voidaan luoda oma lisämoduuli. Pysyvyyttä voidaan käyttää kahdella tapaa: työnkulussa voidaan määritellä *Persistent* -aktiviteetti, joka pakottaa tilan tallennuksen tietokantaan, tai sitten työnkulkua ajava isäntäohjelma voi tallentaa työnkulun automaattisesti, jolloin työnkulku voidaan poistaa muistista resurssien säästämiseksi. [4]

Työnkulkujen seurannalla (engl. *Tracking*) mahdollistetaan työnkulkujen tilan monitorointi. Työnkulut ja aktiviteetit automaattisesti luovat viestejä niiden käynnistyksestä, lopetuksesta ja niissä tapahtuneista virheistä, mutta lisäksi itse tehdyissä aktiviteeteissa voidaan luoda omia viestejä *julkaisija-tilaaja* -tyyppisellä viestien lähetyksellä. Aktiviteetit julkaisevat *TrackingRecord* -viestejä, joita voidaan suodattaa *TrackingProfile* -profiileilla ja lähettää eteenpäin *TrackingParticipant* -moduuleilla, jotka ovat viestien tilaajia. Tilaajat ovat lisättävissä työnkulkuun konfiguraatiotiedostossa. [19]

Työnkulkuja määritellään joko XAML-tiedostoina, tai jollain Common Language Runtime (CLR) kielellä, kuten C#:lla. Käytettäessä workflow designer -käyttöliittymää työnkulun määrittelyyn, tuloksena tulee XAML-tiedosto. Workflow designeria voidaan käyttää suoraan Visual Studiossa, tai se voidaan liittää osaksi itse tehtävää WPF -käyttöliittymäohjelmaa. Työnkulkuja voidaan määritellä myös WF -palveluiksi, jotka ovat WCF -palveluita, joiden toiminta on määritetty työnkulkuina. Nämä mahdollistavat työnkulun ajamisen palveluna, joka voidaan käynnistää metodikutsulla, ja ne voivat palauttaa tiedon siitä, miten prosessi onnistui. WF -palvelut eroavat normaaleista työnkuluista siinä, että ne on määritetty XAMLX-tiedostossa XAML-tiedoston sijaan. XAMLX-tiedostot ovat WF -palveluiden määrittelytiedostoja. Valmis WF-ohjelma on joko XAML- tai XAMLX-

tiedosto, jossa määritellään työnkulku, mutta lisäksi se voi tarvita joitain DLL -kirjastoja, jos tällaisessa on esimerkiksi määritelty itse tehtyjä aktiviteetteja, joita käytetään työnkulussa. Varsinaisten ohjelman määrittelyjen lisäksi näillä voi olla konfiguraatiotiedosto, jossa voidaan määritellä esimerkiksi työnkulun pysyvyys, seuranta ja WF-palvelun tiedot. [4]

Työnkulkuja suoritetaan sellaisenaan omassa applikaatiossa, jos ne on luotu koodissa tai määritelty .xaml- tiedostossa, käyttäen *WorkflowInvoker* -luokkaa. WF -palveluita voidaan isännöidä itse tehdyssä palvelussa, IIS:ssä tai IIS:ssä johon on lisätty Windows AppFabric. Näistä vaihtoehdoista viimeinen on Microsoft:n suosittelema tapa, sillä se mahdollistaa työnkulkujen hallinnoinnin ja monitoroinnin työkaluja. [20]

2.2 Internet Information Services

IIS on Microsoftin verkkopalvelinohjelmisto, joka on mahdollista asentaa suurimpaan osaan Windows NT -käyttöjärjestelmiä ja kuuluu valmiiksi asennettuna kaikkiin Windows Server -käyttöjärjestelmiin. IIS:llä voidaan isännöidä Hypertext Transfer Protocol (HTTP) [21], File Transfer Protocol (FTP) [22], Simple Mail Transfer Protocol (SMTP) [23] ja Network News Transfer Protocol (NNTP) [24] -palveluita. Sen ensimmäinen versio, IIS 1.0, julkaistiin Windows NT 3.51 käyttöjärjestelmään lisättävänä ohjelmistona, ja uusin versio on Windows 10- sekä Windows Server 2016 -käyttöjärjestelmissä käytettävä IIS 10. IIS:n ja siihen liittyvien ohjelmistojen asennuksen helpottamiseen voi käyttää Microsoft Web Platform Installer:ia (WepPI) [25]. [5]

Versiosta 7.0 eteenpäin ovat IIS-toiminnot olleet modulaarisia siten, että useat IIS:n toiminnot hoidetaan siihen lisättävillä ohjelmakirjastoilla. Näillä ohjelmakirjastoilla hoidetaan verkkopalvelimelle tulleiden pyyntöjen autentikointia, sisällön tulkintaa ja muokkausta, vastauksien pakkausta ja lokin keräämistä. Versiosta 7.0 eteenpäin IIS:ssä on myös mahdollista käyttää palveluiden siirtotapana muutakin kuin HTTP-protokollaa, esimerkiksi Transmission Control Procotol (TCP) -protokollaa [26], *nimettyjä putkia* (engl. *named pipes*) [27] ja Microsoft Message Queueing -viestijonoja (MSMQ) [28]. [5]

2.3 AppFabric

AppFabric laajentaa IIS:n toiminnallisuutta parantamalla erityisesti WF -palveluiden hallinnointia ja monitorointia. AppFabric on asennettavissa WebPI -sovelluksella, ja se tarvitsee tietokannan kaikkien ominaisuuksien toteuttamiseen. AppFabric lisää IIS -käyttöliittymän kautta käytettäviä asetus- ja tilastointityökaluja, mutta myös WF -instanssien ajamisen pysyvyyteen (Persistence) sekä monitorointiin (Monitoring) liittyviä toimintoja, joita on selitetty tarkemmin alla. Nämä ominaisuudet käytännössä automatisoivat WF:n pysyvyys- ja seuranta-ominaisuuksien käytön. [6]

AppFabric Persistence mahdollistaa WF -palveluiden tilan tallennuksen tietokantaan kesken suorituksen. Tämä parantaa palveluiden luotettavuutta sekä hallittavuutta, sillä kesken suorituksen keskeytetty tehtävä voidaan käynnistää paremmalla hetkellä samasta tilasta. Samasta tilasta jatkaminen voi olla tärkeää, jos tehtävä on pitkä tai alussa olleita toimintoja ei haluta toistaa. Tehtävän voi keskeyttää käyttäjä halutessaan, IIS resurssien puutteen takia, tai tehtävä itse jonkin virheen takia.

AppFabric Monitoring mahdollistaa palveluiden tilasta ja WF-instanssien ajoista syntyvien tapahtumien tallentamisen tietokantaan käyttäjän niin halutessa. Jotta tämä olisi mahdollista, täytyy jokaisessa WF-palvelussa olla määritettynä *ETW Tracking Participant* -seurantamoduuli. AppFabric osaa kerätä tämän moduulin lähettämän viestit ja tallentaa ne tietokantaan. IIS:n käyttöliittymästä voidaan tarkastella palveluiden tilatietoja, WF -palveluun tehtyjen kutsujen määriä ja ajossa tapahtuneita virheitä. Tallennetusta datasta voidaan lisäksi muodostaa rekonstruktio ajetun WF -instanssin suorituksesta. Käytännössä tämä mahdollistaa suoritettujen tehtävien ajokertojen historian tutkimisen, mikä on erityisen mielenkiintoista meidän järjestelmämme kannalta.

2.4 Web Deploy

Microsoft Web Deploy on työkalu joka helpottaa web -sovellusten ja palveluiden käyttöönottoa IIS -palvelimille. Web Deploy on asennettavissa WebPI -sovelluksella. IIS -palvelinkoneella täytyy olla asennettuna *Remote Agent Service* tai *Web Management Service* etäkäyttöä varten, joista ensimmäinen on tarkoitettu ylläpitäjien käyttöön, ja jälkimmäinen voidaan konfiguroida käytettäväksi millä tahansa käyttäjätunnuksella. Tarvittavien komponenttien asennuksen jälkeen haluttu web-applikaatio voidaan siirtää käyttöön halutulle IIS -palvelimelle etänä *msdeploy.exe* -ohjelmalla. [7]

Tässä järjestelmässä on tarkoitus käyttää Web Deploy -työkalua automatisoimaan Workflow -palveluiden siirtoa ajettavakasi IIS -palvelimella.

2.5 Quartz.NET

Quartz.NET (jatkossa: Quartz) on .NET -ohjelmakirjasto, jolla hoidetaan tehtävien ajastamista. Se on .NET -sovitus Java -ohjelmointikielelle [29] tehdystä Quartz -ohjelmakirjastosta. Sen käyttäminen ohjelmassa on yksinkertaisimmassa tapauksessa erittäin helppoa ja vaatii vähän ohjelmakoodia ja konfiguroimista, mutta samalla se kuitenkin sisältää hyvän määrän mukautettavia ominaisuuksia. Seuraavissa esitellään Quartz -vuorontajan perustoiminta, konfigurointi, datan pysyvyys, sekä tehtävän, ajastuksen, kalenterin ja kuuntelijan käsitteet. Quartzissa on joitain ominaisuuksia joita ei tässä esitellä, kuten vuorontaja -instanssien klusterointi, valmiit tehtävät ja kuuntelijat sekä liitännäiset. [8]

Quartz sisältää staattisen *ISchedulerFactory* -luokan, jonka kautta saadaan instanssi *IScheduler* -rajapinnan toteuttavaan vuorontajaan. Tämän rajapinnan kautta Quartz -vuorontaja voidaan käynnistää ja pysäyttää, ja siihen voidaan lisätä tehtävien ajastukseen liittyviä tietoja. Tämä on Quartzin perustoimintaa. Quartzissa ajastuksiin liittyy neljä erillistä objektia: *tehtävät*, *ajastukset*, *kalenterit* ja *kuuntelijat*. Quartzissa tehtävät ja ajastukset ovat erillisiä asioita ja ne lisätään vuorontajalle omina kokonaisuuksinaan. Ajastuksen täytyy kuitenkin viitata johonkin olemassa olevaan tehtävään, kun taas tehtävä voi olla olemassa itsenäisestikin. Ajastukset ja tehtävät identifioidaan niiden nimien perusteella, joiden täytyy olla uniikkeja.

Quartzin konfigurointi tapahtuu joko konfiguraatiotiedostossa tai ohjelmallisesti. Konfiguraatiossa täytyy asettaa ainakin käytettävän vuorontajan instanssin nimi, käytettävien säikeiden maksimäärä, käytettävä tiedon tallennusmuoto, ja lokin keräämisen konfigurointi. Quartz käyttää Common.Logging -rajapintaa lokiviestien julkaisemiseksi. Käytettävien säikeiden maksimimäärä vaikuttaa suoraan siihen montako tehtävää voi olla yhtä aikaa käynnissä. Käytettävä tiedon tallennusmuoto tarkoittaa tässä sitä, että käytetäänkö jotain tietokantaa datan pysyvään tallennukseen.

Quarz voidaan asettaa toimimaan pysyvästi, jolloin kaikki sen tarvitsema data tallennetaan pysyvään talteen tietokantaan. Tällöin vuorontaja voidaan sammuttaa hetkellisesti ilman, että mitään dataa häviää. Quartz tukee useita erilaisia tietokantoja, esimerkiksi SQL Server -tietokantaa. Tarvittavan tietokannan luomiseen on olemassa valmiit skriptit eri tietokantoja varten, ja tietokannan yhteystiedot voidaan asettaa vuorontajan konfiguraatiossa, joten pysyvyys -ominaisuuden käyttöönotto on vaivatonta. Jos pysyvälle tietojen tallennukselle ei ole tarvetta, niin Quartz voi pitää tietoja tallessa myös yksinkertaisesti muistissa.

Tehtävä on Quartzissa mikä tahansa luokka, joka toteuttaa *IJob* -rajapinnan. Tämä rajapinta sisältää ainoastaan *Execute()* -metodin jossa tehtävä hoitaa halutun toiminnallisuuden. Tehtäville voidaan myös lisätä parametreja, jotka Quartz -vuorontaja osaa antaa parametreina tehtävälle sen suorituksen ajaksi. Tehtävä on asetettavissa sellaiseksi, että siitä voi olla ainoastaan yksi instanssi ajossa kerrallaan.

Ajastukset ovat vuorontajaan tallennettavia objekteja, jotka sisältävät tiedon käynnistettävästä tehtävästä, sekä aikataulun ja kalenterin. Kalentereista lisää alla. Aikatauluja on muutamia erilaisia, ja niillä pystyy hoitamaan käytännössä mitkä tahansa ajastustarpeet. Aikatauluista mainittakoon mahdollisuus *cron* -lausekkeiden käyttöön. Cron -lausekkeet ovat käteviä varsinkin kalenteriin perustuvien ajastusten määrittämiseen ja ovat peräisin UNIX -käyttöjärjestelmien [30] ajastuspalveluista.

Kalenterit ovat Quartzissa ominaisuus, jolla voidaan poistaa aikatauluista aikoja, jolloin ajastus ei ole käytössä. Tämä käytännössä mahdollistaa minkälaisen vain aikataulun muodostamisen. Kalenterit lisätään vuorontajaan ajastuksista erillisinä kokonaisuuksina, ja

ajastus voi sitten viitata tällaiseen jo luotuun kalenteriin. Kalenterit ovat käteviä esimerkiksi poistamaan jokin juhlapyhä pois aikataulusta. Kalentereita voi ketjuttaa, toisin sanoen kalenteri voi olla linkitettynä toiseen kalenteriin. Tämä ominaisuus on olemassa lähinnä siksi, ettei aikataulu voi viitata kuin yhteen kalenteriin, joten monimutkaisempien kalenterien luominen on täytynyt mahdollistaa jollain muulla keinolla.

Quartzissa on olemassa tehtävien sekä ajastuksien kuuntelijoita. Nämä ovat itse tehtäviä luokkia, jotka toteuttavat *IJobListener*- tai *ITriggerListener* -rajapinnan. Näiden tarkoituksena on mahdollistaa ohjelmallisten toimien suorittaminen tiettyjen tehtäviin tai ajastuksiin liittyvien tilanteiden tapahtuessa. Näitä tilanteita ovat tehtävän kuuntelijan tapauksessa tehtävän käynnistys tai sen suorituksen loppuminen, ja ajastuksen kuuntelijan tapauksessa ajastukseen linkitetyn tehtävän käynnistys, tai sen tehtävän suorituksen loppuminen. Kuuntelija voidaan linkittää käytettäväksi yksittäisen tehtävän tai ajastuksen kanssa, jonkin tehtävä- tai ajastusjoukon kanssa tai vaikkapa kaikkien tehtävien tai ajastusten kanssa.

2.6 Common.Logging

Common.Logging on .NET -ohjelmakirjasto, joka toimii lokin keräämisen abstraktiona *julkaisija-tilaaja* -mallilla. Kirjaston tarkoituksena on erottaa lokiviestien julkaiseminen ohjelmakodissa niiden varsinaisesta tallennustavasta, jolloin varsinainen lokitoteutus voidaan valita myöhemmin ja sitä voidaan vaihtaakin tarvittaessa. Common.Logging tukee tällä hetkellä ainakin Log4net- ja NLog -lokitoteutuksia [31], tarjoten näitä varten valmiit sovittimet, mutta lisäksi tarvittavan sovittimen minkä tahansa muun lokitoteutuksen käyttöön voi tehdä itsekin. Käytettävän sovittimen konfigurointi tapahtuu joko määrittelemällä se konfiguraatiotiedostossa tai sitten ohjelmallisesti. [9]

Lokiviestien julkaiseminen Common.Logging -rajapintaa käyttäen onnistuu seuravan esimerkin mukaisesti:

```
ILog logger = LogManager.GetLogger(typeof(MyClass));
logger.Debug("example log message")
```

Esimerkissä otetaan käyttöön *logger* -instanssi staattisen *LogManager* -luokan avulla, jonka jälkeen lokiviestejä voidaan kirjoittaa kyseisellä lokittajalla. Lokittaja on terminä merkityksellinen, sillä joidenkin lokitoteutusten konfiguraatiossa voidaan suodattaa viestejä niiden lokittajan perusteella. Esimerkiksi alla esiteltävä log4net on tällainen.

2.7 Log4net

Apache log4net on .NET -ohjelmakirjasto, jolla hoidetaan lokin keräämistä. Se on sovitus alkuperäisestä Java -ohjelmointikielelle tehdystä Apache log4j -kirjastosta. Log4net on kätevä neljästä eri syystä: Se tukee erilaisia lokitasoja, tukee usean yhtäaikaisen lokiso-

vittimen käyttöä, sisältää useita valmiita lokisovittimia mutta mahdollistaa omienkin lokisovittimien luomisen ja mahdollistaa asetusten määrittämisen monipuolisesti säädettävän konfiguraatiotiedoston avulla. Lokisovittimella tarkoitetaan tässä *lisääjää* (engl. *Appender*), joka on *log4net.Appender.IAppender* -rajapinnan toteuttava luokka ja hoitaa lokiviestin ohjauksen haluttuun kohteeseen. [10]

Valmiit lokisovittimet mahdollistavat esimerkiksi tiedostoon ja tietokantaan kirjoittamisen, joten normaalitapauksessa mitään ylimääräistä ei tarvitse ohjelmoida itse. Konfiguraatiotiedostossa voidaan määrätä käytettävät lokisovittimet ja kussakin lokisovittimessa käytettävät, esimerkiksi lokitasoon perustuvat, viestien suodattamiset ja viestien ulkomuodot. Tällöin ohjelmakoodiin ei välttämättä tarvitse koskea, jos lokiviestien muodostusta halutaankin muuttaa jossain vaiheessa.

2.8 Topshelf

Topshelf on .NET -ohjelmakirjasto, joka helpottaa .NET -ohjelmien asentamista Windows -palveluiksi. Windows -palvelut ovat melkein kuin normaaleja ohjelmia, mutta ne on rekisteröity Windows:n Service Control Manager:lle, jota kautta palveluiden käynnissä oloa voidaan hallinnoida, esimerkiksi automatisoida palvelun käynnistys laitteen käynnistyessä. Topshelf mahdollistaa myös ohjelman ajamisen konsoliapplikaationa, joka helpottaa ohjelmien testausta. Käytännössä Topshelfia käyttäen ohjelma määritetään applikaatioksi, jonka käynnistyksessä ajetaan staattisen *TopShelf.HostFactory* -luokan *Run()* -metodi. *Run()* -metodille annetaan parametrina ainakin käynnistettävä luokka, sekä sen luokan metodit, jotka toteuttavat palvelun käynnistämisen ja pysäyttämisen. Lisäksi voidaan määrittää lisätoimintoja palveluun liittyen, joita ovat esimerkiksi palvelun riippuvuudet muista palveluista sekä virheistä palautumisessa tehtävät toiminnot. Normaalisti näitä ei kuitenkaan tarvitse käyttää ja palvelun nimi, käynnistysmoodi, nimi ja käyttäjätunnus ovat asetettavissa ajonaikaisesti komentoriviparametreilla. Tarvittavan koodin määrä on yleisessä tapauksessa siis minimaalinen. [11]

3. MOTIVOINTI

Tarve tässä diplomityössä esiteltävälle uudelle sovellukselle tuli eräältä asiakkaalta, jolla on tuotantojärjestelmässään suorituksessa useita ajastettuja tehtäviä, joiden hallitsemiseen ei ole kunnollista työkalua. Tätä työkalua, jota ei ole, kutsutaan kuitenkin jatkossa nimellä vanha järjestelmä selkeyden vuoksi. Tässä luvussa esitellään vanha järjestelmä ja sen ongelmat. Ongelmien esittelyn jälkeen esitetään niiden analysointi, ja ratkaisu siitä, että lähdetäänkö vanhaa järjestelmää korjaamaan, vai luodaanko uusi korvaava järjestelmä.

