LAB University of Applied Sciences

Liiketoimintaprosessin automatisointi

LAB-ammattikorkeakoulu
Insinööri (YAMK)
2024
Tomi Koskinen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Tomi Koskinen	Opinnäytetyö, YAMK	2024
	Sivumäärä 47	

Työn nimi

Liiketoimintaprosessin automatisointi

Tutkinto ja koulutusala

Insinööri (YAMK), IoT:stä tekoälyyn

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

Keskinäinen Työeläkeyhtiö Varma

Tiivistelmä

Opinnäytetyön tavoitteena oli tutkia aiemmin paljon manuaalista työtä vaatineen liiketoimintaprosessin automatisointia käyttäen liiketoimintaprosessien hallintaan ja automaatioon tarkoitettua BPMN-kuvaustapaa. Suunnittelutieteellinen menetelmä toimi tutkimusmenetelmänä ja tapaustutkimus tutkimusstrategiana. Lisäksi haluttiin vertailla vaihtoehtoista ratkaisua, jossa BPMN:n sijaan sovelletaan tapausten hallinnan kuvaustapaa (CMMN). Lopuksi kokeiltiin päätösten hallinnan kuvaustavan (DMN) toiminnallisuutta ja yhteensopivuutta edellä mainittujen kuvaustapojen kanssa.

Prosessi saatiin automatisoitua onnistuneesti ja voitiin todeta automatisoinnista saatavan hyötyä monin tavoin. Vaihtoehtoinen ratkaisu CMMN:ää käyttäen vahvisti käsitystä siitä, että BPMN oli oikea valinta tässä kehitysprojektissa, mutta vaikutti myös toimivalta kuvaustavalta joissakin tilanteissa. Päätösten kuvaaminen DMN:llä taas vaikutti toimivan yhteen molempien kuvaustapojen kanssa. Ratkaisuja vertaamalla voidaan valita tulevaisuudessa erilaisiin tilanteisiin paras kuvaustapa.

Asiasanat

BPMN, CMMN, DMN, liiketoimintaprosessi, prosessiautomaatio

Abstract

Author(s)	Type of Publication	Published
Tomi Koskinen	Thesis, UAS	2024
	Number of Pages 47	

Title of Publication

The Automatization of a Business Process

Degree, Field of Study

Master of Engineering, From IoT to AI

Organisation of the client (if the thesis work is commissioned by another party)

Varma Mutual Pension Insurance Company

Abstract

The purpose of this thesis was to investigate the automation of a business process that previously required a lot of manual work using the BPMN standard for business process management and automation. Design science was used as the research method, while case study served as the research strategy. An alternative solution where BPMN is replaced with the case management modeling method (CMMN) was compared to BPMN solution. Additionally, functionality and compatibility of the decision management modeling method (DMN) was evaluated with the aforementioned modeling methods.

The process was successfully automated, and it was found that automation provided various benefits. The alternative solution (CMMN) reinforced the belief that BPMN was the right choice for this development project, but it also proved to be a viable modeling method in some situations. Describing decisions with DMN, on the other hand, seemed to work well with both modeling methods. By comparing the solutions, the best modeling method can be chosen for different situations in the future.

Keywords

BPMN, CMMN, DMN, business process, process automation

Sisällys

1 Johdanto ... 1
1.1 Työn tausta ja tavoitteet ... 1
1.2 Tutkimuskysymykset ja rajaus ... 1
1.3 Tutkimusmenetelmä ja -strategia ... 2
2 BPM, BPA ja BPMN ... 5
3 CMMN-kuvauskieli tapausten hallintaan ... 8
4 DMN-standardi ja päätösmoottorit automaattiseen päätöstentekoon ... 10
5 Liiketoimintaprosessin automatisointi ... 11
5.1 Teknisen alustan osaset ... 11
5.2 Vaiheet ... 11
5.2.1 Aikataulu ... 11
5.2.2 Esivalmisteluja ... 12
5.2.3 Kehitysympäristön valmistelu ... 15
5.2.4 Testiympäristön valmistelu ... 19
5.2.5 Tuotantoympäristön valmistelu ... 21
5.2.6 Prosessikaavion kehittäminen ... 21
5.3 Vaihtoehtona tapausten hallinta ... 30
5.4 Päätösmoottorin hyödyntäminen ... 35
6 Yhteenveto ... 37
7 Pohdinta ... 39
7.1 Haasteet ... 39
7.2 Jatkokehitys ... 41
Lähteet ... 43

Lyhenteet ja käsitteet

Microsoft Azure on Microsoft Corporationin pilvialusta, jonka tarjontaan kuuluu satoja erilaisia palveluja. Palvelut ovat saatavilla tarpeen mukaan Microsoftin konesaleissa ympäri maailman ja pilvipalvelujen lisäksi Azure tarjoaa myös muun tyyppisiä palveluja, esimerkiksi reunalaskentaa. Palvelut eivät rajoitu pelkästään Microsoftin omaan tarjontaan, vaan Azuren palveluissa voi suorittaa haluamiaan ohjelmistoja esimerkiksi konteissa. Palveluista laskutetaan yleensä käytön mukaan. (Microsoft c; Microsoft b.)

Azure Container Instances on pilvipalvelu, joka mahdollistaa yksittäisten sovelluskonttien suorittamisen Azure-pilvipalvelussa. Container Instances tukee sekä Linux- että Windows -käyttöjärjestelmiin perustuvia kontteja, jotka nopeuttavat sovellusten käyttöönottoa virtuaalikoneisiin verrattuna. Konttien hallinta on osin manuaalista, sillä esimerkiksi konttien määrän automaattista kasvattamista, eli skaalausta, tai automaattisia koordinoituja päivityksiä ei tueta vaan ne ovat Azure Kubernetes Servicen ominaisuuksia. (Microsoft 2024d.)

Azure DevOps on kokoelma erilaisia pilvipalveluja ohjelmistokehitysprojektien tarpeisiin. Se sisältää projektinhallintatyökaluja, versionhallinnan, asennus- ja julkaisuputket, testaustyökaluja ja varaston käännetyille sovelluspaketeille. Palveluja voi ottaa käyttöön tarpeen mukaan ja DevOpsiin voi myös yhdistää muita ulkoisia palveluja kuten esimerkiksi viestintäpalvelu Slackin. (Microsoft 2024e.)

Azure Key Vault on pilvipalvelu, jonka tarkoituksena on tarjota säilytyspaikka ja keino rajata pääsyä arkaluontoiseen tietoon. Tällaisia tietoja voivat olla esimerkiksi salasanat tai pääsyavaimet. (Microsoft 2024a.)

Azure SQL Database nimellä kutsutaan Microsoftin SQL Server -tietokantamoottoriin perustuvaa pilvipalvelua. Tietokanta on Microsoftin hallinnoima ja sille luvataan korkeaa saatavuutta, suorituskykyä ja SQL Serverin uusimmat päivitykset. Azure SQL Database tarjoaa myös työkaluja tietokannan valvontaan sekä tiedon varmistukseen muun muassa automaattisilla varmuuskopioilla ja tallentamalla tietoja suoritetuista tietokantaoperaatioista. (Microsoft. 2024f; Microsoft. 2024b.)

Azure Storage account on pilvitallennustila. Storage account on tarkoitettu sekä yleiskäyttöiseen tiedostojen tallentamiseen että datamassojen tallentamiseen. Storage accountin sisältämiin tiedostoihin pääsee käsiksi sen tarjoaman rajapinnan kautta. (Microsoft 2024c.)

Docker Compose on Docker Incorporatedin luoma useiden konttien hallintaan tarkoitettu sovellus. Sovellusympäristön kuvaamiseen Docker Composella käytetään YAML-muotoista Compose-tiedostoa. Tiedostoon voi kirjata muun muassa palveluja, eli kontteja, verkkoasetuksia ja loogisia levyjä. Docker Composen komentorivityökalulla voi hallita tiedostoon kuvattua ympäristöä ja sen osasia. (Docker a; Docker c.)

Dockerfile on Docker-komentorivityökalulle tarkoitettu ohjetiedosto. Se sisältää komentoja, jotka kertovat mistä osista kontti koostetaan. Yleensä Dockerfilen alussa uuden imagen pohjaksi otetaan jokin toinen image. (Docker b.)

GitHub on sovelluskehitykseen tarkoitettu verkkopalvelu, jonka tarjontaan kuuluu versionhallinta, automatisoidut testaus- ja julkaisuputket, salaisuuksien hallinta sekä projektihallintatyökalut. (GitHub.)

Image on muuttumaton paketti, jonka pohjalta kontti käynnistetään. Image sisältää tiedostoja, asetuksia, sovelluskoodin ja tarvittavat kirjastot sekä tiedostojärjestelmän muutoshistorian kerroksina. (Docker d; Docker e.)

Java KeyStore on Java-ohjelmointikielellä kirjoitettujen ohjelmistojen käyttämä säilö, johon voidaan tallentaa valtuutussertifikaatteja tai julkisen avaimen sertifikaatteja. (Anicas 2014.)

Java Keytool on hallintatyökalu avainparien ja sertifikaattien luomiseen ja käsittelyyn. (Oracle.)

JSON on JavaScript-ohjelmointikieleen perustuva merkintätapa, joka koostuu avain-arvopareista. JSON tukee useita datatyyppejä ja sitä käytetään tiedon tallentamiseen tai siirtämiseen sovellusten välillä, sillä monet ohjelmointikielet tukevat sitä. (Introducing JSON.; Mozilla 2024a.)

Kontti tai *säilö* eli *container* sisältää sovelluksen ja sen tarvitsemat sovelluskirjastot. Kontti on yksi imagea pohjana käyttävä instanssi. Erona virtuaalikoneisiin, kontti ei virtualisoi laitteistoa ja siten useampi kontti käyttää yhteistä laitteistoa ja käyttöjärjestelmän ydintä. Kontit suoritetaan kuitenkin eristyksissä toisistaan. (Docker d.)

REST on tietojärjestelmäarkkitehtuurityyli, jonka mukaan palvelin ja asiakas ovat toisistaan riippumattomia, jolloin toisen osapuolen tekninen toteutus voi taustalla muuttua vaikuttamatta toiseen. Niiden välinen kommunikointi on tilatonta, jolloin palvelimen ei tarvitse pitää kirjaa asiakkaan pyynnöistä eli tilasta. Vuorovaikutus asiakkaan ja palvelimen välillä perustuu resursseihin, joihin vaikutetaan HTTP-standardin metodeilla. (Codecademy.)

Sertifikaatti eli *varmenne* on jonkin osapuolen allekirjoittama todistus siitä, että jonkin toisen osapuolen julkinen avain on luotettava. Sertifikaatteja käytetään varmistamaan datan koskemattomuus ja luotettavuus. (Oracle.)

Windows Subsystem for Linux on Microsoft Windows-käyttöjärjestelmän ominaisuus, joka mahdollistaa Linux-käyttöjärjestelmän suorittamisen virtuaalikoneessa ja Linux-jakelun suorittamisen kontissa Windows-käyttöjärjestelmän sisällä. (Microsoft 2023.)

XML on merkintäkieli, jossa ei ole valmiiksi määriteltyjä elementtejä eli tageja vaan käyttäjä voi määritellä ne itse. XML-merkintäkieliset dokumentit koostuvat tagien sisältämistä tietokentistä. (Mozilla 2024b.)

YAML on merkintäkieli, jonka tavoitteena on olla ihmisen ymmärrettävissä ja helposti luettavissa. Sen käyttökohteena on yleensä sovelluksen asetusten säilöminen tai tiedon siirtäminen. YAML-muotoinen tiedosto rakentuu avain-arvopareista ja listauksista. (Goodwin & Khan 2023.)