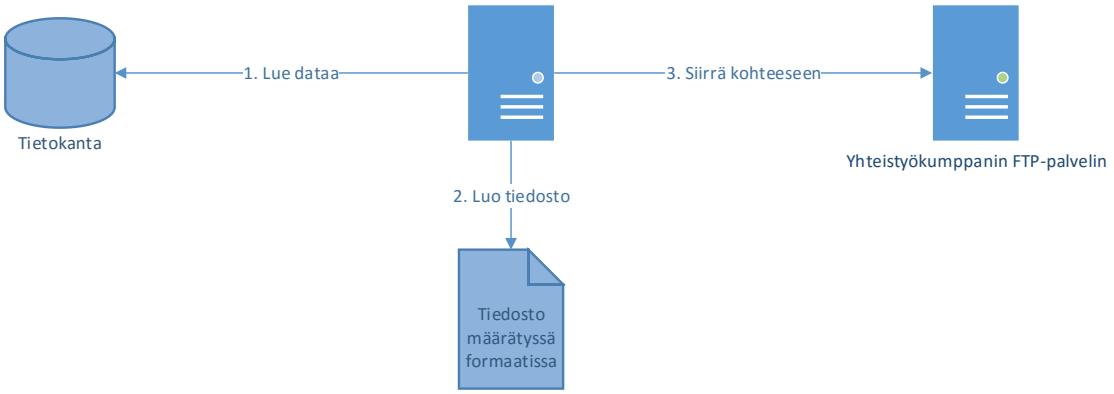
3.1 Vanha järjestelmä

Asiakkaalla on käytössä tuotannon hallinta- ja monitorointijärjestelmä, joka muodostuu useista Windows Server -palvelinkoneista sekä SQL Server -tietokantapalvelimista. Nämä koneet ovat yhteydessä toisiinsa Internet Protocol (IP) -verkossa [32] palomuurilla rajoitettuna. Näistä koneista on liityntöjä kolmannen osapuolen yhteistyökumppaneiden järjestelmiin ja rajapintoihin laskujen, raporttien ja muun datan siirtämistä varten. Tässä työssä ei keskitytä varsinaiseen tuotantojärjestelmään, vaan sen liityntöihin eli integraatioihin näihin kolmannen osapuolen järjestelmiin. Kolmannen osapuolen järjestelmät sisältävät SQL Server -tietokantoja, Windows Server -palvelinkoneita, FTP -palveluita, ja muita sovelluspalveluita. Jotkin harvat yhteistyökumppanien palvelinkoneet ovat Unix-palvelimia. Yhteistyökumppanien koneet toimivat samassa IP -sisäverkossa kuin asiakkaamme palvelimet.

Järjestelmien välillä dataa siirretään useilla eri tavoilla. Näistä mainittakoon suora tiedostonjako, FTP, HTTP, Simple Object Access Protocol (SOAP) [33] sekä tietokantojen käyttäminen niitä lukemalla ja niihin kirjoittamalla. Lisäksi näiden siirtojen yhteydessä saatetaan tehdä jotain oheistoimintoja, esimerkiksi dataa saatetaan ensin muodostaa eri lähteistä, ja siirroista saatetaan muodostaa jonkinlainen hälytys-/monitorointi-viesti tietokantaan, tiedostoon tai sähköpostiin. Siirrot sekä niihin liittyvät muut toimet muodostavat käytännössä yhden koherentin tehtävän, joka suoritetaan ajastetusti esimerkiksi periodisesti kerran päivässä johonkin kellonaikaan, tai vaikkapa kerran kuussa.

Tehtävät on toteutettu osin käynnistettävinä prosesseina eli ohjelmina, osin taustapalveluina, skripteinä ja pieni osa jopa manuaalisesti suoritettavina toimenpiteinä. Skriptien kielinä on käytetty Practical Extraction and Report Language:a (PERL) [34], Restructured Extended Executor:ia (REXX) [35] ja Windows Batch:ia [36]. Skriptien ja prosessien automaattinen ajastus on toteutettu Windows Task Scheduler:n [37] avulla. Taustapalvelut ovat käynnissä aina ja sisältävät jonkinlaisen ajastuksen jonka lauetessa ne suo-

rittavat jonkin toiminnon. Varsinaisten toimintojen lisäksi tehtävät saattavat kirjoittaa toiminnastaan lokia tiedostoon tai tietokantaan tai lähettää sähköpostin ylläpitäjille. Kuvassa 1 on esitetty esimerkkitapaus tehtävästä, joka lukee tietokannasta dataa, muodostaa siitä tiedoston jossain määrätyssä formaatissa ja siirtää sen yhteistyökumppanin FTP -palvelimelle.



Kuva 1. Esimerkki tehtävästä joka luo tiedoston ja siirtää sen kohteeseen.

3.2 Ongelmat

Luotettavuudeltaan vanha järjestelmä on varsin hyvä. Tehtävät käynnistyvät halutulla ajanhetkellä ja suorittavat toimintansa ilman virheitä poikkeuksetta. Myöskään minkäänlaisia tehokkuusongelmia ei ole havaittu. Kaikki ongelmat liittyvät ylläpidettävyyteen, johtuen siitä, ettei tätä vanhaa järjestelmää ole alun perin suunniteltu juuri millään tavalla. Ajan saatossa tehtäviä on vain kertynyt paljon, ja nyt niiden ylläpitäminen alkaa olla mahdotonta. Ylläpidettävyysongelmat voidaan jakaa tässä vanhassa järjestelmässä kolmeen osaan: tehtävien dokumentointi, tehtävien toteutustavat ja tehtävien toteutukseen käytetyt ohjelmointikielet. Lisäksi vanhassa järjestelmässä tehtävien ajastus, monitorointi sekä virheitten tai onnistumisten raportointi on hoidettu hankalasti.

Dokumentoinnillisia ongelmia vanhasta järjestelmästä löytyy kahdenlaisia: Tehtävälistauksen puuttuminen ja yksittäisen tehtävän toiminnan määrittely. Tehtävälistauksen puuttumisen vuoksi järjestelmää tuntemattoman on lähes mahdotonta tietää, minkälaisia erilaisia tehtäviä järjestelmästä löytyy, missä ne sijaitsevat ja minkälainen niiden ajastus on. Tämä johtuu siitä, että tehtävät voivat sijaita fyysisesti erillisillä palvelinkoneilla, ne voivat olla toteutettuna eri tekniikalla (skripti vs. palvelu), tehtävien ohjelmakoodi voi olla tallennettuna minne kohtaa tahansa tiedostojärjestelmää ja niiden ajastus voi olla integroituna tehtävään (palvelu) tai sen hoitaa jokin toinen komponentti, kuten Windows Task Scheduler. Yksittäisistä tehtävistäkään ei ole olemassa hyviä kuvauksia, eivätkä ne itsessään ole hyvin tai millään tavalla kommentoituja. Tästä johtuen yksittäisen tehtävän toimintaa on vaikea analysoida.

Tehtävien toteutustavoilla ei tässä tarkoiteta ohjelmointikieliä, vaan tässä keskitytään periaatteelliseen, ylemmän tason tarkasteluun. Toteutustavat vaihtelevat tehtäväkohtaisesti satunnaisen oloisesti skriptin, ohjelman, palvelun ja tietokanta-proseduurin välillä. Usean eri toteutustavan käyttö ei ole tässä järjestelmässä millään tavalla perusteltua, sillä millä tahansa yhdellä ja samalla tavalla pystyisi kyseiset tehtävät toteuttamaan[1]. Järjestelmän ylläpitäjien olisi siis kannattanut valita jokin yksi tekniikka ja pysyä sen käytössä selkeyden vuoksi. Toteutustavoista erityisesti ohjelmien ja palveluiden käyttö on haitallista, sillä näiden toimintaa ei voi tutkia näkemättä lähdekoodia, mikä ei välttämättä ole saatavilla.

Tehtävien toteutukseen käytettyjä ohjelmointikieliä on järjestelmässä monenlaisia. Rajataan tässä tarkastelu ainoastaan skriptaamalla toteutettuihin tehtäviin, joita niitäkin on kirjoitettu kolmella eri kielellä. Kulloisenkin tehtävän toteutukseen käytetty kieli on valittu satunnaisen oloisesti, luultavasti kyseisen tehtävän luoneen ylläpitäjän oman tottumuksen mukaisesti. Usean eri ohjelmointikielen käyttäminen lisää järjestelmän monimutkaisuutta entisestään kahdesta eri syystä. Ensinnäkin Windows-käyttöjärjestelmälle einatiiveille REXX ja PERL -skripteille täytyy palvelinkoneella olla tallessa asianmukainen ohjelma, joka osaa suorittaa näitä skriptejä. Toisekseen se pakottaa ylläpitäjät käyttämään tarpeettomasti useita eri kieliä tehtävien ylläpidossa, joista osa on vieläpä vanhentuneita tekniikoita, esimerkiksi REXX -skriptit.

Tehtävien ajastukset on suurelta osin hoidettu Windows Task Schedulerilla, joka toimii hyvin. Sitä käyttäen voidaan asettaa monipuolisia ajastuksia ja niitä voidaan muokata näppärästi. Mutta osa tehtävistä, lähinnä ne, jotka on toteutettu palveluina, käyttävät omia ajastuksiaan, joita voi olla vaikea muokata tai sitten niiden muokkaus on kokonaan estetty ohjelmallisesti. Myös Windows Task Schedulerin käyttö muodostuu ongelmaksi kahdella eri tapaa. Ensinnäkin siksi, että järjestelmässä käytetään useita palvelinkoneita; Ei ole koottua näkymää kaikista mahdollisista ajastuksista järjestelmässä. Halutessaan muokata jotain ajastusta käyttäjän tulee ottaa etäyhteys kyseisen tehtävän palvelinkoneelle ja muokata ajastusta siellä. Toinen ongelma Windows Task Schedulerin käytössä on se, että tehtävän toteutus ja ajastus hallinnoidaan täysin eri paikasta, ja käyttäjän tulee vain jollain tavalla tietää missä nämä paikat ovat.

Tehtävien monitorointi on toteutettu tehtäväkohtaisesti jonkin tyyppisellä virheistä ja onnistumisista lokia keräämällä. Yleisin lokin keräämistapa on kirjoittaa lokia tiedostoon, mutta joissain tehtävissä lokia kirjoitetaan tietokantaan. On täysin tehtävän toteutuksesta kiinni, voidaanko sen toimintaa – käynnistymistä, osatehtävien onnistumista ja lopputulosta – monitoroida. Kuten ei tehtävälistauksiin ja ajastuksiin, ei myöskään tehtävien monitorointiin ole mitään yhteistä näkymää mistä tuloksia voisi kätevästi katsella, eivätkä luodut tehtävät implisiittisesti muodosta mitään lokia tai jälkeä niitä suoritettaessa.

[1] Joissain tapauksissa kannattaa hyödyntää erillisiä tietokantaproseduureja tehokkuuden näin vaatiessa, vaikka varsinainen tehtävä olisi toteutettu jollain muulla tekniikalla

3.3 Ongelmien analysointi

Kohdassa 3.2 esitetyt vanhan järjestelmän ongelmat ovat tiivistetysti siis seuraavat:

1. Tehtävien nimien, toiminnan kuvauksien, fyysisien sijaintien ja ajastuksien dokumentointiin ei ole minkäänlaista työkalua
2. Tehtävät on toteutettu monella eri tavalla, joka monimutkaistaa järjestelmää turhaan. Lisäksi osa näistä tavoista (prosessi, palvelu) kätkee tehtävien toiminnan ja tekee tehtävien ylläpitämisestä vaikeaa
3. Tehtävien toteutukseen on käytetty useita eri ohjelmointikieliä, joista osa on vanhentunutta tekniikkaa. Tämä monimutkaistaa järjestelmää turhaan
4. Tehtävien ajastuksiin ei ole yhtenäistä tapaa eivätkä ne ole aina helposti muokattavissa. Suurin osa ajastuksista on hoidettu Windows Task Schedulerilla, joka sekin tuottaa ongelmia, sillä koottua näkymää usean eri koneen ajastuspalveluun ei ole eivätkä tehtävien toteutukset ole suoraan linkitettävissä niiden ajastuksiin.
5. Tehtävien lopputuloksia ja osatehtävien onnistumisia ei automaattisesti monitoroida eikä yhteistä näkymää monitoroinnille ole.

Edellä mainitut vanhan järjestelmän ongelmat tiedostaen täytyy lähteä tekemään päätöstä siitä, halutaanko vanha järjestelmä korjata vai korvata se uudella. Päätöksiin vaikuttavat enimmäkseen ongelmakohtien ratkaisemiseen tarvittavat työmäärät sekä järjestelmän ylläpidettävyys.

Ensimmäinen ongelmakohta, eli dokumentointi-työkalun puute, olisi korjattavissa jollain yksinkertaisellakin sovelluksella johon voitaisiin listata kaikki järjestelmän tehtävät kaikkine tietoineen. Tätä taulukkoa täytyisi kuitenkin pitää yllä manuaalisesti, sen sijaan että tehtäviä lisätessä tai muokatessa ne automaattisesti dokumentoisivat itsensä. Tämän ongelman ratkaisu olisi siis järkevintä rakentamalla uusi järjestelmä.

Toinen ja kolmas ongelmakohta olisivat korjattavissa sillä, että luotaisiin tehtävät uudelleen käyttäen esimerkiksi ainoastaan jotain tiettyä skriptikieltä. Skriptikielet olisivat tässä hyvä lähestymistapa, koska tehtävät ovat pääosin luonteeltaan yksinkertaisia ja niitä täytyy pystyä muuttamaan helposti. Toisaalta jos tehtävät joudutaan joka tapauksessa toteuttamaan uudelleen, niin uuden järjestelemän käyttöönotto ei tässä suhteessa aiheuttaisi ainakaan merkittävästi lisätyötä.

Neljäs ongelmakohta ei ole helposti korjattavissa kokonaan. Vaikka kaikki tehtävät ajastettaisiin vain yhden palvelimen Windows Task Schedulerilla, niin silti tehtävät ja niihin linkitetyt ajastukset eivät olisi suoraan tarkasteltavissa samasta paikasta.

Viides ongelmakohta, eli tehtävien tulosten monitorointi, on vaikeasti korjattavissa kokonaan. Korjausta varten tarvitsisi jokaisen tehtävän toteutukseen tehdä ominaisuus, joka tallentaisi tehtävän askeleitten onnistumisen tai epäonnistumisen johonkin tiedostoon tai

tietokantaan. Jotta tuloksia voisi vielä tarkastella kätevästi, niin tarvitsisi rakentaa jonkinlainen käyttöliittymä tätä varten. Jouduttaisiin siis luomaan täysin uusi monitorointi-sovellus, joten tämänkin ongelman korjaaminen olisi helpointa täysin uudella järjestelmällä.

Yhteenvetona vanhasta järjestelmästä voidaan todeta, että se on liian puutteellinen, jotta sitä kannattaisi lähteä korjaamaan. Samalla työmäärällä voidaan luoda uusi järjestelmä, jossa jo lähtökohtaisesti otetaan huomioon nämä edellä mainitut ongelmat.

4. MÄÄRITTELY

Tässä luvussa esitellään uuden järjestelmän määrittely ja vaatimukset. Vaatimukset on kehitetty korjaamaan luvussa 3 esitellyt vanhan järjestelmän ongelmat. Osa vaatimuksista on pakollisia ja osa lisäominaisuuksia. Lisäominaisuudet voidaan toteuttaa, jos ne nähdään tarpeellisiksi ja/tai aikaa riittää. Määrittely on jaettu neljään osioon: Toiminnalliseen määrittelyyn, ei-toiminnalliseen määrittelyyn, rajoituksiin ja arkkitehtuuriin.

4.1 Toiminnalliset vaatimukset

Tässä kohdassa esitellään järjestelmän toiminnalliset vaatimukset liittyen yleisesti käyttöliittymään, tehtäviin, ajastuksiin sekä käyttäjiin. Nämä vaatimukset esitetään käyttötapauskuvina kootusti alakohdassa 4.1.5.

4.1.1 Käyttöliittymä

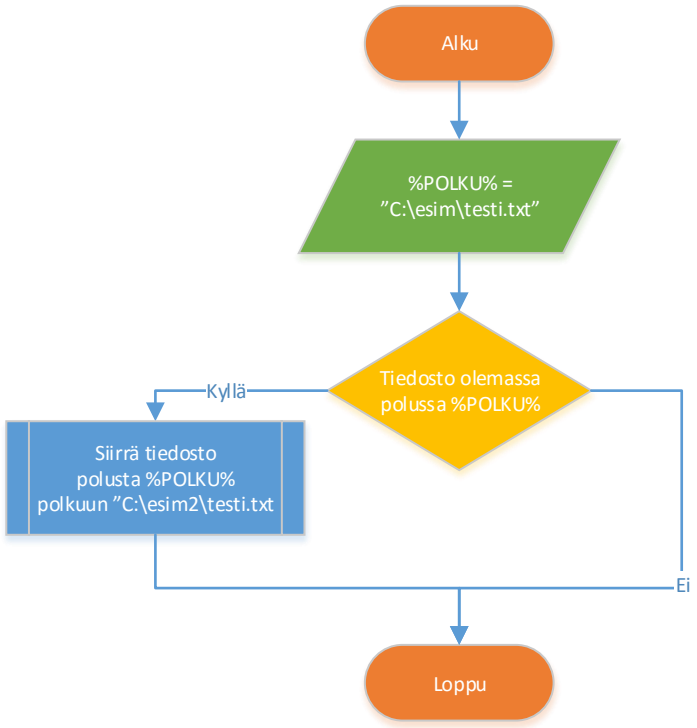
Ensimmäinen ja tärkein vaatimus uudelle järjestelmälle on käyttöliittymä, jossa hoidetaan kaikki käyttäjän interaktio, kuten tehtävien ja ajastuksien listaaminen, luominen, muokkaaminen ja poistaminen. Käyttöliittymän on tarkoitus yksinkertaistaa järjestelmän käyttöä, sillä se keskittää kaikki toiminnot yhteen sovellukseen, pakottaa yhtenäiseen tapaan tehtävien ja ajastuksien muodostamisessa, automatisoi tehtävien ja ajastuksien dokumentointia ja auttaa tehtävien monitoroinnissa. Kaikki järjestelmän loppukäyttäjän toiminnallisuus hoidetaan käyttöliittymästä käsin. Järjestelmän käyttöliittymä on perustoiminnallisuuksiltaan yksinkertainen, eikä käyttöliittymän ulkoasuun ole kiinnitetty paljoa huomiota. Näistä syistä tässä dokumentissa ei esitetä käyttöliittymän ulkoasun rautalankamalleja, vaan tässä toiminnallisessa määrittelyssä keskitytään abstraktiin käyttäjän toimintojen määrittelyyn.

4.1.2 Tehtävät

Käyttöliittymän tehtävälistauksessa näytetään tehtävän nimi ja selite. Valitusta tehtävästä täytyy pystyä siirtymään nopeasti tehtävän muokkaus-näkymään ja tarkastelemaan tehtävän ajastuksia.

Tehtävien toteutustapana tulee käyttää jotain graafista ohjelmointitapaa, jolla tehtävien muokkaus on mahdollista käyttöliittymästä, ja samalla se pakottaa käyttäjät käyttämään samanlaista tehtävien toteutustapaa. Moni järjestelmän loppukäyttäjä ei ole ohjelmoija, joten graafisen ohjelmointitavan tuoma abstraktio pienentää järjestelmän käyttöönottokynnystä. Graafisesti luotu tehtävä muodostaa osatehtävistä koostuvan suorituspuun,

jossa edellisen osatehtävän tuloksen perusteella voidaan valita seuraava valittava osatehtävä tai keskeyttää tehtävä. Esimerkki tällaisesta tehtävän työnkulusta on esitetty kuvassa 2. Tässä esimerkin työnkulussa testataan, onko tiedosto olemassa, ja siirretään se uuteen polkuun, jos näin oli.



Kuva 2. Esimerkki tehtävän työnkulusta.

Käyttöliittymän kautta on tarkoitus luoda tehtäviä tähän tapaan, eli tehtävä sisältää alun ja lopun (punaiset kuviot) lisäksi muuttujien määrittelyjä (vihreä kuvio), valintoja (keltainen kuvio) ja ennalta määrättyjä *toimintoja* (sininen kuvio). Toiminnoille voidaan asettaa parametreja, kuten tässä esimerkissä käytetty toiminto siirtää tiedoston levyllä paikasta toiseen ja se saa nämä lähde- ja kohde-tiedostopolkunsa parametrina. Toiminnot abstrahoivat tehtävän logiikkaa sopivasti, jolloin tietyt toteutusyksityiskohdat ovat piilossa käyttäjältä mutta tehtävän työnkulku on silti selvästi nähtävillä. Toimintoja on tarkoitus olla monia useanlaisia ja niitä täytyy tarvittaessa pystyä luomaan uudenlaisia. Esimerkkinä mainittakoon osatehtävä joka lähettää sähköpostia. Tällaiselle toiminnolle annettaisiin parametrina luultavasti sähköpostiviestin otsikko, sisältö ja vastaanottajat. Toiminto hoitaisi sähköpostin lähetyksen ja samalla se piilottaisi käyttäjältä tehtävän logiikan kannalta turhat toteutusyksityiskohdat.

Lisätoimintona voidaan toteuttaa tehtävien suoritushistorian selaus käyttöliittymästä, johon päästään käsiksi tehtävälistasta käsin. Suoritushistoriassa näytetään ajat, milloin tehtävä on käynnistetty ja kauanko suoritus on kestänyt, sekä mahdollisuus yksittäisen tehtävän suorituskerran yksityiskohtaisen etenemisen tarkastelu. Etenemisen tarkastelu voidaan näyttää tekstimuotoisena lokiviestien sarjana. Käyttäjän kannalta vielä parempi

vaihtoehto olisi suorituskerran graafisen suorituspuun esittäminen, josta käyttäjä voisi katsoa työnkulun etenemisreitin, muuttujien arvot ja mahdolliset virheviestit. Tällainen suorituspuu olisi siis kuvan 4 tapainen esitys työnkulusta.

4.1.3 Ajastukset

Ajastukset tulee toteuttaa tehtävistä erillisinä hallittavina kokonaisuuksina. Yksittäisestä tehtävästä täytyy päästä käyttöliittymässä navigoimaan siihen liitettyjen ajastuksien listaukseen. Tämä mahdollistaa usean erillisen ajastuksen lisäämisen yksittäiselle tehtävälle, ja toisaalta tehtävällä ei ole pakko olla yhtään ajastusta asetettuna. Ajastus on aina linkitetty yhteen tehtävään, eikä se voi olla olemassa ilman tehtävää. Ajastus täytyy pystyä asettamaan tilapäisesti pois käytöstä. Tämä siksi että ajastus voi olla monimutkainen luoda, eikä sitä haluta poistaa tilapäisen käytöstä poiston vuoksi.

Ajastus täytyy pystyä asettamaan monipuolisesti. Ottamalla mallia Windows Task Schedulerin ajastustavoista, täytyy ajastus voida asettaa laukeamaan ainakin kerran heti tai jonain ajanhetkenä tulevaisuudessa, tai toistuvasti. Toistuvat ajastukset täytyy voida asettaa laukeamaan tietyn aikavälin välein, päivittäin jonain tiettynä aikana, joinain viikonpäivinä tiettynä aikana tai kuukausittain tiettyinä päivinä. Toistuvat ajastukset ovat oletusarvoisesti toiminnassa niiden luomishetkestä ikuisuuteen, mutta niille täytyy pystyä asettamaan toiminnan alkamis- ja loppumisajanhetket. Esimerkiksi päivittäin ajettava tehtävä voidaan haluta olevan toiminnassa ainoastaan viikon ajan.

4.1.4 Käyttäjät

Käyttöliittymän käyttäjien todennusta, valtuutusta ja tilastointia varten järjestelmään täytyy lisätä käyttäjä-ominaisuus. Tämä on lisätoiminnallisuutta jota ei ole pakko toteuttaa. Käyttäjiä täytyy olla kahdentyyppisiä, normaaleja ja ylläpitäjiä, joista jälkimmäisillä on erityiset oikeudet. Käyttäjien todennus käyttöliittymään hoidetaan käyttäjätunnus-salasana- parilla. Ylläpitäjä-käyttäjällä on mahdollisuus lisätä, muokata ja poistaa käyttäjiä.

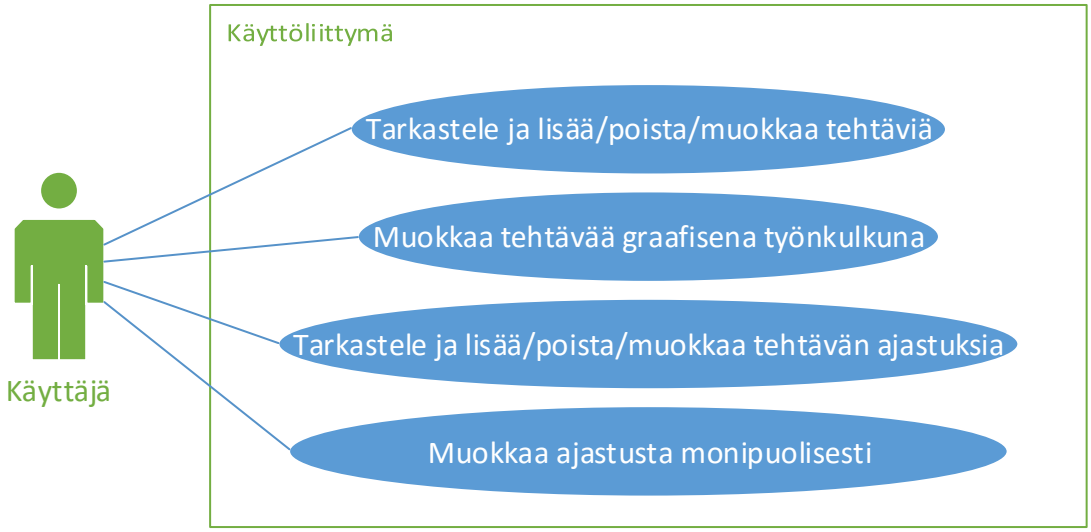
Suuressa järjestelmässä on todennäköistä, ettei kaikkien käyttäjien tarvitse tai kuulu nähdä kaikkia olemassa olevia tehtäviä. Tästä syystä käyttäjät valtuutetaan tehtäviin käyttäen käyttäjäryhmiä siten, että käyttäjäryhmälle voidaan lisätä oikeus johonkin tehtävään, jonka jälkeen kaikilla käyttäjillä jotka on linkitetty tähän käyttäjäryhmään, on myös oikeus tehtävään. Normaalin käyttäjän täytyy kuulua aina vähintään yhteen, mutta mahdollisesti useaankin käyttäjäryhmään. Tehtävän luoneen käyttäjän käyttäjäryhmälle tulee oikeus tehtävään, mutta jos käyttäjä kuuluu useaan käyttäjäryhmään, niin käyttäjän tulee valita käyttäjäryhmä tai -ryhmät, joille oikeus annetaan. Luonnin yhteydessä linkitetyille käyttäjäryhmille tulee *omistusoikeus* tehtävään. Jos käyttäjäryhmällä on omistusoikeus tehtävään, niin kaikilla sen ryhmän käyttäjillä on oikeus lisätä tai poistaa tehtävään linkitettyjä ei-omistavia käyttäjäryhmiä. Ylläpitäjä-käyttäjillä on automaattisesti kaikki oikeudet kaikkiin tehtäviin.

Kaikki käyttäjien tekemät järjestelmän muokkaukset tulee tilastoida ja näyttää käyttäjille. Järjestelmän muokkauksia ovat ainakin käyttäjien, tehtävien ja ajastuksien muokkaaminen. Ylläpitäjillä täytyy olla mahdollisuus katsella kaikkien tapahtumien aikajärjestyksessä olevaa tilastointia kokonaisuudessaan käyttöliittymästä. Normaaleille käyttäjille tilastoinnit tulee näkyä järkevästi aihepiiriin liittyen. Esimerkiksi tehtävän muokkaus -näkymässä käyttäjä voisi nähdä suoraan viimeisimmät tehtäviin liittyneet muokkaukset, ajankohdat, muokkauksen tehneen käyttäjän nimen ja käyttäjän jättämän kommentin. Tehtävän muokkauksesta tallennettu tilastointi, joka näkyisi *Default*- nimiselle käyttäjäryhmälle, voisi olla seuraavanlainen:

```
Date: 05-09-2016 12:32:01
UserGroup: Default
User: tuikkjar
Operation: Add
Comment: Esimerkki-tehtävä lisätty
```

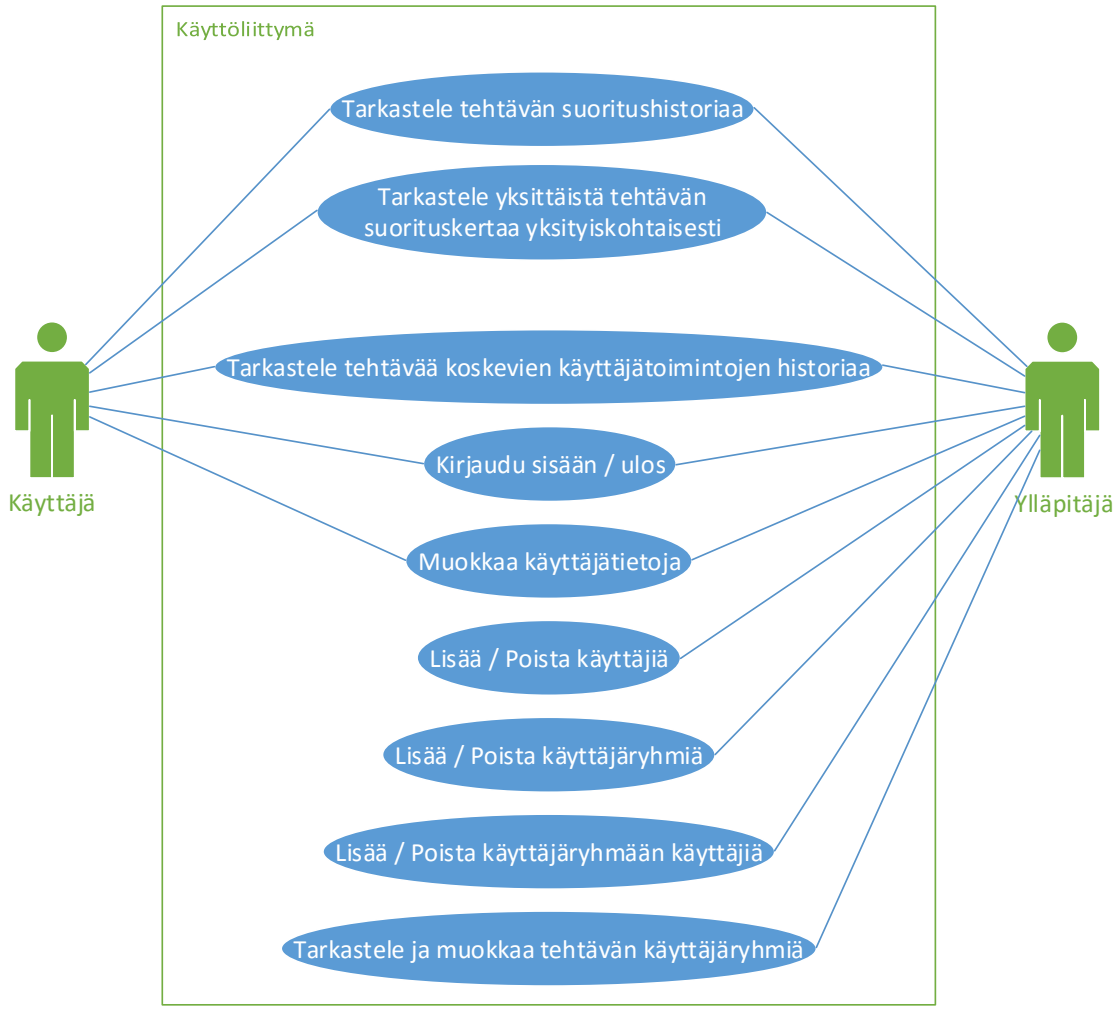
4.1.5 Käyttötapaukset

Kuvassa 3 on esitetty järjestelmän toiminnallisuuksien käyttötapaukset. Perustoiminnallisuuksissa käyttäjä pystyy tarkastelemaan tehtävälistausta ja tehtävän ajastuslistausta. Tehtäviä ja ajastuksia täytyy pystyä lisäämään, poistamaan sekä muokkaamaan. Tehtäviä muokataan graafisina työnkulkuina.



Kuva 3. Toiminnallisuuksien käyttötapaukset.

Kuvassa 4 on esitetty järjestelmän lisäominaisuuksien tuomat käyttötapaukset. Näitä ovat tehtävän suoritushistorian tarkastelu, yksittäisen tehtävän suorituskerran graafinen tarkastelu sekä käyttäjiin liittyvät toimenpiteet. Käyttäjiin liittyviä toimenpiteitä ovat käyttäjän kirjautuminen käyttöliittymään, käyttäjien tehtäviä tai ajastuksia koskevien toimenpiteiden historian näyttäminen sekä käyttäjätietojen, kuten salasanan, muokkaus. Lisätoiminnallisuuksissa sovelluksella on myös ylläpitäjiä, joiden täytyy pystyä lisäämään tai poistamaan käyttäjiä sekä käyttäjäryhmiä.



Kuva 4. Lisätoiminnallisuuksien käyttötapaukset.

Tehtävän suoritushistorian sekä yhden tehtävän suorituskerran yksityiskohtainen tarkastelu käyttöliittymästä käsin on tärkein lisäominaisuus, sillä se lisää käytettävyyttä eniten. Muut ovat ominaisuuksia jotka lisäävät lähinnä järjestelmän tietoturvaa ja siirrettävyyttä.

4.2 Ei-toiminnalliset vaatimukset

Järjestelmän komponenttien muistin ja muiden resurssien kulutukseen sekä toimintojen vasteaikoihin ei liity mitään varsinaisia vaatimuksia, sillä järjestelmän oletetaan olevan suhteellisen kevyt. Tehtäville on asetettu ei-toiminnallisia vaatimuksia liittyen tehtävien suoritustapaan, ajastuksille liittyen reaaliaikavaatimuksiin, tehtävien ja järjestelmän monitoroinnille liittyen datan tallennus- ja esitystapaan sekä tietoliikenteen tietoturvaan. Osa näistä on lisäominaisuuksia, joita ei ole pakko toteuttaa.

Tehtävä tulee ajaa aina samalla Windows -käyttäjätunnuksella, joka on asetettavissa järjestelmäkohtaisesti. Lisäksi tehtäviä täytyy suorittaa vain yksi instanssi kerrallaan; Toisin sanoen järjestelmän täytyy estää monen yhtäaikaisen saman tehtävän ajamisen. Vaikka

joskus voisi olla mielekästä ajaa samaa tehtävää eri parametreilla yhtaikaa[2], niin tällä rajoituksella halutaan estää virhetilanteita joita voi tulla vastaan ajettaessa samoja toimintoja yhtaikaa.

Ajastuksille asetetaan kovat reaaliaikavaatimukset. Jos tehtävällä on kovat reaaliaikavaatimukset, niin se tulee ajaa ajallaan tai ei ollenkaan, kun taas pehmeät reaaliaikavaatimukset omaava tehtävä voidaan ajaa myöhässäkin. Tässä järjestelmässä voidaan olettaa, että ajastuksen vasteaika, eli aikaväli ajastuksen laukeamisesta todelliseen tehtävän käynnistykseen, on verrattain lyhyt ja voidaan jättää huomiotta. Tehtävän käynnistys ajallaan voi silti epäonnistua ainakin, jos järjestelmä ei ole käynnissä lainkaan. Voi olla tehtäväkohtaista halutaanko sille kovat vai pehmeät reaaliaikavaatimukset. Kuitenkin yksinkertaisuuden vuoksi kaikkia tehtäviä tulee kohdella kuin ne omaisivat kovat reaaliaikavaatimukset. Jos siis jonkin tehtävän ajastuksen laukeaminen on jäänyt toteutumatta johtuen siitä, ettei järjestelmä ole ollut tuolloin käynnissä, niin kyseinen ajastuksen laukeaminen voidaan unohtaa.

Tehtävien monitorointi toteutetaan käytännössä tallentamalla tehtävien suorituksesta automaattisesti jonkinlaista monitorointitietoa pysyväistalteen joko tietokantaan tai tiedostoon. Yksittäisestä tehtävästä tallennettaisiin tähän lokiin ainakin tehtävän käynnistysajankohta, osatehtävien suoritusjärjestys ja näiden osatehtävien onnistuminen tai epäonnistuminen. Tällaista suorituslokia voidaan käyttää monipuolisesti virheiden paikantamiseen järjestelmää testatessa, mutta sitä voitaisiin käyttää myös tehtävän suorituksen havainnollistamiseen käyttöliittymässä.

Järjestelmän monitorointi toteutetaan lokia keräämällä. Komponenttien tulee kerätä lokiviestejä tietotakantaan. Testatessa voidaan lokia kerätä myös muihin formaatteihin, kuten tiedostoon, konsoliin tai käyttöliittymässä dialogiin. Lokiviestejä tulee muodostaa komponenteissa vakavuudeltaan eritasoisina, jotta lokia tarkastellessa voidaan nopeasti huomata, mikä on kyseisen viestin vakavuus ja toisaalta voidaan kontrolloida tallennettavan lokin määrää asettamalla ainoastaan vakavimmat lokiviestit tallennettaviksi. Tässä järjestelmässä tulee olla vähintään seuraavanlaiset lokitasot käytössä:

- Debug – Testausta varten.
- Information – Hyödyllinen tieto ohjelman toiminnasta.
- Warning – Odotettu virhe, ei vaadi korjausta.
- Error – Odottamaton virhe järjestelmässä, vaatii korjausta.

Käytännössä tuotannossa Information -tasoiset ja sitä vakavammat viestit tallennetaan pysyväistalteen, kun taas Debug -tasoisia viestejä tulee tallentaa ainoastaan järjestelmää testatessa. Esimerkki Warning -tasoisesta virheestä voisi olla sellainen, jossa käyttäjä yrittää tehdä jotain sopimatonta mihin on kuitenkin varauduttu. Error -tasoinen virhe voisi

[2] Jos luotuna on yleiskäyttöinen tehtävä, jolle saadaan annettua ajastuksen mukana parametreja

olla esimerkiksi tilanne, jossa johonkin verkon yli käytettävään palveluun ei saada yhteyttä. Järjestelmän lokitasot voivat olla erinimisiä ja lokitasoja voi olla enemmänkin, kunhan logiikka pysyy samana. Jos itse lokin kerääminen epäonnistuu, niin se on vakava virhetilanne, joka ei saa kuitenkaan kaataa ohjelmaa. Käytännössä lokin kerääminen voi epäonnistua johtuen vääränlaisesta konfiguraatiosta tai verkko-ongelmista. Monitoroinnin lisäominaisuutena voidaan toteuttaa jonkinlainen reaktiivinen järjestelmän monitorointitapa. Esimerkiksi sähköpostin lähettimen Error-tasoisista lokiviesteistä järjestelmän ylläpitäjille mahdollistaisi nopean virheisiin puuttumisen.