1 Johdanto

1.1 Työn tausta ja tavoitteet

Toimeksiantajan organisaatiossa on lukuisia liiketoiminnallisia prosesseja. Prosessit ovat muotoutuneet ajan kuluessa esimerkiksi henkilöstön vaihtuessa, uusien työkalujen käyttöönoton myötä tai ulkoisten tekijöiden, esimerkiksi lakimuutosten, johdosta. Prosessien automatisointiin on käytetty vaihtelevia ratkaisuja, jotka voivat toisaalta olla pienimuotoisia yhden teknisen järjestelmän omilla välineillä luotuja toimintosarjoja ja nykyisin myös järjestelmien välisiä integraatioita. Liiketoiminnallisia prosesseja on kuvattu ja visualisoitu myös organisaation eri tasoilla, mutta kuvaukset ovat yleensä staattisia esityksiä, piirroksia tai tekstipohjaisia kertomuksia.

Toimeksiantajan tavoitteena on yleisellä tasolla saada lisää tietoa ja kokemuksia prosessiautomaation, tapaustenhallinnan ja päätöksenteon välineiden ominaisuuksista, eroavaisuuksista ja hyödyistä. Lisäksi tavoitteena on löytää sopiva ohjelmisto tai ohjelmistokokonaisuus, jota voidaan jatkossa käyttää prosessien automatisoinnissa, tapausten hallinnassa tai päätöstenteossa.

Tutkimuksen tapaus on liiketoimintaprosessi, jonka vaiheet oli määritetty ja teknisesti toteutettu jo vuonna 2023 pilviympäristön välineillä, joten vaiheiden määrittäminen ja ohjelmistologiikan toteuttaminen rajataan opinnäytetyön ulkopuolelle. Valmista ohjelmistologiikkaa käytetään kuitenkin hyödyksi myös opinnäytetyössä automatisoidun prosessin taustapalveluna rajapinnan kautta. Prosessia suoritetaan useita kertoja vuosittain tarpeen mukaan. Toimintosarja on suoritettu käynnistämällä yksittäisiä operaatioita manuaalisesti peräkkäin edellisen valmistuttua. Siten prosessin hallinta on vaatinut toistuvaa huomiota ja välivaiheiden tulosten tarkastelua.

1.2 Tutkimuskysymykset ja rajaus

Tässä opinnäytetyössä keskitytään selvittämään seuraavat asiat:

1. Millaisia hyötyjä prosessiautomaation työkaluista on prosessin automatisoinnissa?
2. Mitä tuotantokäyttöön soveltuvan prosessimoottorin teknisen ympäristön pystyttäminen vaatii?
3. Millaisia eroja BPMN-kuvaustavassa on vaihtoehtoiseen CMMN-kuvaustapaan verrattuna?

4. Mitä hyötyjä useisiin prosessiautomaatio-ohjelmistoihin sisältyvä DMN-kuvaustapaa hyödyntävä päätösten hallinta voisi tuoda automatisoitaviin prosesseihin?

Rajauksena työssä tutkittiin vain kahta eri prosessiautomaatioon käytettävää avoimen lähdekoodin ohjelmistoa niin aiemman kokemuksen ohjaamana, kuin myös helpon saatavuuden vuoksi. Lisäksi BPMN- ja CMMN-kuvaustapojen vertailussa ei nähty ajankäytön kannalta oleellisena rakentaa teknisesti täysin vastaavaa toimintaympäristöä molemmille ohjelmistoille, vaan keskityttiin vertailemaan kuvaustapojen eroja. Microsoftin Azure-pilviympäristössä täytyy tehdä palvelujen välisiä verkko- ja käyttöoikeusasetuksia, mutta nämä rajattiin pois opinnäytetyöstä muiden hoidettavaksi toimeksiantajan pilviympäristöä koskevien käytäntöjen mukaisesti.

1.3 Tutkimusmenetelmä ja -strategia

Opinnäytetyön tutkimusmenetelmänä käytettiin suunnittelutieteellistä menetelmää. Suunnittelutiede tutkii ja pyrkii luomaan artefakteja, jotka ratkaisevat käytännön ongelmia. Artefakti on jokin tutkijan luoma väline, fyysinen tai abstrakti, joka ratkaisee tutkittavan ongelman. Verrattuna empiiriseen tutkimukseen, suunnittelutieteellistä menetelmää käytettäessä ei siis pelkästään tutkita ja pohdita ympäristöä vaan myös suunnitellaan ja luodaan uutta. Suunnittelutieteellisessä kehitysprojektissa on olennaista tutkimuskysymysten asettelu, vaatimusten ja tavoitteiden määrittäminen sekä tulosten arviointi. Kehitysprojektin lopputuloksena on artefaktin lisäksi tietoa esimerkiksi sen ominaisuuksista, vaikutuksista ympäristöönsä ja parannuksia käytäntöihin. Käytännöt ovat ihmisten toteuttamia toistuvia toimintosarjoja, jonka vaiheet liittyvät toisiinsa. (Johannesson & Perjons 2021, 14–29.)

Tutkimusstrategiana opinnäytetyössä käytettiin tapaustutkimusta. Tapaustutkimus keskittyy syvällisesti tutkimaan yksittäistä tutkittavan ilmiön ilmentymää. Ennen tutkittavan ilmentymän valintaa, on valittava laajempi tutkimuksen kohteena oleva ilmiö tutkimuskysymysten pohjalta. Tapaustutkimuksen tarkoitus voi olla vähän tunnettua ilmiötä selventävä, tunnettua ilmiötä tarkasti kuvaileva tai syy-seuraussuhteita selittävä. Tutkimusstrategiana sitä käytetään, kun tutkittavaan tapaukseen vaikuttavat monet ympäröivät tekijät, joiden olemassaolo ja vaikutus on olennaista tutkimuksen onnistumiselle. Siksi tapaustutkimuksen kohdetta ei eristetä erilliseen tutkimusympäristöön, vaan siitä keskitytään keräämään laajasti tietoa eri näkökulmista useita keinoja käyttämällä ja myös sitä ympäröivistä tutkimukseen vaikuttavista tekijöistä. Tapaustutkimuksen tuloksia käsiteltäessä on huomioitava pätevätkö tulokset vain yksittäiseen tutkittuun ilmentymään vai ovatko ne yleistettävissä laajempaan joukkoon. (Johannesson & Perjons 2021, 46–48.)