Järjestelmän tietoturvan ei-toiminnalliset vaatimukset liittyvät komponenttien väliseen tietoliikenteeseen. Järjestelmä toteutetaan suljettuun sisäverkkoon, joten järjestelmän tietoturva on korkealla tasolla johtuen infrastruktuurista. Tietoturvaa koskevat ominaisuudet parantavat kuitenkin järjestelmän uudelleenkäytettävyyttä kohtuullisella työmäärällä, jos ne on huomioitu järjestelmän suunnittelussa. Lisäominaisuutena vaaditaankin, että järjestelmän osien väliset kommunikoinnit täytyy pystyä salaamaan Transport Layer Security (TLS) -salauksella [38] ja komponentit täytyy pystyä todentamaan Primary Key Infrastructure (PKI) -sertifikaateilla [39].

4.3 Rajoitukset

Järjestelmän toteutuksen rajoittavia tekijöitä ovat asiakkaan tuotantoympäristön infrastruktuuri, siinä käytettävissä olevat tekniikat, sekä asiakkaan yleiset käytännöt. Luvussa 3 esitettiin asiakkaan tuotantoympäristön infrastruktuuri. Tämän järjestelmän muodostaa usea suljetussa IP -verkossa toimiva tietokone, joista osa on asiakkaan hallinnassa olevia palvelimia ja osa kolmannen osapuolen palvelimia. Näiden koneiden välillä on säädettävissä olevia palomuuri -palvelimia. Lisäksi tästä suljetusta verkosta on pääsy Internetiin yhdeltä palvelimelta, joka toimii *eteisverkossa*.

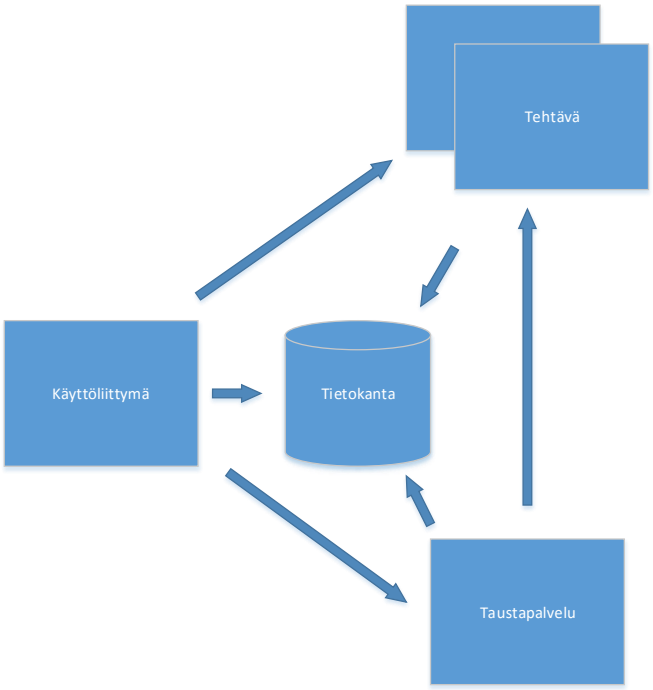
Kaikki asiakkaan koneet ovat Windows Server -palvelinkoneita joissa on asennettuna .NET -ohjelmistokehys sekä IIS. Käyttöjärjestelmät näissä koneissa ovat mallia Windows Server 2008 R2, tai sitä uudempia versioita. Minimiversio koneilla olevista .NET -versioista on 4.0, eikä esteitä uudempien .NET -versioiden tai muiden *middleware* -ohjelmien asennukselle ole, jos tarvetta tällaiselle esiintyy. IIS -versiot koneilla ovat vähintään 7.0. Lisäksi verkossa on ainakin yksi käytettävissä oleva tietokantapalvelin, jonka tietokannan hallintajärjestelmä on Microsoft SQL Server. Tietokantaan kirjautuminen tapahtuu käyttäen Windows -käyttäjätunnusta. Muita tietokantapalvelimia tuotantoympäristöön ei saa asentaa.

4.4 Arkkitehtuuri

Tässä kohdassa esitellään järjestelmän arkkitehtuuri, eli teknillinen määrittely. Ensiksi hahmotellaan järjestelmässä tarvittavat komponentit abstraktilla tasolla, sitten tarkennetaan arkkitehtuuria lisäämällä abstraktioon käytettävät tekniikat. Lopuksi esitellään järjestelmän lohkokaavio alakohdassa 4.4.3

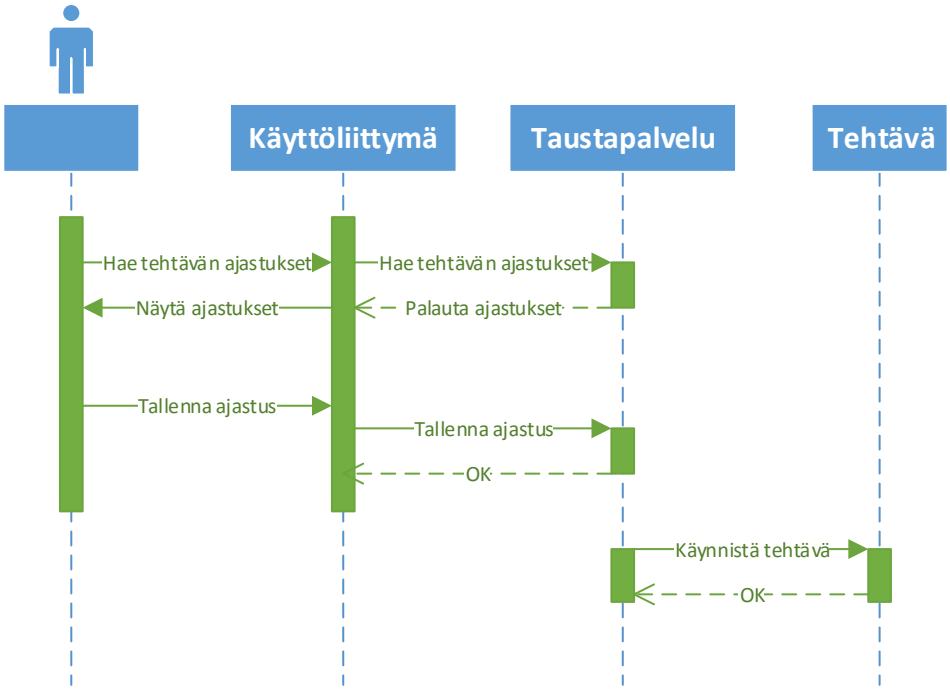
4.4.1 Abstraktio

Vaadittujen ominaisuuksien perusteella järjestelmässä on varmasti ainakin kaksi komponenttia: Käyttöliittymä ja tietokanta. Käyttöliittymä on prosessi, joka on käynnissä ainoastaan hetkellisesti. Ajastuksien täytyy kuitenkin olla käynnissä aina, joten järjestelmässä täytyy olla jokin taustapalvelu näitä varten. Käyttöliittymällä täytyy olla jokin tapa tallentaa ajastuksia tähän taustapalveluun. Tämän taustapalvelun täytyy olla datan suhteen pysyvä, eli sen täytyy pystyä palautumaan samaan tilaan, vaikka se sammutettaisiin hetkeksi. Sammutus voisi käytännössä johtua vaikkapa sähkökatkosta tai hallitusta sammuttamisesta päivitystä varten. Tästä syystä ajastustiedot täytyy olla pysyvässä tallessa tietokannassa. Lisäksi tietokantaa tulee käyttää myös taustapalvelun lokitietojen tallennukseen. Käyttöliittymässä muodostetaan tehtäviä ja ajastuksia; Ajastukset tallennetaan taustapalveluun ja tehtävät joko taustapalveluun tai jonnekin muualle. Tehtävien tallennussijainnilla ei sinänsä ole väliä, kunhan ajastusten lauetessa taustapalvelulla on jokin tapa käynnistää ajastukseen liitetty tehtävä. Tehtävä kannattaa kuitenkin tallentaa jollain yhtenäisellä tavalla siten, että tehtävän käynnistys on mahdollisimman helppoa. Kuvassa 5 on esitetty abstrakti järjestelmä, joka sisältää nämä edellä kuvatut komponentit.



Kuva 5. Järjestelmän abstraktio.

Kuvassa 6 on esitetty sekvenssikaavio tilanteesta, jossa käyttäjä tallentaa onnistuneesti ajastuksen tehtävälle, jonka jälkeen jossain vaiheessa taustapalvelu käynnistää tehtävän.



Kuva 6. Sekvenssikaavio ajastuksen tallentamisesta ja laukeamisesta

4.4.2 Tekniikoiden valinta

Kuvan 5 järjestelmän hahmotelma toimii pohjana toteutukselle. Tähän abstraktioon täytyy nyt valita varsinaiset toteutustekniikat. Lähdetään liikkeelle tehtävien toteutustekniikasta käyttöliittymässä, joka määrittelee aika paljon muita järjestelmän osia.

Tehtävien graafista toteutusta varten kannattaa käyttää jotain valmista tekniikkaa toteutuksen nopeuttamiseksi. .NET -ohjelmistokehyksen WF -tekniikka sopii tähän hyvin. Käyttöliittymän kannalta on tärkeää, että WF:lla voidaan määritellä työnkulkuja graafisesti, vaikka niitä voi määritellä myös ohjelmallisestikin. Lisäksi WF mahdollistaa uusien toimintojen määrittelemisen, joita WF:ssa sanotaan aktiviteeteiksi. Työkalua, jolla työnkulkuja voidaan WF:lla graafisesti muokata, kutsutaan Workflow Designer:ksi. Se on yleisesti käytettävissä Visual Studiosta, mutta sitä on myös mahdollista käyttää WPF-applikaatiossa. Internet-selaimessa Workflow Designeria ei voi käyttää. Tämän järjestelmän käyttöliittymän täytyy siis olla WPF -sovellus, jotta WF -tekniikkaa voidaan käyttää.

Käyttäen WF:ia meillä on tapa luoda tehtäviä graafisesti. Seuraavaksi täytyy mahdollistaa käyttöliittymässä luotujen WF -tehtävien käynnistäminen taustapalvelusta käsin. Käyttöliittymässä luodut WF -instanssit ovat ajettavia kirjastoja, joita voidaan sitten käynnistää samaan tapaan kuten mitä tahansa .NET- luokkaa [40]. WF- instanssit voidaan kuitenkin

luoda myös palveluina, jolloin ne tarjoavat Web Service -metodeja, joilla niiden kanssa voidaan vuorovaikuttaa. Meidän tapauksessamme riittää, että niitä voidaan käynnistää. Tällaista WF -palvelua voidaan isännöidä itse luodussa palvelussa tai ohjelmassa, mutta Microsoft suosittelee isännöimään WF-palveluita Windows AppFabric -palvelinohjelmassa, joka on IIS:n lisäys [41]. Käyttöliittymässä luodut palvelut saadaan lisättyä Windows AppFabriciin joko käsin lisäämällä, tai ohjelmallisesti käyttäen Web Deploy -työkalun ohjelmointirajapintaa.

Käyttöliittymän ja taustapalvelun välinen kommunikaatio ajastusten tallentamiseksi voidaan toteuttaa jollain prosessien välisellä kommunikaatiotavalla. Tähän tarkoitukseen soveltuu hyvin etäproseduurikutsut, jollaisia käytetään myös tehtävien käynnistämiseen. Taustapalvelu siis isännöi jonkin luokkarajapinnan toteuttavaa WCF -palvelua jota voidaan käyttää käyttöliittymästä käsin.

Seuraavaksi meidän täytyy ratkaista, miten hoidamme ajastus-logiikkaa taustapalvelussa. Tätäkään tarkoitusta varten ei kannata itse lähteä suunnittelemaan ja toteuttamaan uutta komponenttia, vaan kannattaa käyttää olemassa olevia toteutuksia. Nopeallakin etsinnällä löytyy kolme käyttökelpoista .NET -kirjastoa tähän tarkoitukseen: Quartz.NET, NCron [42], ja Task Scheduler Managed Wrapper [43], jolla voidaan ohjelmallisesti käyttää Windows Task Scheduleria. Aikaisempaa kokemusta näiden komponenttien käytöstä meillä ei ollut ja nopealla tutkinnalla mikä vain näistä näyttäisi toimivan tässä järjestelmässä. Valinta osuu kuitenkin Quartz.NET:iin, joka on muita vaihtoehtoja hieman parempi tähän järjestelmään. Task Scheduler Managed Wrapper pudotetaan pois ehdokkaista, koska sitä käyttäen tehtävän ajastuksen lauetessa täytyisi käynnistää jokin ohjelma, kun taas Quartz.NET:llä ja NCron:lla suoritetaan jokin ohjelmoitu toiminto, jossa voidaan käytännössä tehdä mitä vain. Koska meidän tapauksessamme ajastuksen lauetessa täytyy suorittaa yksinkertainen etäproseduurikutsu haluttuun WF -palveluun, niin sellaisen tallennus levylle ohjelmaksi olisi epäkäytännöllistä. Quartz.NET:n ja NCron:n vaikuttavat hyvin paljon samankaltaisilta, mutta Quartz.NET on ylläpidetympi ja huomattavasti enemmän käytetty kuin NCron, ja näin ollen luotettavampi valinta. Quartz.NET tarjoaa käytännössä konfiguroitavan vuorontaja-luokan, joka hoitaa ajastuksien toteutumisen. Konfiguroinnilla voidaan muun muassa määrittää ovatko ajastukset tallessa muistissa vai pysyväistallessa tietokannassa. Ajastuksia voidaan tällä komponentilla määritellä erittäin monipuolisesti ja ajastuksen lauetessa suoritettava toiminto on täysin itse määriteltävissä ohjelmallisesti. Käyttäen WCF:ia ja Quartz.NET:ia meidän on siis mahdollista luoda etähallittava ja pysyvä taustapalvelu, joka hoitaa ajastuksia.

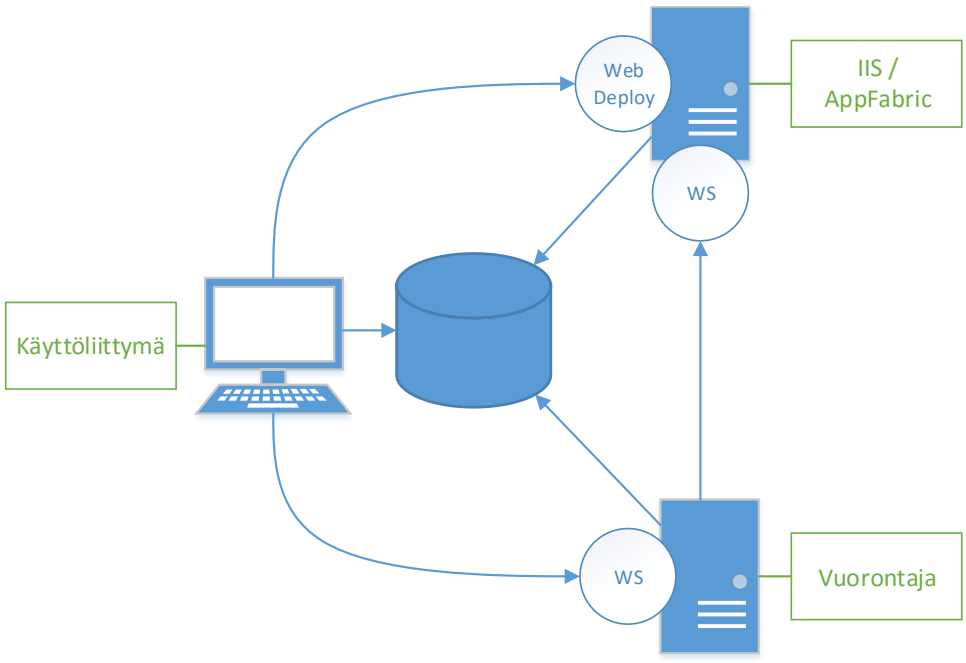
Lisäksi oli määritetty, että ohjelmien tulee kerätä lokia tietokantaan. Meillä oli tietokantaan lokin keräämistä varten olemassa oleva .NET -kirjasto, jota tässä järjestelmässä tullaan käyttämään. Tätä komponenttia käyttäen tietokannan taulujen rakenne täytyy olla määrätynlainen ja viestit tallennetaan luokkarajapintaa käyttäen. Voi tulla kuitenkin vastaan tilanne jossa lokiviestit haluttaisiin tallentaa johonkin muuhun formaattiin ja ainakin

järjestelmää testatessa lokia voidaan haluta kerätä yksinkertaisesti tiedostoon tai konsoliin. Common.Logging on komponentti, joka abstrahoi lokin keräämisen siten, että ohjelmassa määritetyt lokien julkaisut eivät vielä määrää lokin varsinaista tallennustapaa, vaan käytettävä lokitoteutus on liitettävissä ohjelmaan erikseen. Common.Logging tukee ainakin log4net- ja NLog -lokitoteutuksia [9]. Log4net -kirjastoa olemme käyttäneet ennenkin, joten se valitaan käytettäväksi loki-implementaatioksi kokemuksen vuoksi. Log4net:llä ohjelman tuottamat lokiviestit voidaan yhtäaikaisesti tallentaa moneen paikkaan käyttäen konfiguraatiossa lisättäviä moduuleja. Konfiguraatiolla voidaan lisäksi määrittää myös se, minkä tasoisia lokiviestejä moduuli välittää eteenpäin. Näitä moduuleja on valmiina useita, mutta niitä voidaan luoda myös itse. Tätä järjestelmää varten täytyy luoda kaksi uutta moduulia: Meidän valmista lokikomponenttia käyttävä sekä sähköpostia lähettävä moduuli.

Taustapalvelu täytyy pystyä asentamaan Windows -palveluna. Tässä järjestelmässä tullaan käyttämään TopShelf -komponenttia ohjelman asentamiseksi Windows-palveluksi. TopShelfiä käyttäen normaali Windows -applikaatio saadaan helposti asennettua palveluksi sopivilla parametreilla, mutta se mahdollistaa myös palvelun ajamisen prosessina, joka on kätevää ohjelmaa testatessa. Tätä komponenttia tullaan siis käyttämään ainoastaan taustapalvelussa.

4.4.3 Lohkokaavio

Lisäämällä edellä mainitut tekniikat kuvassa 5 esitettyyn järjestelmän abstraktioon saadaan tarkempi määrittely käytettävästä järjestelmästä. Tämä järjestelmän ylemmän tason lohkokaavio löytyy kuvasta 7.



Kuva 7. Järjestelmän lohkokaavio.

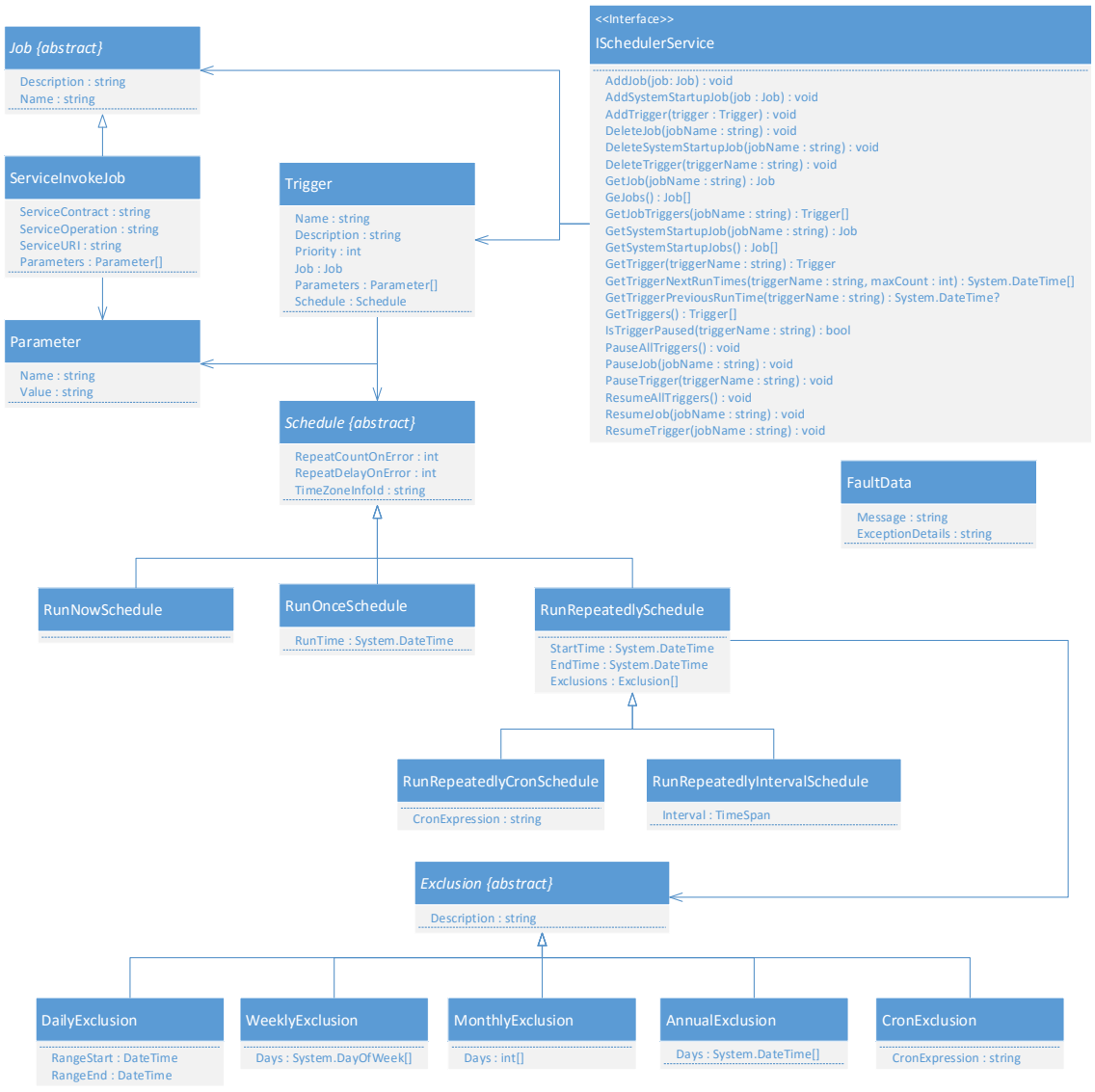
Tietokantana toimii siis vaatimusten mukaisesti Microsoft SQL Server. IIS, AppFabric ja WebDeploy ovat valmiita komponentteja. Käyttöliittymä ja taustapalvelu, joka kuvassa 6 on nimetty *vuorontajaksi*, täytyy toteuttaa itse käyttäen mainittuja tekniikoita.

5. TOTEUTUS

Tässä luvussa esitellään vuorontajan rajapinta sekä vuorontajan että käyttöliittymän toteutukset. Toteutus suoritettiin kahden ihmisen ryhmässä iteratiivisesti määritelmien pohjalta, luoden aluksi rungon toteutukselle ja sitten lisäten ominaisuuksia palasittain.

5.1 Vuorontajan rajapinta

Kuvassa 8 on esitetty vuorontajan rajapinnan luokkakaavio. Rajapintaluokka *ISchedulerService* sisältää kaikki metodit, jotka vuorontaja tarjoaa käyttöliittymälle. Nämä metodit mahdollistavat monipuoliset tehtävien ja ajastuksien haut, lisäykset ja poistot käyttöliittymästä käsin.



Kuva 8. Vuorontajan rajapinnan luokkakaavio.

Huomattavaa on, että *ISchedulerService* -rajapintaa käyttäen käyttöliittymästä lisätään vuorontajaan erikseen tehtävä ja tehtävään liittyvä ajastus. Tällainen erottelu tehtiin jo määrittelyvaiheessa luvussa 4. Rajapinta käyttää luokkia *Trigger*, *Job*, *Parameter*, *Schedule* ja *Exclusion* datan määrittelyyn. *Parameter* mahdollistaa tehtävän käynnistämisen parametreilla, ja on huomattavaa, että sekä *Job* että *Trigger* sisältävät parametrilistauksen. Tällä mahdollistetaan se, että tehtävä voi sisältää jotkin oletusarvoiset parametrit, jotka mikä tahansa tehtävän ajastus voi ohittaa omilla erilaisilla parametreillaan. *Job*, *Schedule* ja *Exclusion* ovat abstrakteja kantaluokkia, joista on periytetty aliluokat toteuttamaan tarvittavat tarkentavat toiminnallisuudet. Käyttöliittymä ja vuorontaja saattavat sijaita erillisillä palvelimilla ja nämä palvelimet saattavat sijaita jopa erillisillä aikavyöhykkeillä; Tästä syystä *Schedule*-luokka sisältää *TimeZoneInfoId* -kentän jolla voidaan määrittää tarkalleen mitä aikavyöhykettä tarkoitettu aika, esimerkiksi *RunOnceSchedule* -luokan *RunTime* -ominaisuus, käyttää.

Rajapintaan on lisätty muutama metodi jotka mahdollistavat lisätoimintoja, jotka eivät kuulu pääasialliseen toimintaan, kuten mahdollisuus ajastuksien tilapäiseen disablointiin (esimerkiksi *PauseAllTriggers()*) ja jälleen-enablointiin (esimerkiksi *ResumeAllTriggers()*), mahdollisuus hakea ajastuksien tulevia (*GetTriggerNextRunTimes()*) ja menneitä laukeamisaikoja (*GetTriggerPreviousRunTime()*), sekä mahdollisuus lisätä tehtäviä jotka suoritetaan Vuorontaja-palvelun käynnistyessä (*SystemStartupJob*).

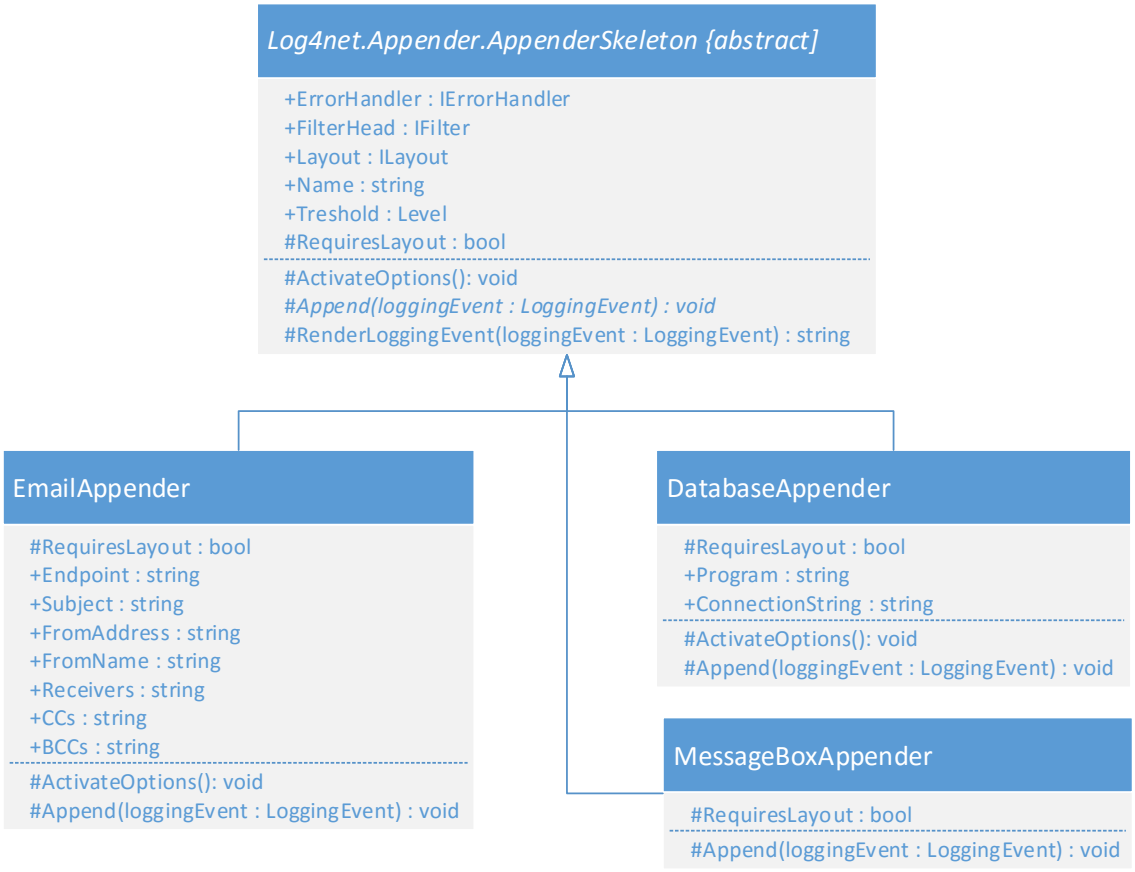
Virheiden hallintaan ei käytetä paluuarvoja tai -viestejä, jotka ilmaisisivat onnistuiko metodin suoritus vai ei. Jos metodia suorittaessa sattuu mikä tahansa virhe, niin tästä informoidaan metodin kutsujaa palauttamalla *SOAP Fault* [42] joka sisältää *FaultData*-luokan. Käytännössä metodin kutsujan kannalta tämä näyttäytyy kuin metodin kutsu tuottaisi poikkeuksen joka voidaan käsitellä. Virheiden tuottaminen ei saa kuulua järjestelmän normaaliin toimintaan, vaan ne liittyvät aina johonkin järjestelmän virheeseen joka vaatii korjauksen. Virheet ovat siis tarkoitettu lähinnä järjestelmän kehittämisen apuvälineeksi. Tietysti tuotetta käytettäessä on mahdollista, että sattuu jotain odottamatonta, josta on hyvä informoida käyttäjää. Esimerkiksi jos yhteys tietokanta-palvelimeen ei syystä tai toisesta toimi, on tällainen tilanne.

5.2 Lokin kerääminen

Toteutettavien komponenttien lokiviestin muodostus toteutettiin Common.Logging -rajapinnalla ja log4net -lokitoteutuksella, joista ensimmäinen mahdollistaa lokiviestien lisäämisen abstraktilla tavalla haluttuihin kohtiin ohjelmakoodia lokitoteutuksesta riippumatta ja jälkimmäinen lokiviestien tallentamisen monipuolisesti erilaisiin kohteisiin valmiilla tai kustomoitavilla *LogAppender* -moduuleilla.

Log4net sisältää valmiit, parametrisoitavat toteutukset lokien ohjaamiseksi esimerkiksi konsoliin, tiedostoon, sähköpostiin tai tietokantaan. Tässä järjestelmässä tietokantaan kir-

joitus halutaan kuitenkin tehdä valmiilla loki-komponentilla, sähköpostien lähetys asiakkaan oman sähköpostipalvelun kautta ja käyttöliittymässä virheiden esittäminen virhedialogin avulla, joten näitä varten täytyy luoda omat *Log4net.IlogAppender*- toteutukset. Testatessa lokiviestejä voidaan ohjata myös tiedostoon tai konsoliin log4net:n valmiilla toteutuksilla. Kuvassa 9 on esitetty luokkakaavio lokiviestien ohjaukseen käytettävistä kustomoiduista luokista *EmailAppender*, *DatabaseAppender* ja *MessageBoxAppender*, jotka periytetään abstraktista kantaluokasta *Log4net.Appender.AppenderSkeleton*.



Kuva 9. Lokiviestien ohjaukseen käytettävät luokat.

AppenderSkeleton mahdollistaa yhtenäisen tavan Appender -luokkien asetuksien, viestien formaattien, ja viestien suodattimien konfigurointiin asetustiedostossa. Kuvan 9 luokkakaaviossa on esitetty sekä luokkien julkiset, että suojatut ominaisuudet. Yksityiset metodit ja ominaisuudet, sekä tämän järjestelmän kannalta merkityksettömät *AppenderSkeleton* -luokan metodit, on jätetty merkitsemättä kuvaan. Appender -luokissa täytyy toteuttaa kantaluokan abstrakti metodi *Append()*, jossa *LoggingEvent* -tyyppiä oleva lokiviesti voidaan ohjata haluttuun määränpäähän. *LoggingEvent* -objektia ei itsessään tarvitse käsitellä, vaan sen saa muutettua merkkijonoksi kätevästi kantaluokan *RenderLoggingEvent()* -metodilla, jonka lopputuloksen formaatti määräytyy konfiguraatiossa määrätyn *Layout*:n perusteella. Tästä syystä Appender -toteutuksissa on toteuttu *RequiresLayout* -ominaisuus, joka on aina tosi, jolloin syntyy virhe *RenderLoggingEvent()* -metodissa jos *Layout* -ominaisuutta ei ole määritetty.

Jos toteutuksien *Append()* -metodeissa tapahtuu virheitä, niin ne välitetään *ErrorHandler* -ominaisuudella *IErrorHandler* -rajapinnan *Error()* -metodille. *IErrorHandler* -moduulin tarkoituksena on tehdä jotain virheelle, joka tapahtuu itse lokin keräämisessä. Oletuksena käytetään *log4net.Util.OnlyOnceErrorHandler* -toteutusta, joka kirjoittaa kyseisen Appender -toteutuksen ensimmäisen virheviestin konsoliin. Tähän olisi ehkä järkevää tehdä vaikkapa sellainen virhekäsittelijä, joka kirjoittaa viestin tiedostoon tai jopa kaataa ohjelman. Tässä on kuitenkin oletettu, ettei lokin kerääminen ikinä epäonnistu, joten siihen ei ole varauduttukaan.

ActivateOptions() -metodia kutsutaan kun luokan instanssi on alustettu ja kaikki instanssin ominaisuudet on asetettu. Tässä metodissa voidaan siis suorittaa kaikki lokikomponentin alustukseen liittyvät toimenpiteet, kuten parametrien validointi ja tietoliikenneyhteyksien testaus.

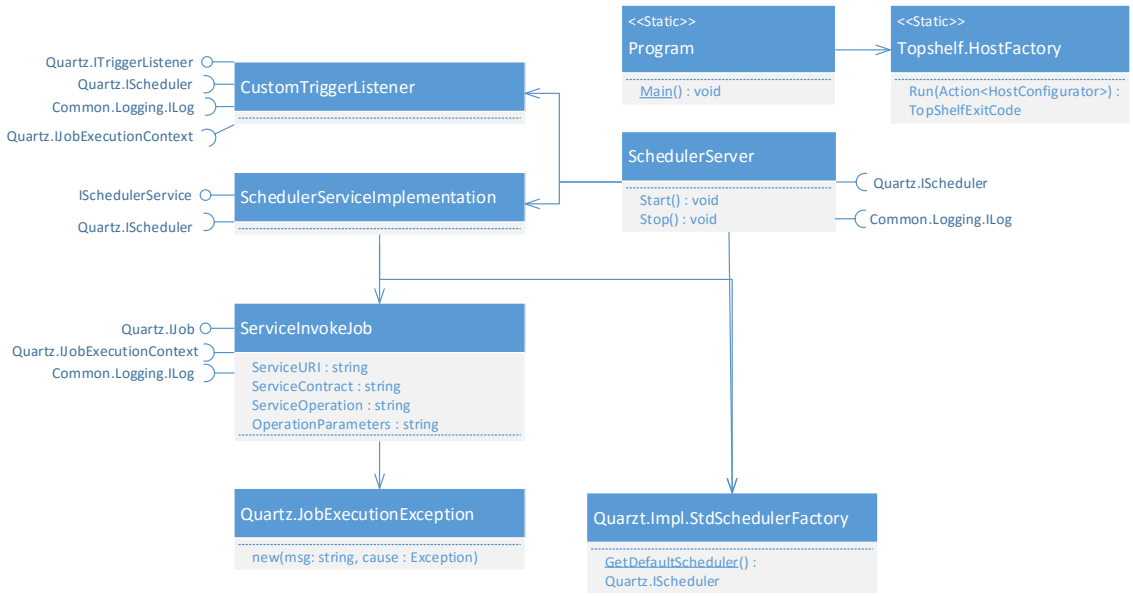
DatabaseAppender tallentaa lokiviestin SQL Server -tietokantaan. Tietokanta täytyy olla valmiiksi luotuna tätä komponenttia käytettäessä. *DatabaseAppender* vaatii toimiakseen *Program* ja *ConnectionString* -parametrit. Program -asetus sisältää lokiviestiin liitettävän ohjelman nimen ja ConnectionString sisältää tietokanta-yhteyden asetukset.

EmailAppender lähettää lokiviestin sähköpostilla halutulle jakelulle. Tämän järjestelmän asiakkaan järjestelmässä sähköpostia ei voida lähettää suoraan, vaan sähköpostiviestit lähetetään WCF -palvelulle, joka lähettää viestin oikealle sähköpostipalvelimelle. *EmailAppender* vaatii toimiakseen seuraavat parametrit: *Endpoint*, *Subject*, *FromAddress*, *FromName*, *Receivers*, *CCs* ja *BCCs*. Näistä esimerkiksi Endpoint sisältää Sähköpostipalvelun osoitteen ja Receivers sähköpostin vastaanottajat puolipilkuilla erotettuina.

MessageBoxAppender luo uuden WPF- ikkunan, joka sisältää viestin tapahtumasta. Viesti-ikkuna on helppo tapa informoida käyttäjää ohjelman virheistä. Tätä toimintoa kannattaa käyttää tuotantokäytössä luonnollisesti ainoastaan virheistä tiedottamiseen, sillä jatkuva ylimääräisten ilmoitus-ikkunoiden avautuminen olisi rasittavaa käyttäjälle.

5.3 Vuorontaja

Vuorontajan toteutuksessa on kolme huomioitavaa kohtaa: Se täytyy pystyä asentamaan Windows -palveluna, sen täytyy sisältää *IschedulerService* -rajapinnan toteuttava WCF-palvelu, sekä tehtävien vuoronnukseen käytetään siinä *Quartz.Ischeduler* -rajapinnan toteuttavaa luokkaa. Vuorontajan konfigurointi tapahtuu sen konfiguraatiotiedoston avulla. Konfiguroitavia osia ovat Quartz -vuorontaja, WCF -palvelu sekä lokin keräykseen käytetyt komponentit Common.Logging ja log4net. Kuvassa 10 on esitetty Vuorontajan luokkakaavio.



Kuva 10. Vuorontajan luokkakaavio.

Palvelun rungon muodostavat kuvassa 10 esitetyt luokat *Program*, *Topshelf.HostFactory* ja *SchedulerServer*. *Program* -luokka sisältää ainoastaan staattisen metodin *Main()*, jolla ohjelma käynnistetään. .NET -sovelluskehystä käytettäessä *Main()* -metodissa ei tarvitse olla parametrina ohjelmalle annettavien parametrien listaus, vaan parametrit pystyy ohjelmassa käsittelemään staattisen *Environment* -luokan avulla. *Main()* -metodissa kutsutaan staattisen *Topshelf.HostFactory*-luokan *Run()*- metodia, jolla saadaan aikaiseksi palvelun asennus, poistaminen tai käynnistäminen riippuen komentoriviparametreista. *SchedulerServer* on palvelun tärkein luokka. Sen tarkoitus on mahdollistaa palvelun logiikan käynnistäminen ja sammuttaminen *Start()* ja *Stop()* -metodeilla. Palvelun käynnistyksessä konfiguroidaan ja käynnistetään *Quartz.IScheduler*, johon päästään käsiksi *Quartz.Impl.StdSchedulerFactory* -luokan *GetDefaultScheduler()* -metodilla, sekä IschedulerService -rajapinnan toteuttava SchedulerServiceImplementation.