<table>
<tr><td colspan="3">Practice
Describe the practice in which the problem exists, in particular its purpose, main activities and participants.</td></tr>
<tr><td>Problem
Describe the practical problem to be addressed. Formulate it in a precise and concise way. Explain why the problem is important and of general interest. Specify the stakeholders of the problem and how they are affected by it.</td><td rowspan="2">Research Process
Describe the research process of the project. State and justify the selection of research strategies and research methods. Possibly discuss which alternative research strategies and methods that were considered and why they were discarded. Clarify which of the main activities in design science (problem explication, requirements definition, design and development, and evaluation) that were included in the project. Describe how the research was carried out in the project. Discuss design rationale when applicable. Describe ethical issues encountered and how they were addressed.</td><td>Artefact
Specify the type of artefact. Describe the artefact, in particular, its structure, behaviour and functions. Describe how the artefact is to be used in the practice. Explain why and how the artefact can address the problem.</td></tr>
<tr><td>Requirements
Describe requirements on the artefact in a precise and concise way. Include functional as well as non-functional requirements. Justify the requirements by relating them to both the problem and the stakeholders.</td><td>Quality and Effects
Describe how well the artefact fulfils the requirements and to what extent it solves the practical problem. Describe the effects of using the artefact, including the side-effects. Discuss ethical and societal consequences of using the artefact.</td></tr>
<tr><td colspan="3">Knowledge Base
Describe the knowledge base that was used as a foundation for the project. The knowledge base may include theories and models as well as existing artefacts. Explain how the knowledge base was utilized in the research process.</td></tr>
</table>

Kuva 1. The Design Science Canvas (Johannesson & Perjons 2021, 85)

Opinnäytetyön suunnittelun pohjana käytettiin kuvan 1 mukaista Design Science Canvasta. Sen tarkoituksena on tuoda tapaustutkimuksen eri osa-alueet selkeästi ja nopeasti omaksuttavasti yhteen kaavioon. Suunnittelun apuna toimimisen lisäksi kaavioita voi käyttää tutkimuksen viestinnässä tai kehitysprojektin seurantatyökaluna. Kaavion osa-alueista käytäntö, jonka ongelmakohtiin opinnäytetyössä paneuduttiin, on kuvattu tämän työn osuudessa ”Työn tausta ja tavoitteet”. Vaatimukset on myös kuvattu tämän työn osuudessa ”Työn tausta ja tavoitteet”. Tutkimusprosessi on kuvattu tämän työn osuudessa ”Liiketoimintaprosessin automatisointi”. Artefakti on kuvattu tämän työn osuudessa ”Yhteenveto”. Laatua ja vaikutuksia kuvataan tämän työn osuudessa ”Pohdinta”. Tietopääoma taas on kuvattu tämän työn osuudessa 2, 3 ja 4 eli ”BPM, BPA ja BPMN”, ”CMMN tapausten hallintaan” sekä ” DMN-standardi ja päätösmoottorit automaattiseen päätöstentekoon”. (Johannesson & Perjons 2021, 84–86.)

2 BPM, BPA ja BPMN

Liiketoimintaprosessi on joukko tehtäviä ja tapahtumia, joiden myötä jokin liiketoiminnallinen tavoite täyttyy. Tavoite voi olla esimerkiksi verkkokauppaostoksen onnistuminen, kvartaaliraportin luonti yrityksen johtoryhmälle tai loma-anomuksen hyväksyntä. Useimmiten liiketoimintaprosessit sisältävät monia tehtäviä ja tapahtumia. Prosessin kokonaisuuteen voi sisältyä niin täysin koneellisesti tehtäviä, kuin myös ihmisen toimintaa vaativia osuuksia. (Red Hat 2022; IBM.)

Liiketoimintaprosessien hallintaa kutsutaan lyhenteellä Business Process Management (BPM). Se sisältää toistuvien prosessien havainnointia, analysointia ja visualisointia, mallinnusta ja suunnittelua, mittausta sekä optimointia. Prosesseja tarkastellaan ja kehitetään kokonaisuuksina yksittäisen tehtävän tarkastelun sijaan. Kehitystyö vaatii sekä liiketoimintaa, että teknisiä järjestelmiä tuntevien henkilöiden yhteistyötä. Liiketoimintaprosessien hallinnan tavoitteena on yleensä tehokkuuden parantaminen yksinkertaistamalla ja yhdenmukaistamalla prosesseja sekä ongelmakohtia poistamalla. BPM voi auttaa myös selkeyttämään kokonaisuuksia ja vastuualueita parantaen henkilöstön ja asiakkaiden tyytyväisyyttä. Liiketoimintaprosessien hallintaa voi edistää monilla tavoin esimerkiksi Lean-ajattelun keinoin. (Freund & Rücker 2019, 1; IBM.)