ISchedulerService -rajapinta on toteutettu *SchedulerServiceImplementation* -luokassa, jonka toiminta on suoraviivaista. Jokaisen metodin käsittelyssä tarvittaessa muutetaan *ISchedulerService*- rajapinnan luokkien data *Quartz.Ischeduler*:lle sopivaan malliin ja kutsutaan *Quartz.IScheduler* -rajapinnan metodeja sopivalla tavalla. Ohjelmassa 1 on esitetty esimerkin vuoksi *DeleteJob()* -metodin toteutus *SchedulerServiceImplementation* -luokassa. Metodi saa parametrina poistettavan tehtävän nimen, jonka se poistaa Quartz -vuorontajasta, ja heittää poikkeuksen, jos tehtävää ei löydy.

```
public void DeleteJob(string paJobName)
{
   try
   {
      var scheduler = Quartz.Impl.StdSchedulerFactory.GetDefaultScheduler();
      var jobKeys = scheduler
         .GetJobKeys(GroupMatcher<JobKey>.GroupEquals(typeof(ServiceInvokeJob).Name))
         .Where(jobkey => jobkey.Name == paJobName);
      if (jobKeys.Any())
      {
         System.Diagnostics.Debug.Assert(jobKeys.Count() == 1,
            "Multiple jobkeys with the same name");
         scheduler.DeleteJob(jobKeys.First());
      }
      else
      {
         throw new Exception("job with name " + paJobName + " not found");
      }
   }
   catch (Exception e)
   {
      throw new FaultException<FaultData>(
         new FaultData(e), new FaultReason("Delete failed; " + e.Message));
   }
}
```

Ohjelma 1. SchedulerServiceImplementation.DeleteJob()- metodin toteutus.

Metodin toteutus sisältää *Assert* -lauseen, joka on käytössä vain *debug* -käännöksessä ja on tarkoitettu ohjelman logiikan testaamiseen. Jos *debug* -käännöstä ajettaessa *Assert*:n väite ei ole tosi, niin ohjelma esittää virheviestin käyttäjälle, jonka jälkeen testaajalla on mahdollisuus keskeyttää tai jatkaa ohjelman toimintaa. [45]

Tehtävien vuoronnukseen käytettävä *Quartz.Ischeduler*- rajapinnan toteuttava objekti saadaan *Quartz.Impl.StdSchedulerFactory*-luokan staattisella *GetDefaultScheduler()* -metodilla. Metodi luo vuorontaja-objektin, jos sitä ei oltu vielä luotu, ja palauttaa sen paluuarvona. Tehtävien vuoronnusta varten on tarvinnut luoda kaksi luokkaa: *ServiceInvokeJob* ja *CustomTriggerListener*.

ServiceInvokeJob toteuttaa *Quartz.IJob* -rajapinnan, ja tämä on ainut tehtävätyyppi, jota Vuorontajassa tarvitsee käyttää. Sen toteuttamassa *Quartz.IJob.Execute()* -metodissa suoritetaan tehtävän käynnistävä etämetodikutsu parametreissa määriteltyyn palveluun. Vuorontajan ei tarvitse tietää minkälainen rajapinta kutsuttavalla palvelulla on, vaan ServiceInvokeJob:ssa voidaan hakea palvelun Web Services Description Language (WSDL) -kuvaus [46] kutsuttavalta palvelulta, ja vasta tämän jälkeen suorittaa itse operaatio. Tällaisesta toiminnasta ei tosin ole juurikaan hyötyä, sillä jokaisen tässä ohjelmassa käytettävän WF -palvelun rajapinta on samanlainen, mutta ei mainittavaa haittaakaan, sillä se ei mainittavasti hidasta ServiceInvokeJob -tehtävän suoritusta. Jos *ServiceInvokeJob*:n suorituksessa tapahtuu virheitä, niin tällaisista heitetään *Quartz.JobExecutionException* -poikkeus, jonka Quartz -vuorontaja automaattisesti käsittelee kirjoittamalla lokiviestin tapahtuneesta virheestä.

CustomTriggerListener toteuttaa *Quartz.ITriggerListener* -rajapinnan. Tämän luokka mahdollistaa suoritettavan toimintoja esimerkiksi silloin, kun jokin ajastus on laukeamassa (*TriggerFired()*) tai on lauennut ja ajastukseen linkitetty tehtävä on suoritettu (*TriggerComplete()*). *CustomTriggerListener* sisältää toteutuksen ainoastaan *TriggerComplete()* -metodille. Tuolla metodilla on kaksi tehtävää: Poistaa ajastukseen mahdollisesti liitetyt *Quartz.ICalendar* -määritykset, jos ajastus ei ikinä enää laukea, sekä mahdollistaa suoritetun tehtävän toistaminen tietyn viiveen jälkeen, jos tehtävässä sattui jokin virhe. Ajastuksiin liitetyt kalenterimääritykset, joilla suljetaan pois aikavälejä ajastuksen laukeamisajoista, täytyy poistaa tällä tavalla, muutoin ne jäävät roskaamaan vuorontajan tietokantaa. Tehtävien suorituksen toisteisuus virheen sattuessa voi olla haluttu ominaisuus joidenkin tehtävien suorituksessa ja se on mahdollistettu edellä mainitulla tavalla. Tehtävän toisteisuus virheen sattuessa riippuu ajastuksen *Schedule* -kentän *RepeatCountOnError*- ja *RepeatDelayOnError* -kenttien arvoista.

5.4 Käyttöliittymä

Käyttöliittymän toteutuksen pohjana käytettiin Microsoft:n referenssitoteutusta WF -editoria isännöivästä WPF -käyttöliittymästä, joka toimii yhdessä IIS/AppFabric:n kanssa Web Deployn avulla. Tämä referenssitoteutus löytyy osoitteesta http://social.technet.microsoft.com/wiki/contents/articles/1577.appfabric-reference-implementation-manage-the-lifecycle-of-a-workflow-service.aspx.

Referenssitoteutusta pystyttiin käyttämään osin suoraan tässä toteutuksessa. Käytännössä ainoat merkittävät muutokset referenssitoteutukseen olivat *ISchedulerService* -rajapinnan käyttäminen sekä ajastusnäkymien lisäys. Seuraavaksi esitellään referenssitoteutuksen ominaisuudet, joita käytettiin toteutuksessa, tai jotka poistettiin toteutuksesta. Tämän jälkeen esitellään pääpiirteet tehtävien muokkauksesta, aktiviteettien luomisesta, ajastuksien käsittelystä, tehtävien monitoroinnista sekä käyttöliittymän konfiguroinnista.

5.4.1 Referenssitoteutuksen ominaisuudet

Referenssitoteutus sisälsi paljon hyviä ominaisuuksia, joista osaa käytettiin suoraan ja osa jätettiin pois toteutuksesta. Suoraan käytettyjä ominaisuuksia olivat WF designer -käyttöliittymäkontrollin isännöinti WPF -sovelluksessa, mahdollisuus luoda omia aktiviteetteja työnkulkuihin, graafisen työnkulun kääntäminen WF -palveluksi sekä WF -palvelun siirto isännöitäväksi IIS:een. Ominaisuuksia, jotka otettiin pois käytöstä, olivat WF -palveluiden versiointi, WF -palveluiden käytön suojeleminen käyttäjätunnuksilla, WF -palveluiden kontrollointi, suoritettujen WF-palveluiden historian tutkiminen, yksittäisen WF -palvelun suorituskerran etenemisen graafinen esittäminen, sekä WF -palveluista tilastoitavien sovelluskohtaisten tietojen tallennus sekä esittäminen. Käytöstä poistettuja ominaisuuksia kuvataan seuraavaksi hieman tarkemmin.

WF -palveluiden versioinnilla voitaisiin IIS:een siirretyt palvelut versioida. Tällöin yksittäisestä WF -palvelusta voisi olla yhtä aikaa käynnissä useita versioita ja palvelun kutsuja käyttäisi palvelun jotain tiettyä versiota. Tämä ei olisi huono ominaisuus tässäkään järjestelmässä, mutta se otettiin pois käytöstä yksinkertaistamisen vuoksi. WF -palveluiden käytön suojeleminen käyttäjätunnuksilla otettiin pois käytöstä yksinkertaistamisen vuoksi, mutta myös siksi että järjestelmän tietoliikenteen tietoturvallisuutta aiotaan tarvittaessa toteuttaa PKI -sertifikaateilla ja TLS salauksella. WF -palveluiden kontrollointi tarkoittaa IIS:ssä tallessa olevien palveluiden käynnistämistä, pysäyttämistä ja jatkamista käyttöliittymästä käsin. Tällä ominaisuudella voitaisiin käynnistää virheeseen pysähtynyt työnkulku uudelleen siitä kohdasta, mihin se on pysähtynyt. Tällainen vaikutti kuitenkin olevan turha ominaisuus tässä järjestelmässä, joten sitä ei haluttu käyttää. Suoritettujen WF-palveluiden historian tutkiminen ja yksittäisen suorituskerran – onnistuneen tai epäonnistuneen – graafinen esittäminen perustuvat AppFabric Monitoring -ominaisuuteen. Tämä olisi hyvä ominaisuus tässäkin järjestelmässä, mutta sen käytössä koettiin olevan konfiguraatio-ongelmia, jotka vaatisivat korjauksia. Sovelluksen tilastoitavan datan tallentaminen työnkuluissa mahdollistaisi jonkin sovelluskohtaisen mitattavan datan esittämisen käyttöliittymässä. Tämä olisi mielekästä, jos haluttaisiin näyttää jonkin saman työnkulun tilastointia eri parametreilla ja käynnistysajoilla. Tässä järjestelmässä varmaankin ainoa mielenkiintoinen data, mitä työnkulkujen toteutumisesta voitaisiin näyttää, olisi niiden suoritukseen kulunut aika, ja se voidaan tarvittaessa esittää ilman tätä ominaisuutta.

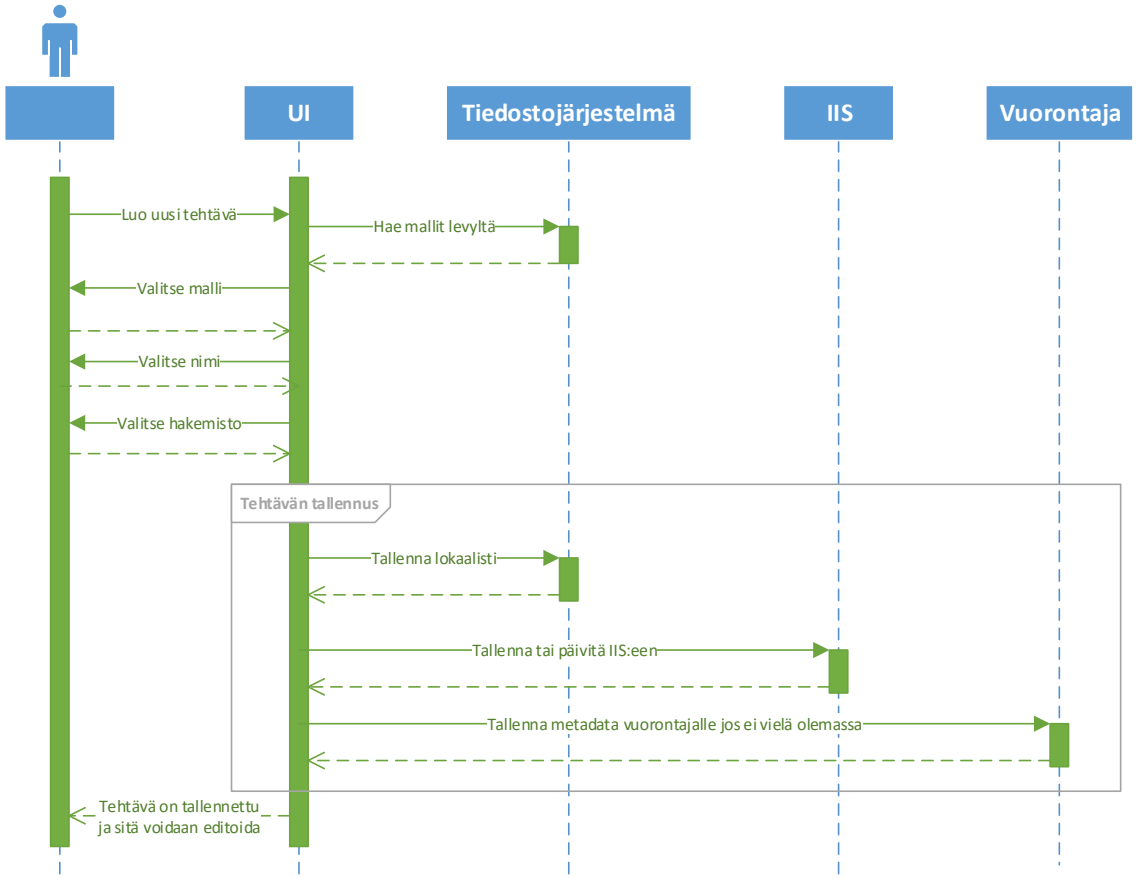
5.4.2 Tehtävät

Tehtävät ovat yhtä aikaa tallessa käyttöliittymän lokaalissa tiedostojärjestelmässä sekä AppFabric:ssa ja tehtävien metatiedot ovat tallessa lisäksi vuorontajassa. Näiden tietojen täytyy olla synkronissa; Jos vaikka lokaalit datat tehtävistä häviävät syystä tai toisesta, niin tällöin tehtävän muokkaaminen ei enää onnistu vaan se täytyy poistaa ja luoda uudelleen.

Käyttöliittymä avautuu näkymään missä näkyy kaikki järjestelmän tehtävät. Tehtävien metatiedot haetaan vuorontaja -palvelusta ja varsinainen data työnkulun editointia varten ladataan lokaalista tiedostojärjestelmästä. Tehtävä -näkymässä voidaan lisätä tai importoida uusia tehtäviä järjestelmään. Olemassa olevia tehtäviä voidaan muokata, poistaa tai ajaa.

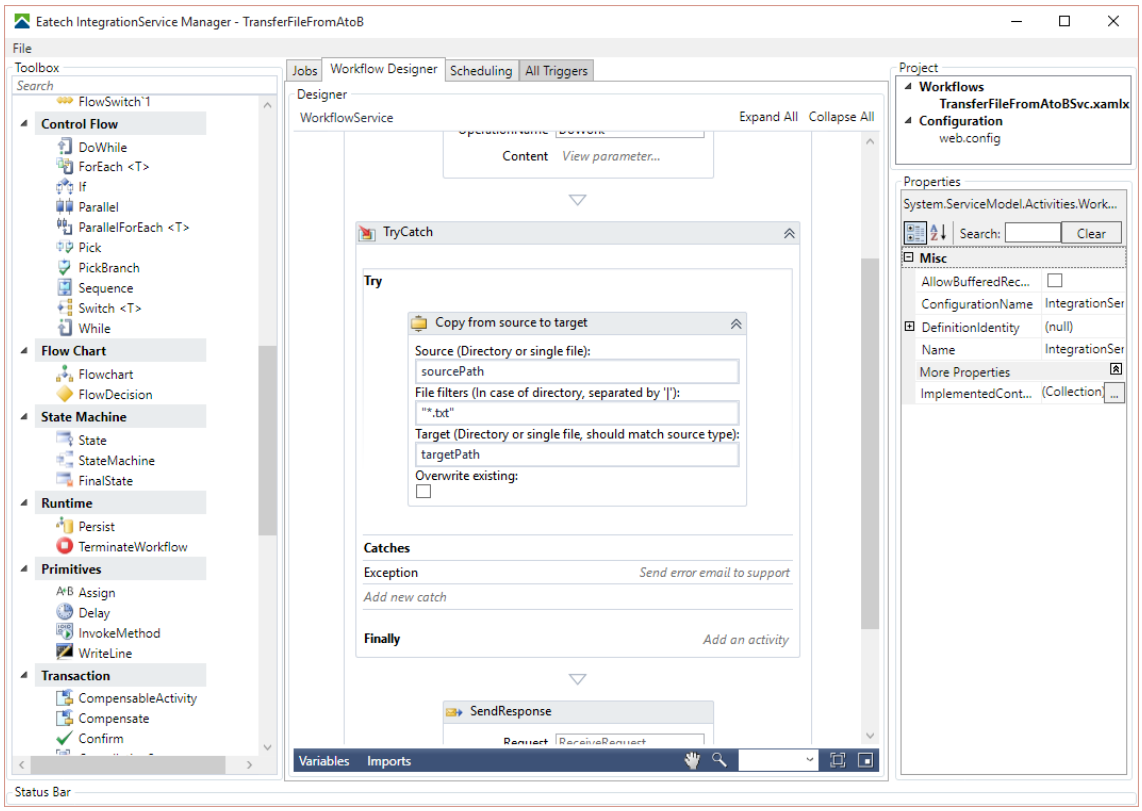
Jos tehtävä luodaan tyhjästä, niin tällöin käyttäjän tulee valita malli tehtävälle. Mallit ovat WF -projekteja jotka sisältävät pohjan työnkulun määrittelylle, käytettävät aktiviteetit ja konfiguraation. Käyttäjällä on valittavissa yhdenlainen malli tehtävälle. Tämä on WF -palvelu, joka sisältää tehtävän käynnistävän palvelupyynnön *ReceiveRequest* -aktiviteetissa ja tähän vastauksen *SendResponse* -aktiviteetissa, joihin käyttäjän ei tarvitse koskea. *ReceiveRequest* -aktiviteetti voisi vastaanottaa myös parametreja ja *SendResponse* -akti-

viteetissa voisi määrittää millainen vastaus kutsujalle lähetetään, mutta näitä ei tässä järjestelmässä käytetä tällä hetkellä. Varsinainen tehtävä sijoitetaan pyynnön ja vastauksen väliin. Kuvassa 11 on esitetty uuden tehtävän luomisen sekvenssikaavio. *”Tehtävän tallennus”* on osio, joka toistuu myös tehtävän editoinnin jälkeisessä tallennuksessa.



Kuva 11. Uuden tehtävän luomisen sekvenssikaavio.

Tehtävä voidaan myös tuoda olemassa olevasta, mahdollisesti muualla luodusta työnkulusta. Tällöin käyttäjän tulee asettaa polku olemassa olevan työnkulun sisältävään kansioon. Kuvassa 12 on näkymä käyttöliittymästä, jossa on avattuna ”pyyntö-vastaus”-työnkulkumalliin perustuva esimerkkitehtävä *”TransferFilesFromAtoB”* muokattavaksi. Käsiteltävän tehtävän nimi näkyy käyttöliittymän otsikossa ja tehtävä on tallennettavissa *”File”* -valikosta tai näppäinyhdistelmällä *”Ctrl+s”*. Tehtävän muokkausnäkymä sisältää kolme lohkoa, jotka ovat *”Toolbox”*, *”Designer”* ja *”Project”*. *”Toolbox”* sisältää käytettävissä olevat aktiviteetit, jotka ovat siirrettävissä työnkulkuun *”drag and drop”* -tyyppisesti. *”Designer”* sisältää työnkulun muokkauksen piirtoalueen. *”Project”* sisältää kyseisen tehtävän WF -projektitiedot, kuten konfiguraatiotiedoston. Näitä projektitietoja käyttäjän ei tarvitse normaalisti käsitellä ja ne voitaisiin mahdollisesti piilottaakin käyttäjältä.