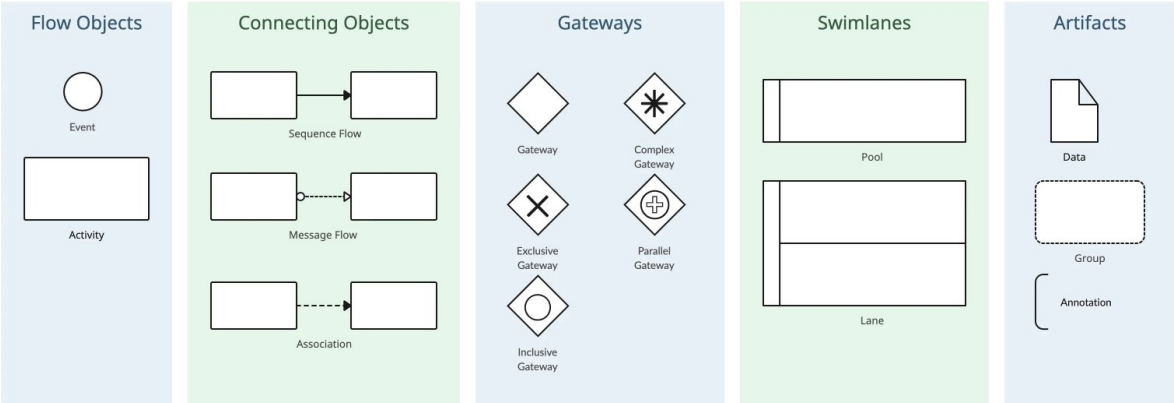
Yksittäisen liiketoimintaprosessin vaiheiden automatisoinnista käytetään termiä Business Process Automation (BPA). Se tarkoittaa yhden tai useamman tehtävän sisältävän prosessin toistettavaa koneellista suorittamista hyödyntäen tietoteknisiä järjestelmiä ja sovelluksia. Prosessit voivat sisältää myös ihmisen tekemää työtä sisältäviä tehtäviä. Liiketoimintaprosessien automaatiota voi hyödyntää osana liiketoimintaprosessien hallintaa, sillä niiden tavoitteet ovat yhteneviä. Sekä BPM että BPA tähtäävät tehokkuuden ja tuottavuuden kasvattamiseen. (Red Hat 2022; Mucci & Stryker 2024.)

Herzbergin ym. (2020) mukaan liiketoimintaprosessien automatisointi yleistyy organisaatioissa. Organisaatioiden, jotka ovat vähintään pilotoineet liiketoimintaprosessin automaatiota, osuus vastaajista nousi yhdeksällä prosenttiyksiköllä 67 %:iin vuosina 2018 ja 2020 toteutetun kyselytutkimuksen mukaan. Vastaavasti organisaatioiden, jotka eivät edes harkinneet liiketoimintaprosessien automaatiota, osuus laski neljällä prosenttiyksiköllä ollen 16 % vuonna 2020. Käytetyin automaatioteknologia oli liiketoimintaprosessien ja tapausten hallinnan automaation alustat.

Liiketoimintaprosessien kuvaamiseen ja automatisoimiseen on kehitetty standardoitu kuvaustapa, josta käytetään nimeä Business Process Model and Notation (BPMN). Alkujaan se on Business Process Management Initiativen kehittämä ja sittemmin Object

Management Groupin hallinnoima. Standardin tuorein laaja versio 2.0 on julkaistu vuonna 2011 ja kaksi vuotta myöhemmin myös kansainvälisenä ISO-standardina. BPMN:n tavoitteena on toimia yhdistävänä tekijänä liiketoiminnallisia päätöksiä tekevien sekä tietoteknisten asiantuntijoiden välillä molempien ymmärtämän visuaalisen kuvaustavan kautta. BPMN:n kohdeyleisöön kuuluvat niin liiketoimintaa ymmärtävät, että teknisestä toteutuksesta vastaavat henkilöt. Sitä voi hyödyntää sekä laajempien useita järjestelmiä ja osapuolia koskettavien prosessien ja kokonaisuuksien, että yksittäisten yhden järjestelmän sisäisten prosessien tai yksittäisten tehtävien kuvaamiseen. BPMN:ää voi käyttää sekä liiketoimintaprosessien hallinnan (BPM) että automatisoinnin (BPA) työkaluna. Prosessikaavioita voi myös julkaista prosessimoottorille suoritettavaksi, jolloin ohjelmisto huolehtii prosessin suorituksesta pois lukien ihmisen osallistumista vaativat tehtävät. BPMN:n kuvaustapa on tila- tai vuokaaviota muistuttava ja kaavioiden sisältämät elementit kuvaavat yleensä liiketoiminnalliseen prosessiin sisältyviä toimenpiteitä (Kaavio 1). Kuvassa 2 on kategorioittain jaoteltuna erilaisia prosesseihin sisältyviä elementtejä. Niitä ovat muun muassa aktiviteetit, jotka voivat olla esimerkiksi tehtäviä tai aliprosesseja, tapahtumat, risteykset, nuolet, uimaradat ja kaistat. Aktiviteetit kuvaavat jotakin, mitä täytyy tehdä, tapahtumat taas jotakin, mitä tapahtuu prosessin aikana. Risteykset ohjaavat prosessin suorituksen kulkua ja nuolet liittävät prosessin osasia yhteen kertoen suunnan. Uimaradat ja kaistat toimivat elementtien järjestelemisen apuna jakaen eri toimijoiden aktiviteetit omiin lokeroihinsa. Lisäksi prosesseissa voi olla dataobjekteja, viestejä ja tekstikuvauksia omina elementteinään ymmärryksen lisäämiseksi. (Object Management Group 2013; Freund & Rücker 2019, 1–12; Athuraliya 2023; Piirainen 2023; Belcic & Stryker 2024; Visual Paradigm a.)