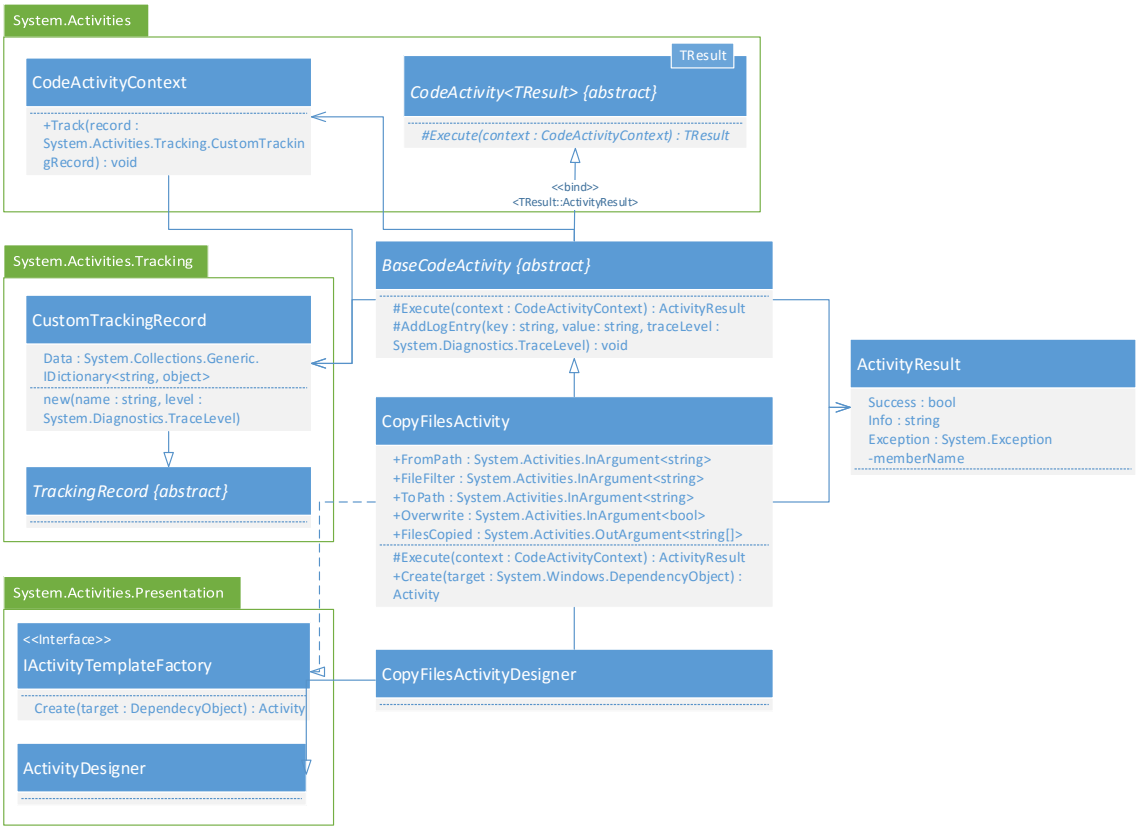
Kuva 12. Tehtävän muokkauksen näkymä.

Esimerkkitehtävä *”TransferFilesFromAtoB”* sisältää yhden *try-catch* -lohkon, jonka *try* -osassa suoritetaan itse tehty *CopyFiles* -aktiviteetti, ja siinä heitetyt poikkeukset otetaan kiinni *catches* -lohkossa. Poikkeuksista lähetetään sähköpostia ylläpitäjille itse tehdyllä *SendEmail* -aktiviteetilla. Minkä tahansa aktiviteetin näkyvää nimeä voidaan muuttaa, joten tässä *CopyFiles* -aktiviteetin nimi on muutettu muotoon *”Copy from source to target”* ja *SendEmail* -aktiviteetin nimi muotoon *”Send error email to support”*. *Catches* -lohkon sisältö on lisäksi kutistettu, joten siinä näkyy vain siinä suoritettavan aktiviteetin nimi. Lisäksi esimerkissä huomion arvoista on, että *CopyFiles* -aktiviteetin *Source* -parametrille annettu arvo *sourcePath* ja *Target* -parametrille annettu arvo *targetPath* ovat muuttujia, joiden oletusarvo näkyisi alaosan *Variables* -listauksessa ja niiden arvoa voitaisiin muuttaa *Assign* -aktiviteetilla kesken työnkulun suorituksen.

5.4.3 Aktiviteettien luominen

Kuvassa 13 on esitetty itse tehtyjen aktiviteettien luokkakaavio. Selvyyden vuoksi kuva esittää ainoastaan yksinkertaisen *CopyFiles* -aktiviteetin luomiseen käytetyt luokat. Aktiviteettia varten täytyy luoda kontrolleri (*CopyFilesActivity*) ja näkymä (*CopyFilesActivityDesigner*). Kontrollerissa määritellään, mitä tehdään, kun aktiviteetti ajetaan, ja näkymässä määritellään miltä aktiviteetti näyttää editorissa. Lisäksi nämä täytyy olla *kyt-*

kettynä (engl. *bind*) toisiinsa jotta editorissa lisätyt aktiviteetin kenttien arvot olisivat käytettävissä kontrollerissa. *CopyFiles* -aktiviteetin näkymä on nähtävissä ylempänä kuvassa 12.



Kuva 13. Aktiviteettien luokkakaavio.

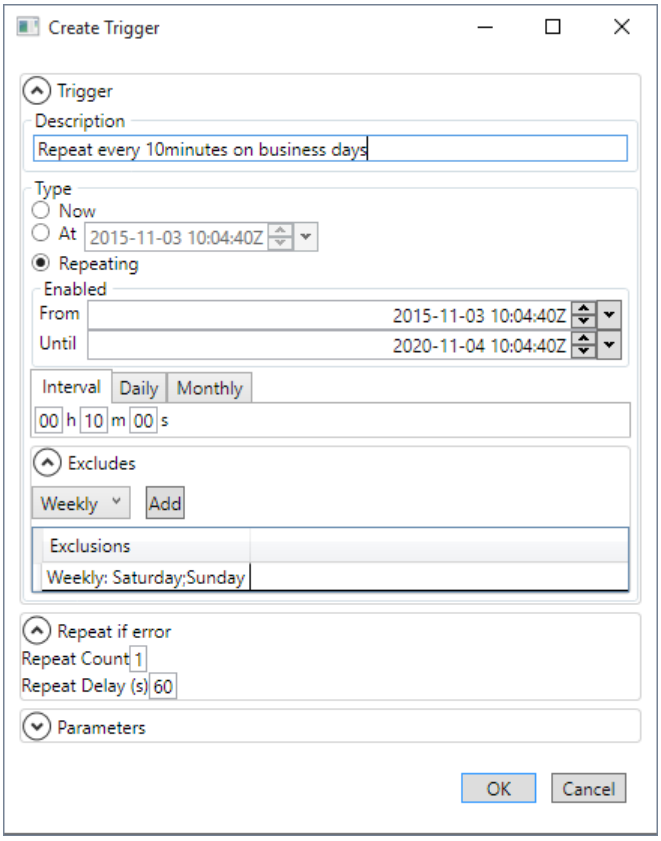
Kaikki itse tehdyt aktiviteetit, kuten *CopyFiles* -aktiviteetti, periytetään abstraktista *BaseCodeActivity* -luokasta. Tämä luokka taasen periytetään abstraktista ja geneerisestä *System.Activities.CodeActivity<TResult>* -luokasta käyttäen paluuarvona *ActivityResult* -luokkaa. Näin tehden kaikki itse tehdyt aktiviteetit luovat *ActivityResult* -instanssin *Execute()* -metodin paluuarvona, jota voidaan käyttää työnkulun editoinnissa. Tarkoituksena tällä on, ettei yksikään itse tehty aktiviteetti vuoda poikkeuksia, ja käyttäjä voi halutessaan tarkastaa aktiviteetin lopputuloksen *ActivityResult* -instanssin avulla. *BaseCodeActivity* -luokka sisältää lisäksi *AddLogEntry()* -metodin, jota varsinaiset aktiviteettien toteutukset voivat käyttää lokiviestien tallentamiseen. *AddLogEntry()* -metodissa lisätään uusi *CustomTrackingRecord()*, jonka järjestelmään liitetyt *TrackingParticipant* -toteutukset voivat vastaanottaa ja käsitellä. Lokimerkintöjä tehdään aktiviteeteissa aina kiinniotetuista poikkeuksista, mutta joskus myös muista mielenkiintoisista tapahtumista.

5.4.4 Ajastukset

Käyttöliittymässä voidaan katsoa listausta kaikista järjestelmään lisätyistä ajastuksista, tai sitten voidaan siirtyä yksittäisen tehtävän kautta katsomaan kaikkia siihen liitettyjä

ajastuksia. Ajastuslistauksessa voidaan lisätä uusia ajastuksia sekä muokata, poistaa, disabloida ja enabloida olemassa olevia ajastuksia. Ajastukset ovat tallessa vuorontaja -palvelussa, jossa ajastuksilla on nimi, selite, tyyppi, toisteisuus virheen sattuessa ja parametrit tehtävälle. Kaikkien näiden asetuksien muuttaminen on mahdollista käyttöliittymässä, pois lukien ajastuksen nimi. Nimi on uniikki tunniste ajastukselle vuorontajassa, joten se asetetaan käyttöliittymässä automaattisesti eikä sitä näytetä käyttäjälle.

Lisättäessä uusi ajastus tai muokattaessa vanhaa, aukeaa ajastuksen muokkaus dialogiin. Ajastus voi olla tyyppiä heti, tiettynä ajankohtana tai toistuvasti. Toistuville ajastuksille täytyy asettaa aikaväli, jolloin ne ovat käytössä (*”Enabled”*- lohko), toiston tyyppi (*”Interval”*, *”Daily”* tai *”Monthly”*) ja ajastuksesta pois suljetut aikavälit (*”Excludes”*), jolloin ajastuksen ei haluta laukeavan. Ajastuksesta voi sulkea pois aikavälejä, viikonpäiviä, kuukausittaisia päiviä tai vuosittaisia päivämääriä. Ajastuksen muokkauksen dialogi on esitetty kuvassa 14, jossa ajastus on muokattu sellaiseksi, että se on aktiivisena 3.11.2015 klo 10:04:40 – 4.11.2015 klo 10:04:40 välisen ajan, toistuu 10 minuutin välein, pois lukien viikonloput, ja virheen sattuessa tehtävän laukaisu toistetaan kerran minuutin päästä virheestä.

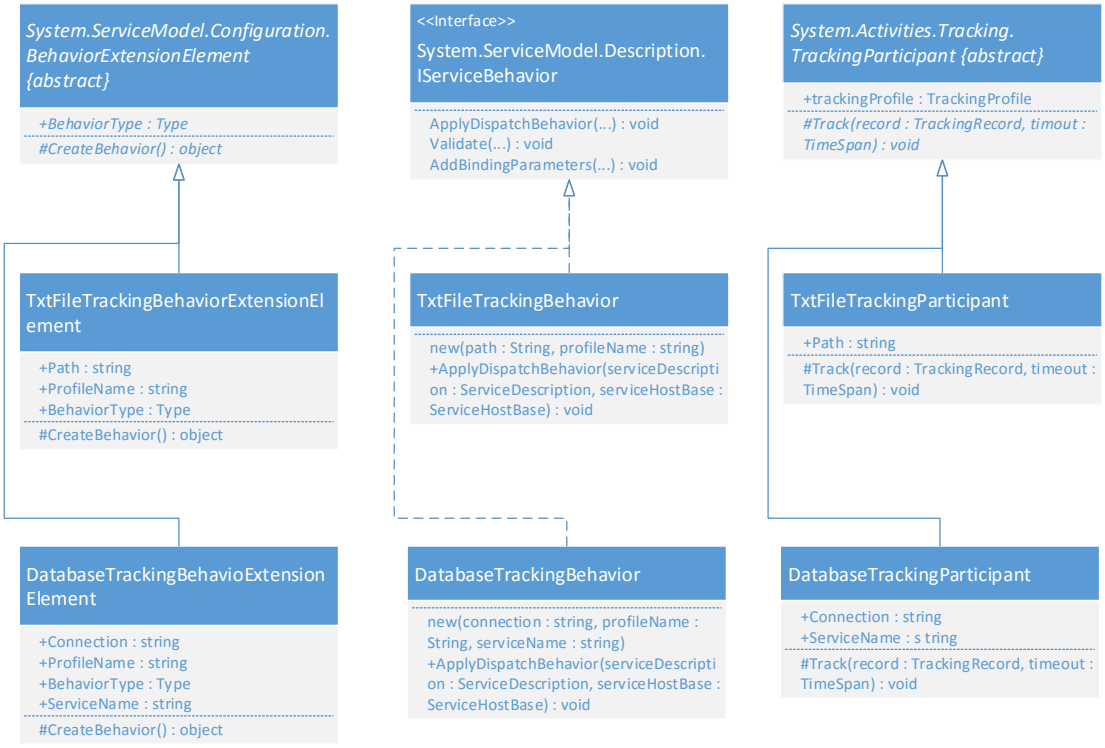


Kuva 14. Ajastuksen muokkaus.

5.4.5 Tehtävän toiminnan monitorointi

Järjestelmän yhtenä vaatimuksena oli, että tehtävien toimintaa voidaan monitoroida jollain tapaa. Referenssi -toteutuksessa oli toteutettuna suoritettujen WF -instanssien visualisointi perustuen AppFabric:n monitorointi -ominaisuuksiin, jota ei kuitenkaan tässä käytetään, vaan tehtäviä voidaan monitoroida keräämällä niiden toiminnasta lokia. Lokia kerätään tuotantokäytössä tietokantaan ja testatessa myös tiedostoon. Lokin kerääminen hoidetaan käyttäen WF Tracking -infrastruktuuria, johon myös AppFabric Monitoring perustuu.

Lokin keräämisä varten täytyi luoda *"TxtFileTracking"* ja *"DatabaseTracking"* -lokitoteutukset, joiden konfigurointi tapahtuu jokaisen tehtävän, eli WF -palvelun, konfiguraatiotiedostossa. Kuvassa 15 on esitetty näiden lokitoteutusten luokkakaaviot.



Kuva 15. WF Tracking -lokitoteutusten luokkakaavio.

Behavior ja *BehaviorExtensionElement* -luokilla mahdollistetaan lokitoteutusten konfigurointi. Lokiviestien julkaiseminen tapahtuu *TrackingParticipant* -luokissa. *DatabaseTrackingParticipant* tarvitsee toimiakseen käytettävän tietokannan yhteystiedot, jotka täytyy olla asetettuna konfiguraatiossa. *TxtFileTrackingParticipant* tarvitsee käytettävän tiedoston polun, joka täytyy olla asetettuna konfiguraatiossa.

5.4.6 Konfigurointi

Käyttöliittymän konfigurointi tapahtuu yhden konfiguraatiotiedoston avulla. Konfiguroinnissa täytyy asettaa vuorontaja -palvelun osoite, IIS -palvelimen osoite, Web deploy -palvelimen osoite sekä siinä käytettävä käyttäjätunnus-salasana-pari, oletushakemisto tehtävien tallentamista varten sekä tehtävä-mallien hakemisto. Lisäksi täytyy konfiguroida lokin keräämiseen käytettävät komponentit Common.Logging ja log4net, kuten vuorontajankin konfiguraatiossa. Käyttöliittymässä kerätään ainoastaan virhetasoiset lokiviestit.

6. ARVIOINTI

Tässä luvussa arvioidaan järjestelmän määrittelyn ja toteutuksen onnistumista. Aluksi esitellään asiat, joissa selvästi onnistuttiin, sekä ongelmakohdat, sitten kokonaisarvio ja lopuksi jatkokehitys-ideat.

6.1 Onnistumiset

Neljä asiaa onnistuivat erityisen hyvin tämän järjestelmän kehittämisessä: Määrittely, tekniikoiden valinta sekä käyttöliittymän toteutus käyttäen pohjana Microsoft:n referenssitoteutusta WF -editorista.

Määrittely onnistui kokonaisuudessaan hyvin, sillä sen pohjalta saatiin luotua järjestelmä, joka ei ainoastaan korjannut tämän yksittäisen tapauksen ongelmia, vaan siitä tuli suhteellisen yleiskäyttöinen vastaavia tapauksia varten. Määrittelyssä ei tehty mitään sellaisia huonoja ratkaisuja, jotka olisi toteutusvaiheessa täytynyt tehdä jollain huomattavan eri tavalla.

Tekniikoiden valinnassa suosittiin paljon valmiita komponentteja ja ne kaikki saatiin toimimaan ilman suurempia ongelmia. Valitut ohjelmistokirjastot ja palvelut nopeuttivat ja helpottivat kehitystä huomattavasti sekä tekivät käyttöliittymän ja taustapalvelun rakenteista selkeitä ja ylläpidettäviä. Käytettyjä palveluita olivat IIS/AppFabric sekä Web deploy. Käytettyjä komponentteja olivat .NET -sovelluskehyksen WCF-, WPF- ja WF -tekniikoiden lisäksi lokin keräämiseen, tehtävien vuoronnukseen sekä taustapalvelun asentamiseksi Windows -palveluksi käytetyt kolmannen osapuolen ohjelmakirjastot.

Käyttöliittymän pohjana käytettiin Microsoft:n referenssitoteutusta WF- editorista. Tämä referenssitoteutus sisälsi valmiiksi ominaisuudet WF- tehtävien graafiseen luomiseen, muokkaamiseen ja niiden siirtämiseen ajettavaksi AppFabric:ssa Web deployn avulla. Ilman tätä referenssitoteutusta käyttöliittymän luomiseen olisi mennyt huomattavasti enemmän aikaa. Referenssitoteutus sisälsi jopa yllättäen liiankin paljon ominaisuuksia, joita karsittiin pois toteutuksessa, mutta ne voidaan ottaa nopeasti uudelleen käyttöön tarpeen vaatiessa.

6.2 Ongelmakohdat

Kolme asiaa voidaan katsoa olleen hieman ongelmallisia tai onnistuneen huonosti tämän järjestelmän määrittelyssä ja toteutuksessa: Taustapalvelun palvelurajapinnan määrittely, käyttöliittymän ulkoasun hiominen sekä AppFabric Monitoring -ominaisuuden käyttämättä jättäminen tehtävien suoritushistorian näyttämiseksi käyttöliittymässä.

Määrittelyvaiheessa ei lyöty lukkoon taustapalvelun webservice- rajapintaa, jota käyttöliittymä käytti tietojen noutamiseen ja tallentamiseen. Tätä rajapintaa jouduttiin sitten toteutusvaiheessa muokkaamaan useaan otteeseen, kun huomattiin ettei sillä saada hoidettua kaikkia tarvittavia toimintoja. Jos määrittelyvaiheessa olisi uhrattu enemmän aikaa tuon rajapinnan suunnitteluun, niin toteutuksessa olisi saatu keskityttyä paremmin varsinaisten toimintojen toteuttamiseen. Ohjelmistojen suunnittelussa täytyy lähes aina tasapainotella määrittelyyn ja toteutukseen käytettävien resurssien suhteen. Johtuen tämän järjestelmän pienehköstä koosta, ei määrittelyvaiheeseen haluttu käyttää paljoa aikaa.

Käyttöliittymän ulkoasua ei käytännössä tyylitelty millään tavalla. Tästä syystä näkymät jäivät suhteellisen karun näköiseksi. Lisäksi näytettävä data muotoillaan tietyissä elementeissä käyttäjän kannalta epäselvästi. Esimerkki tällaisesta huonosta muotoilusta löytyy kuvasta 14, jossa on näkymä ajastuslistaukseen. Tuossa listauksessa ajastuksen kuvaus esitetään merkkijonona, joka ei kerro käyttäjälle välttämättä mitään ajastuksesta. Parempi vaihtoehto olisi jopa jättää se kokonaan pois näkymästä. Lisäksi käyttöliittymässä käyttäjille näytetään tietoja, joita heidän ei tarvitsisi nähdä, kuten WF -palvelun konfiguraatiotiedosto tehtävän muokkauksessa. Näitä tyyliseikkoja varten tarvitaan kuitenkin ensin käyttäjiltä palautetta, ennen kuin niitä kannattaa lähteä korjaamaan. Palautetta ei ole vielä kertynyt, sillä järjestelmä on ollut vasta lyhyen aikaa käytössä.