Kaavio 1. Esimerkki BPMN-kaaviosta (Camunda a)



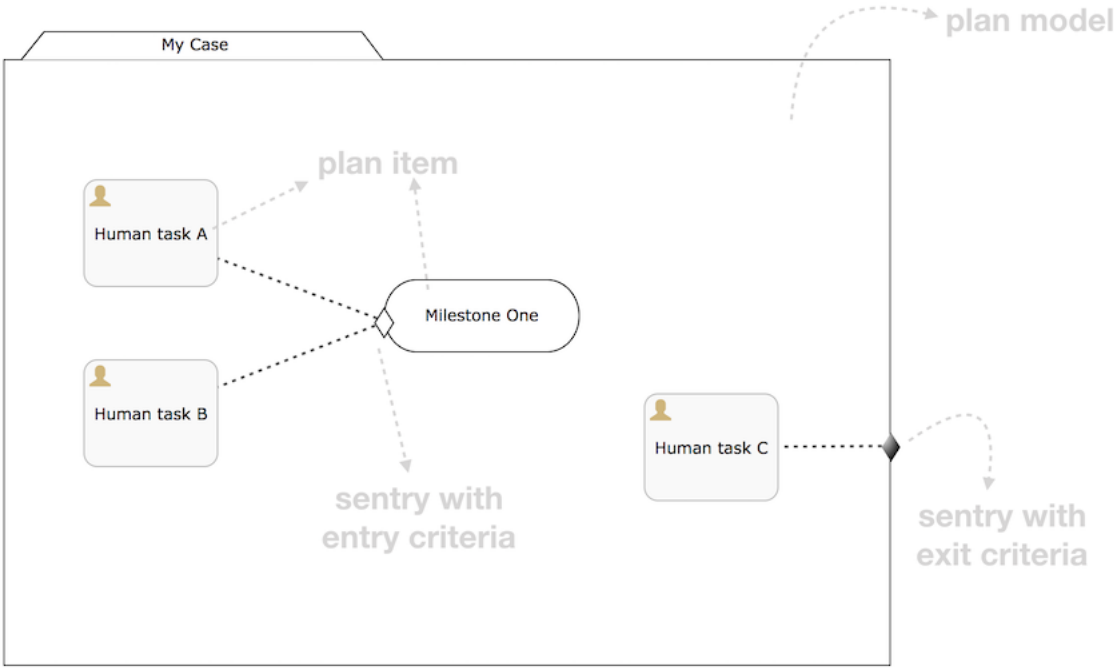
Kuva 2. BPMN-standardin sisältämiä elementtejä (Athuraliya 2024)

Erilaisia BPMN:ää hyödyntäviä sovelluksia ja palveluja on saatavilla monilta toimijoilta. Joissakin tuotteissa BPMN on pienessä roolissa kuten Microsoftin Visio -piirtotyökalussa, joka tukee kaavioiden piirtämistä, mutta työkalu ei tarjoa prosessimoottoria kaavioiden suorittamiseen. Muita tunnettuja yrityksiä, jotka tarjoavat BPMN-sovelluksia ovat muun muassa IBM, Red Hat ja Oracle. IBM:n Business Process Manager sisältää sekä mallinnustyökalut, prosessimoottorin, että valvonta- ja optimointivälineitä. Oracle tarjoaa vastaavilla ominaisuuksilla varustettua Business Process Management Suitea. Red Hatin tarjontaan kuului myös vastaavanlainen Process Automation Manager, joka tunnetaan nykyään IBM Process Automation Manager Open Editionina. Siinä on myös tuki DMN:lle ja CMMN:lle. Red Hatin valikoimaan kuuluu myös avoimen lähdekoodin Kogito ja jBPM. Pelkästään BPMN-prosessikaavioiden mallinnusta tarjoaa SAP Signavio. Pienempiä toimijoita ovat esimerkiksi Alfresco, Camunda ja Flowable, joiden kaikkien juuret ovat Alfrescon avoimen lähdekoodin Activiti-prosessimoottorissa. Nykyisin Alfrescon tarjontaan kuuluu Activitin lisäksi kaupallinen Process Services, joka sisältää BPMN-prosessien mallinnustyökalun, prosessimoottorin, optimointivälineitä ja tuen DMN:lle. Camunda tuottaa oman mallinnustyökalun lisäksi BPMN- ja DMN-standardeja tukevaa avoimen lähdekoodin prosessimoottoria sekä uudempaa kaupallista prosessimoottoria. Flowablen tuoreimmat graafiset mallinnus- ja optimointityökalut ovat kaupalliset, kun taas BPMN:ää, DMN:ää ja CMMN:ää tukeva prosessimoottori on avointa lähdekoodia. (Holmes-Higgin 2016; Meyer 2016; Alfresco 2017; IBM 2021; Red Hat 2021; Brown 2022; Camunda d; Camunda f; Flowable c; Flowable d; Microsoft a; Red Hat; SAP Signavio.)

3 CMMN-kuvauskieli tapausten hallintaan

CMMN (Case Management Model and Notation) on graafinen kuvaustapa, jonka avulla mallinnetaan tapausten käsittelyyn perustuvia työnkulkuja. Kuten BPMN, CMMN on myös OMG:n julkaisema ja ylläpitämä standardi. Uusin versio 1.1 on julkaistu vuonna 2016. Erona BPMN:ään CMMN:llä kuvattu tapaus määrittelee halutut tulokset määrittämättä tarkkaan, miten ne saavutetaan. CMMN keskittyy epämääräisessä järjestyksessä tapahtuvien kokonaisuuksien kuvaamiseen. Yksittäisen prosessin, eli CMMN:n termein tapauksen, aikana suoritettavien aktiviteettien järjestystä, tai sitä, suoritetaanko jotakin aktiviteettia ylipäätään, ei välttämättä tunneta etukäteen. Tapauksen sisältämien mahdollisten aktiviteettien joukko voidaan kuitenkin kuvata tapaussuunnitelmaan. (Object Management Group; Visual Paradigm c.)