AppFabric Monitoring -ominaisuus tallentaisi WF -palveluiden toiminnasta dataa AppFabricin tietokantaan WF Tracking -infrastuktuurilla. Tätä dataa oltaisiin voitu käyttää tehtävien suoritushistorian näyttämiseen käyttöliittymässä, kuten oli käyttöliittymän referenssitoteutuksessa tehty. Tämä ominaisuus otettiin kuitenkin pois käytöstä, sillä sen konfiguraatiossa koettiin olevan joitain ongelmia, eikä korjaamiseen haluttu käyttää aikaa. Samaa WF Tracking -infrastruktuuria käytettiin sitten tehtävien lokin keräämiseen tiedostoon ja tietokantaan, mutta tätä dataa ei näytetty käyttöliittymässä. Koska AppFabric Monitoring -ominaisuutta ei käytetty, niin tehtävien suoritushistoriat eivät näy tällä hetkellä käyttäjälle millään tavalla. Vaikka tämä olikin määritelty lisäominaisuudeksi, niin sen toteuttamatta jättäminen laskee järjestelmän käytettävyyttä

6.3 Kokonaisarvio

Tämän työn tarkoituksena oli korvata asiakkaan vanha järjestelmä, jolla hoidettiin ylläpidollisia toimia heidän tuotantojärjestelmässään. Työ lähti liikkeelle ongelmien analysoinnilla ja uuden järjestelmän määrittelyllä, jonka jälkeen siirryttiin toteuttamaan määrittelyn mukaista järjestelmää. Määrittelyssä esitetyt vaaditut ominaisuudet saatiin toteutettua ja järjestelmä otettiin käyttöön onnistuneesti sen tilanneella asiakkaalla. Vaikka edellä mainittiinkin muutama ongelma järjestelmän määrittelyssä ja toteutuksessa, niin silti on helppo arvioida työn kokonaisuudessaan onnistuneen.

6.4 Jatkokehitysideoita

Kaikki lisäominaisuudet ovat luonnollisesti hyviä jatkokehitysideoita tälle järjestelmälle. Lisäominaisuuksia olivat suoritettujen tehtävien historian esittäminen käyttöliittymässä, tietoliikenne-yhteyksien salaaminen sekä käyttäjien todennus, valtuutus ja tilastointi käyttöliittymässä. Myös käyttöliittymän ulkoasun ja käytettävyyden parantelua kannattaa harkita. Lisäksi, jos järjestelmän kysyntä kasvaa, järjestelmälle kannattaa luoda jonkinlainen lisensointi-toiminto sekä asennusohjelma, jotka helpottavat hallinnointia. Kaikki nämä jatkokehitysideat parantaisivat järjestelmän käytettävyyttä, sekä uudelleenkäytettävyyttä mahdollisten uusien asiakkaiden tuotantoympäristöissä.

Tehtävien historian näyttäminen käyttöliittymässä on näistä jatkokehitysideoista nopein toteuttaa ja hyödyltään suurin. Tämän ominaisuuden ongelmat AppFabric Monitoring -ominaisuuteen perustuen esiteltiin kohdassa 6.2. Vaikka tehtävien historian näyttäminen voitaisiin toteuttaa nykyiseen tehtävien toiminnan lokin keräämiseen perustuen, niin luultavasti AppFabric Monitoring -ominaisuudella historian näyttäminen käyttöliittymässä olisi lopputulokseltaan parempi ja nopeammin toteutettava ratkaisu, sillä käyttöliittymän referenssitoteutukseen tämä ominaisuus oli jo kokonaisuudessaan sisällytetty.

Tietoliikenneyhteyksien salaaminen määriteltiin toteutettavan TLS -protokollalla. TLS -yhteyden avaamista varten vähintään palvelimena toimiva palvelu täytyy pystyä todentamaan. Tietoliikenteen osapuolien todennukseen määriteltiin käytettävän PKI -sertifikaatteja. TLS -protokolla on lisättävissä käytettäväksi WCF- ja WF -palveluihin suoraviivaisesti, kunhan palvelimen PKI -sertifikaatti on olemassa.

Käyttäjien todennus, valtuutus ja tilastointi käyttöliittymässä vaatii uusien tietojen tallentamista tietokantaan. Tarvittava tietokannan rakenne täytyy määritellä ennen toteutusta. Tietokantaan täytyy pystyä tallentamaan ainakin käyttäjätiedot, käyttäjäryhmätiedot, käyttäjäryhmien oikeudet sekä käyttäjien tekemät toiminnot. Käyttäjiin liittyvien tietojen tallennus ja lataus voidaan hoitaa suoralla yhteydellä tietokantaan käyttöliittymästä tai taustapalvelun kautta. Jos käytetään taustapalvelua tiedon välitykseen, niin muutoksia täytyisi tehdä tietokannan ja käyttöliittymään lisäksi myös taustapalveluun. Toisaalta käyttöliittymästä voitaisiin tällöin käyttää jo olemassa olevaa palvelua näiden käyttäjätietojenkin hakemiseen. Jos käytetään suoraa yhteyttä tietokantaan käyttöliittymästä, niin muutoksia ei tarvitsisi tehdä taustapalveluun. Myös lokin kerääminen käyttöliittymästä tietokantaan käyttää suoraa yhteyttä, joten ei liene mitään syytä, miksi suoraa yhteyttä ei voitaisi käyttää. Suora yhteys on tässä tapauksessa parempi ratkaisu yksinkertaisemman toteutuksen vuoksi.

Käyttöliittymän ulkoasun epäkohdat esiteltiin kohdassa 6.2. Ainakin nuo kyseiset epäkohdat olisi syytä korjata, mutta muitakin korjattavia tyylillisiä asioita käyttöliittymässä

voi olla. Tyyliasioihin vaikuttavat paljon järjestelmän tilaavan tahon omat tyylimieltymykset, ja ne voivat vaihdella paljon tapauskohtaisesti. Tästä syystä ne täytyy määrittää aina kyseessä olevan tahon kanssa.

Järjestelmän lisensointi- toiminto estäisi sen käytön, jos käyttölisenssiä ei ole hankittu tai se vanhenee. Tämä estäisi järjestelmän luvattoman käytön. Käytännössä lisenssi- tietoja täytyisi pitää tallessa jossain palvelussa, josta tämän järjestelmän taustapalvelu voisi esimerkiksi päivittäin kysyä, onko lisenssi toiminnassa. Jos lisenssi ei ole aktiivinen, niin tehtäviä ei vuoronneta eikä käyttöliittymää voisi käyttää. Käyttäjiä olisi hyvä informoida lisenssin vanhentumisesta esimerkiksi kuukausi ennen vanhentumista käyttöliittymässä sekä sähköpostein.

Asennusohjelma helpottaisi järjestelmän asentamista, päivittämistä ja poistamista. Asennettavia komponentteja ovat ainakin .NET -ohjelmistokehys, käyttöliittymä, taustapalvelu, IIS, AppFabric, Web Deploy, Quartz.NET:n käyttämä tietokanta ja lokin keräämiseen käytettävä tietokanta. Asennuspaketissa tulisi ottaa huomioon se, että kaikki järjestelmän komponentit saattavat sijaita eri palvelimilla. Johtuen asennusohjelman monimutkaisuudesta, se kannattaa toteuttaa vain, jos järjestelmälle syntyy paljon kysyntää. Asennusohjelma voidaan toteuttaa esimerkiksi Microsoft Visual Studio 2015 Installer Projects -projektina. Kyseinen Visual Studio -projektityyppi on liitännäinen, joka löytyy osoitteesta https://visualstudiogallery.msdn.microsoft.com/f1cc3f3e-c300-40a7-8797-c509fb8933b9

7. YHTEENVETO

Asiakkaalla oli suljetussa sisäverkossa toimivassa tuotantojärjestelmässään ajoitettuja tehtäviä, jotka hoitivat esimerkiksi raporttien siirtämiseksi kolmansille osapuolille. Näitä tehtäviä varten ei ollut kuitenkaan olemassa minkäänlaista hallintatyökalua, joten järjestelmän ylläpitäjät olivat saaneet luoda jokaisen tehtävän helpoimmaksi näkemällään tavalla, jolloin tehtävien määrän kasvettua alkoi muodostua ongelma niiden dokumentaatiosta ja ylläpidettävyydestä. Tämän sekavan vanhan järjestelyn tilalle haluttiin määritellä uusi sovellus, jolla näitä tehtäviä hallinnoitaisiin keskitetystä käyttöliittymästä.

Vanhan järjestelyn ongelmien kautta saatiin pohja uuden sovelluksen määrittelylle. Määrittelyn toiminnallisina vaatimuksina esitettiin käyttöliittymä, joka näyttäisi listauksen olemassa olevista tehtävistä ja mahdollistaisi tehtävien luomisen, muokkauksen ja poistamisen. Tehtäviä muokattaisiin graafisina työnkulkuina, joka olisi sopiva tapa järjestelmäylläpitäjille, jotka eivät ole varsinaisesti ohjelmoijia. Graafiset työnkulut myös dokumentoivat tehtävän toiminnan havainnollisesti ja ovat helppoja ylläpitää. Käyttöliittymässä täytyi lisäksi pystyä siirtymään tehtävästä siihen liitettyihin ajastuksiin ja näitä täytyi pystyä lisäämään, poistamaan ja muokkaamaan. Toiminnalliseksi lisäominaisuudeksi määritettään käyttäjien todennus, valtuutus ja tilastointi sekä tehtävien suoritushistorian näyttäminen ja yksittäisen suorituskerran yksityiskohtainen esittely, mahdollisesti graafisena suorituspuuna.

Ei-toiminnallisiksi ominaisuuksiksi määritettiin, että yksittäisestä tehtävästä saisi olla vain yksi instanssi suorituksessa kerrallaan, ajastuksilla on kovat reaaliaika-vaatimukset, tehtävien suoritusten etenemisestä tallennetaan monitoroitavaa dataa ja järjestelmän komponentit tallentavat lokiviestejä toiminnastaan. Ei-toiminnalliseksi lisäominaisuudeksi määritettiin, että järjestelmän komponenttien väliset tietoliikenneyhteydet salattaisiin ja palvelut tunnistettaisiin.

Järjestelmä toteutettiin Windows -ympäristöön .NET -sovelluskehyksellä. Käyttöliittymän pohjana käytettiin Microsoft:n referenssitoteutusta WF -suunnittelukäyttöliittymästä. Tehtävät toteutettiin WF -työnkulkuina, jotka muokattiin WF -palveluiksi suoritettavaksi IIS:n laajennokseen AppFabric:iin, josta ne ovat käynnistettävissä etäproseduuri -kutsuilla. WF -palveluiden siirto AppFabric:iin automatisoitiin Web Deploy -palvelun avulla. Tehtävien ajastusta varten toteutettiin Windows -palvelu, joka huolehti tehtävien käynnistyksestä käyttöliittymässä määritettyjen ajastusten perusteella. Käyttöliittymän ja taustapalvelun välinen kommunikointi toteutettiin etäproseduuri -kutsuilla. Taustapalvelun täytyi olla datan suhteen pysyvä, joten se tallensi kaiken tarvitsemansa datan tietokantaan. Vuoronnukseen käytettiin Quartz.NET -vuoronnuskomponenttia, joka hoiti pysyvyyden automaattisesti. Taustapalvelun toteutettiin sovelluksena, jonka asennus Windows -palveluksi mahdollistettiin Topshelf -komponentilla. Käyttöliittymän

ja taustapalvelun lokin kerääminen hoidettiin Common.Logging- ja log4net -komponenteilla. tehtävien monitorointi-datan tallentaminen toteutettiin WF Tracking -infrastruktuurilla.

Vaaditut ominaisuudet saatiin toteutettua ja järjestelmä saatiin vietyä tuotantoon onnistuneesti. Näin ollen kokonaisuutena projektin voidaan arvioida onnistuneen. Erityisen hyvin onnistuneiksi osioiksi katsottiin määrittely, josta ei jouduttu joustamaan toteutuksessa, käyttöliittymän toteutus, joka eteni suoraviivaisesti, sekä WF -palveluiden ajaminen AppFabric:ssa. Pienoisiksi ongelmakohdiksi nähtiin, ettei määrittelyssä otettu kantaa tarpeeksi käyttöliittymän ja taustapalvelun väliseen rajapintaan, jonka takia toteutusvaiheessa tuota rajapintaa jouduttiin päivittämään useaan otteeseen, käyttöliittymän ulkoasua ei tyylitelty, joten se jäi karuksi ja osittain hieman sekavaksi, eikä tehtävien historian näyttämistä käyttöliittymässä toteutettu, vaikka näin oli käyttöliittymän referenssitoteutuksessa tehty AppFabric Monitoring -ominaisuuteen perustuen.

Jatkokehitysideoiksi esitettiin lisäominaisuuksiksi määriteltyjä tehtävien historian esittämistä käyttöliittymässä, tietoliikenneyhteyksien salaamista sekä käyttäjien todennusta, valtuutusta ja tilastointia. Lisäksi käyttöliittymän ulkoasua ja käytettävyyttä kannattaisi kehittää, ja järjestelmälle olisi hyvä luoda lisensointi -ominaisuus sekä asennusohjelma. Nämä kaksi jälkimmäistä jatkokehitysideaa kannattaa toteuttaa vain, jos järjestelmän kysyntä kasvaa.

LÄHDELUETTELO

[1] Windows. Microsoft. [WWW] Viitattu 7.12.2015. Saatavissa: https://www.microsoft.com/en-us/windows

[2] .NET. Microsoft. [WWW] Viitattu 7.12.2015. Saatavissa: https://www.microsoft.com/net

[3] C# Programming Guide. Microsoft Developer Network. [WWW] Viitattu 7.12.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx

[4] A Developer's Introduction to Windows Workflow Foundation (WF) in .NET 4. Milner, Matt. [WWW] Viitattu 12.11.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/ee342461.aspx

[5] IIS. Microsoft. [WWW] Viitattu 8.10.2015. Saatavissa: https://www.iis.net/home

[6] Microsoft AppFabric 1.1 for Windows Server. Microsoft Developer Network. [WWW] Viitattu 8.10.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/hh351318.aspx

[7] Web Deploy. Microsoft. [WWW] Viitattu 31.10.2015. Saatavissa: http://www.iis.net/downloads/microsoft/web-deploy

[8] Lahma, Marko: Quartz Enterprise Scheduler .NET. [WWW] Viitattu 11.11.2015. Saatavissa: http://www.quartz-scheduler.net/

[9] common-logging. GitHub, Inc.. [WWW] Viitattu 10.11.2015. Saatavissa: https://github.com/net-commons/common-logging

[10] log4net. Apache Software Foundation. [WWW] Viitattu 10.11.2015. Saatavissa: http://logging.apache.org/log4net/

[11] Smith, Travis, et al.: Topshelf. [WWW] Viitattu 7.12.2015. Saatavissa: http://topshelf-project.com/

[12] Common Language Runtime (CLR). Microsoft Developer Network. [WWW] Viitattu 7.12.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/8bs2ecf4(v=vs.110).aspx

[13] Windows Communication Foundation. Microsoft Developer Network. [WWW] Viitattu 7.12.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/dd456779(v=vs.110).aspx

[14] Windows Presentation Foundation. Microsoft Developer Network. [WWW] Viitattu 7.12.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx

[15] Windows Server. Microsoft Developer Network. [WWW] Viitattu 7.12.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/desktop/dn636873(v=vs.85).aspx

[16] Microsoft SQL Server. Microsoft Developer Network. [WWW] Viitattu 7.12.2015. Saatavissa: https://msdn.microsoft.com/library/mt590198.aspx

[17] XAML Overview. Microsoft Developer Network. [WWW] Viitattu 7.12.2015. Saatavissa: https://msdn.microsoft.com/library/ms752059(v=vs.100).aspx

[18] Dynamic-Link Libraries. Microsoft Developer Network. [WWW] Viitattu 7.12.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/desktop/ms682589(v=vs.85).aspx

[19] Workflow Tracking and Tracing. Microsoft Developer Network. [WWW] Viitattu 12.11.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/ee513992(v=vs.110).aspx

[20] Workflow Hosting Options. Microsoft Developer Network. [WWW] Viitattu 12.11.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/jj635139(v=vs.110).aspx

[21] Hypertext Transfer Protocol HTTP/1.1. RFC-Base.org. [WWW] Viitattu 1.12.2015. Saatavissa: http://www.rfc-base.org/rfc-2616.html

[22] File Transfer Protocol. RFC-Base.org. [WWW] Viitattu 1.12.2015. Saatavissa: http://www.rfc-base.org/rfc-959.html

[23] Simple Mail Transfer Protocol. RFC-Base.org. [WWW] Viitattu 7.12.2015. Saatavissa: http://www.rfc-base.org/rfc-2821.html

[24] Network News Transfer Protocol. RFC-Base.org. [WWW] Viitattu 7.12.2015. Saatavissa: http://www.rfc-base.org/rfc-3977.html

[25] Microsoft Web Platform Installer 5.0. Microsoft web. [WWW] Viitattu 7.12.2015. Saatavissa: http://www.microsoft.com/web/downloads/platform.aspx

[26] Transmission Control Protocol. RFC-Base.org. [WWW] Viitattu 7.12.2015. Saatavissa: http://www.rfc-base.org/rfc-793.html

[27] Named Pipes. Microsoft Developer Network. [WWW] Viitattu 7.12.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/desktop/aa365590(v=vs.85).aspx

[28] Message Queuing (MSMQ). Microsoft Developer Network. [WWW] Viitattu 7.12.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/ms711472(v=vs.85).aspx

[29] Gosling, James, et al.: The Java Language Specification. [WWW] Viitattu 7.12.2015. Saatavissa: https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

[30] What is UNIX. The Open Group. [WWW] Viitattu 7.12.2015. Saatavissa: http://www.unix.org/what_is_unix.html

[31] NLog. NLog-project. [WWW] Viitattu 7.12.2015. Saatavissa: http://nlog-project.org/

[32] Internet Protocol. RFC-Base.org. [WWW] Viitattu 1.12.2015. Saatavissa: http://www.rfc-base.org/rfc-791.html

[33] Simple Object Access Protocol (SOAP) 1.1. W3C. [WWW] Viitattu 7.12.2015. Saatavissa: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[34] Perl Language reference guide. Harvard Mathematic Department. [WWW] Viitattu 1.12.2015. Saatavissa: http://www.math.harvard.edu/computing/perl/perl.pdf

[35] TSO/E REXX Reference. International Business Machines Corporation. [WWW] Viitattu 1.12.2015. Saatavissa: http://silk.nih.gov/dbtek/viewdoc?vdsn=VPOD.IKJ3A302.PDF

[36] Using batch files. Microsoft. [WWW] Viitattu 1.12.2015. Saatavissa: http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/batch.mspx?mfr=true

[37] Task Scheduler. Microsoft Developer Network. [WWW] Viitattu 1.12.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/aa383614.aspx

[38] The Transport Layer Security (TLS) Protocol Version 1.2. RFC Base. [WWW] Viitattu 7.12.2015. Saatavissa: http://www.rfc-base.org/rfc-5246.html

[39] Sorenson, Holt: An Introduction to OpenSSL, Part Three: PKI - Public Key Infrastructure. [WWW] Viitattu 8.10.2015. Saatavissa: http://www.symantec.com/connect/articles/introduction-openssl-part-three-pki-public-key-infrastructure

[40] How to: Run a Workflow. Microsoft Developer Network. [WWW] Viitattu 31.10.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/dd489463(VS.110).aspx

[41] Workflow Hosting Options. Microsoft Developer Network. [WWW] Viitattu 10.8.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/jj635139(v=vs.110).aspx

[42] Schou-Rode, Jørn: NCron - .NET scheduling service. [WWW] Viitattu 8.10.2015. Saatavissa: https://code.google.com/p/ncron/

[43] Task Scheduler Manager Wrapper. CodePlex. [WWW] Viitattu 8.10.2015. Saatavissa: https://taskscheduler.codeplex.com/

[44] Sending and Receiving Faults. Microsoft Developer Network. [WWW] Viitattu 14.10.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/ms732013(v=vs.110).aspx

[45] Debug.Assert Method. *Microsoft Developer Network*. [WWW] Viitattu 20.10.2015. Saatavissa: https://msdn.microsoft.com/en-us/library/e63efys0(v=vs.110).aspx

[46] Web Service Description Language (WSDL) 1.1. W3C. [WWW] Viitattu 3.12.2015. Saatavissa: http://www.w3.org/TR/wsdl