Tapaussuunnitelma on salkkumainen tyhjä alue, jonka sisään kuvataan tapauksen sisältämät aktiviteetit ja muut tapauksen kulkua ohjaavat elementit kuten kuvassa 3. CMMN:n yleinen elementti BPMN:n tavoin on tehtävät ja ne voivat olla koneellisia tai yleisemmin ihmisen suoritettavaksi tarkoitettuja. Tehtäviä ja muita elementtejä voi ryhmitellä pienemmiksi kokonaisuuksiksi yleensä laatikkomaisen näköisten vaiheiden avulla. Tapauksen tilaa kuvataan ja visualisoidaan virstanpylväiden avulla. Tapauksen kulkua voidaan ohjata vahtien ja kuuntelijoiden avulla. Vahteja voi liittää muihin elementteihin, jolloin ne aktivoituvat vasta kun vahdin määrittämä ehto täyttyy. Kuuntelijat taas aktivoituvat jonkin tapahtuman myötä, esimerkiksi tietyn ajan kuluttua. CMMN:ää ja BPMN:ää tukevat sovellukset mahdollistavat myös CMMN:llä kuvattujen tapausten käynnistämisen BPMN:llä kuvatusta prosessikaaviosta ja päinvastoin. (Eisner 2020.)



Kuva 3. Esimerkki tapaussuunnitelmasta (Flowable a)

CMMN-kuvaustapaa ja standardia tukevia ohjelmistoja ja palveluja on tarjolla BPMN:n tavoin useilta toimijoilta ja jotkut BPMN:ää tukevat ohjelmistot tai palvelut sisältävät tuen myös CMMN:lle. Aiemmin mainituista tuotteista esimerkiksi Flowable ja Alfresco Process Services sisältävät tuen myös CMMN-tapaussuunnitelmien mallintamiseen ja suorittamiseen. Mallinnus- ja suoritusalustaa tarjoaa myös Trisotech tuotteillaan Digital Automation Suite ja Digital Modeling Suite. Jotkut ohjelmistot tarjoavat tuen vain tapaussunnitelmien mallintamiseen kuten Visual Paradigmin Process Design Tool. (Trisotech a; Trisotech b; Visual Paradigm b.)

4 DMN-standardi ja päätösmoottorit automaattiseen päätöstentekoon

DMN, eli Decision Model and Notation, on OMG:n ylläpitämä standardi BPMN:n ja CMMN:n tapaan. DMN mahdollistaa liiketoimintapäätösten hallinnan eli Business Decision Managementin (BDM). Sitä tarvitaan organisaatioissa esimerkiksi säädösten mukaisen päätöstenteon varmistamiseen tai yksinkertaisesti toistuvien päätösten automatisointiin. DMN-standardin keskiössä ovat operatiiviset päätökset, jotka koostuvat kolmesta osasta: syötteestä, päätöslogiikasta ja lopputuloksesta. Päätökset visualisoidaan kuvan 4 tapaisilla päätöstauluilla, jotka muistuttavat hieman taulukkolaskentaa. Ne sisältävät syötteet ja päätöslogiikan ehdot vasemmassa sarakkeessa ja lopputuloksen oikealla. Yhdessä taulussa voi olla useampia sääntöjä, joilla on vaihtelevat ehdot. Lisäksi päätökselle voi valita osumalogiikan, jonka mukaan säännöistä valitaan esimerkiksi vain yksi, listan ensimmäinen tai kaikki sääntöihin sopivat osumat. Monimutkaisempia päättelyjä voi jakaa osiin ja yhdistää toisiinsa päätösvaatimuskaavion eli Decision Requirements Diagramin (DRD) avulla. Kaavioon voi kuvata myös tärkeimpiä päätöksiin vaikuttavia syötteitä. DMN-standardi sisältää FEEL-kuvauskielen, jonka avulla päätöslogiikkaa voi kuvata ohjelmointikielen tapaan päätöstaulujen sisällä tai jopa kokonaan päätöstaulujen sijaan. Taustalla DMN-standardin mukaiset päätöstaulut ovat BPMN- ja CMMN-standardin tapaan XML-muotoisia tiedostoja. Syötteet voivat olla esimerkiksi päivämäärä, numero- tai tekstimuotoista ja ehtojen muotoiluun voi käyttää esimerkiksi matemaattisia tai Boolen operaattoreita. DMN-standardin mukaisia päätöstauluja ja päätösvaatimuskaavioita suorittavia ohjelmistoja kutsutaan päätösmoottoreiksi. Yleensä BPMN-standardia tukevat ohjelmistotuotteet sisältävät tuen myös DMN-standardille kuten esimerkiksi kaikki aiemmin mainitut BPMN-standardin mukaisia prosessikaavioita tukevat ohjelmistot. DMN-standardin mukaiset päätösmoottori ja päätökset voivat ohjata prosessikaavion tai tapauksen kulkua monimutkaisempien päätösten kohdalla. (Freund & Rücker 2019, 149–163.)

Dish

Show details

U	Input +	Output +	Annotation
	Season	Dish	
1	"Fall"	"Spareribs"	-
2	"Winter"	"Roastbeef"	-
3	"Spring"	"Steak"	-
4	"Summer"	"Light Salad and a nice Steak"	Hey, why not!?
+	-	-	-

Kuva 4. Esimerkki päätöstaulusta (Camunda g)

5 Liiketoimintaprosessin automatisointi

5.1 Teknisen alustan osaset

Jotta automatisoitu prosessi saadaan suoritettua, tarvitaan tekninen ympäristö sovelluksineen. Kehitysprojektille haluttiin luoda hyvien sovelluskehityksen käytäntöjen mukaisesti erilliset kehitys-, testi- ja tuotantoympäristöt. Kehitysympäristönä toimii omalle työasemalle käynnistettävät sovellukset, joita hallitaan Docker Compose-sovelluksella. Omalla työasemalla kehitystä tehtiin Windowsin Subsystem for Linux-ympäristössä, jossa on asennettuna Ubuntu-jakelu. Testi- ja tuotantoympäristöt luotiin Azuren pilvipalveluista: Container Instances konttien suorittamiseen, Key Vault salasanojen säilöntään, Storage Accounts kontin tarvitsemien tiedostojen säilömiseen ja Azure SQL-tietokanta prosessimoottorin käsittelemien tietojen tallentamiseen. Lisäksi Azure DevOpsista otettiin käyttöön Git-versiohallinnan lisäksi Azure Pipelines testi- ja tuotantoympäristön asennusprosessin automatisointiin.

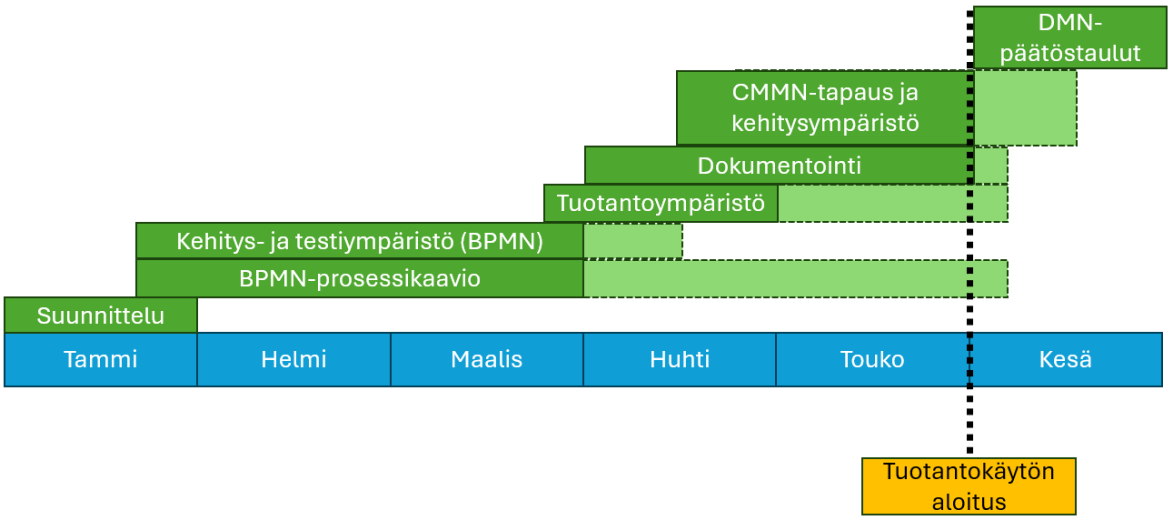
5.2 Vaiheet

Prosessin automatisointi aloitettiin haastattelemalla asiantuntijoita, jotka tunsivat kehitysprojektin kohteena olevan aiheen. Haastattelussa käytiin läpi prosessin tunnistetut vaiheet ja aiemmin luotu ohjelmistologiikka. Haastattelun jälkeen valittiin käytettäviksi ohjelmistoiksi aiemmista työtehtävistä tuttu BPMN-standardiin keskittyvä ja DMN:ää tukeva Camunda. Toiseksi vaihtoehdoksi valittiin Flowable sen monipuolisuuden takia, sillä siitä löytyy BPMN:n ja DMN:n lisäksi tuki myös CMMN-kuvaustavalle. Toimeksiantajalla on käytössään Microsoft Azure-pilviympäristö, joten valittujen ohjelmistojen tulisi olla asennettavissa tai muuten suoritettavissa siinä ympäristössä. Valintoihin vaikutti osaltaan myös hinnoittelu. Avoimen lähdekoodin tai muutoin ilmaiseksi saatavilla olevat ohjelmistot ovat yleensä nopeasti otettavissa käyttöön kehitysympäristöön ja toisaalta haluttiin kerätä kokemuksia, joiden pohjalta on helpompaa arvioida maksullisten vaihtoehtojen tarjoamia etuja.

5.2.1 Aikataulu

Tavoiteaikatauluksi valittiin noin puolen vuoden jakso vuonna 2024jatko tammikuulta toukokuun loppuun, jonka jälkeen päätettäisiin, onko automatisointi riittävällä tasolla luotettavuudeltaan ja muilta osin tuotantokäyttöön. Tälle aikajanalle laadittiin alustava aikataulu, joka on kuvattu tummalla vihreällä kuvassa 5. Suunnitelma sisälsi vaiheet, jotka sillä hetkellä nähtiin pakollisiksi toteuttaa. Jo suunnitelmaa laadittaessa nähtiin, että on tärkeää päästä nopeasti käyttämään valittua sovellusta käytännössä. Tällöin mahdollisesti ilmenevät ongelmat ehditään ratkaista, ja aikaa jää tärkeään testausvaiheeseen. Lisäksi päätettiin, että keskitytään alkuun liiketoimintaprosessien hallinnan prosessimoottorin pystyttämiseen

ja jätetään tapausten hallinnan kokeilu odottamaan, kunnes prosessin automatisointi saadaan tuotantokäyttöön tai hylätään. Aikataulun alkuun tammi-maaliskuulle tavoitteeksi asetettiin siten BPMN-ohjelmiston kehitys- ja testiympäristön rakentaminen sekä prosessin automatisoinnin aloittaminen eli suoritettavan prosessikaavion piirtäminen. Täten tavoitteena oli, että huhtikuussa olisi mahdollista aloittaa tuotantoympäristön rakentaminen testiympäristön mukaisesti ja tuotantokäytön aloittaminen toukokuussa. Lisäksi huhti-toukokuulle tavoitteeksi asetettiin tapausten hallinnan kehitysympäristön rakentaminen ja kehitysprojektin dokumentointi. Tuotantokäytön aloitus lykkääntyi kuukaudella taustajärjestelmiin liittyvistä syistä, mikä mahdollisti useita toimenpiteitä: Tuotantoympäristön ja automatisoidun prosessin laajemman testaamisen sekä prosessikaavion jalostamisen pidemmälle. Toteutunutta aikataulua on kuvassa 5 kuvattu vaaleanvihreällä.



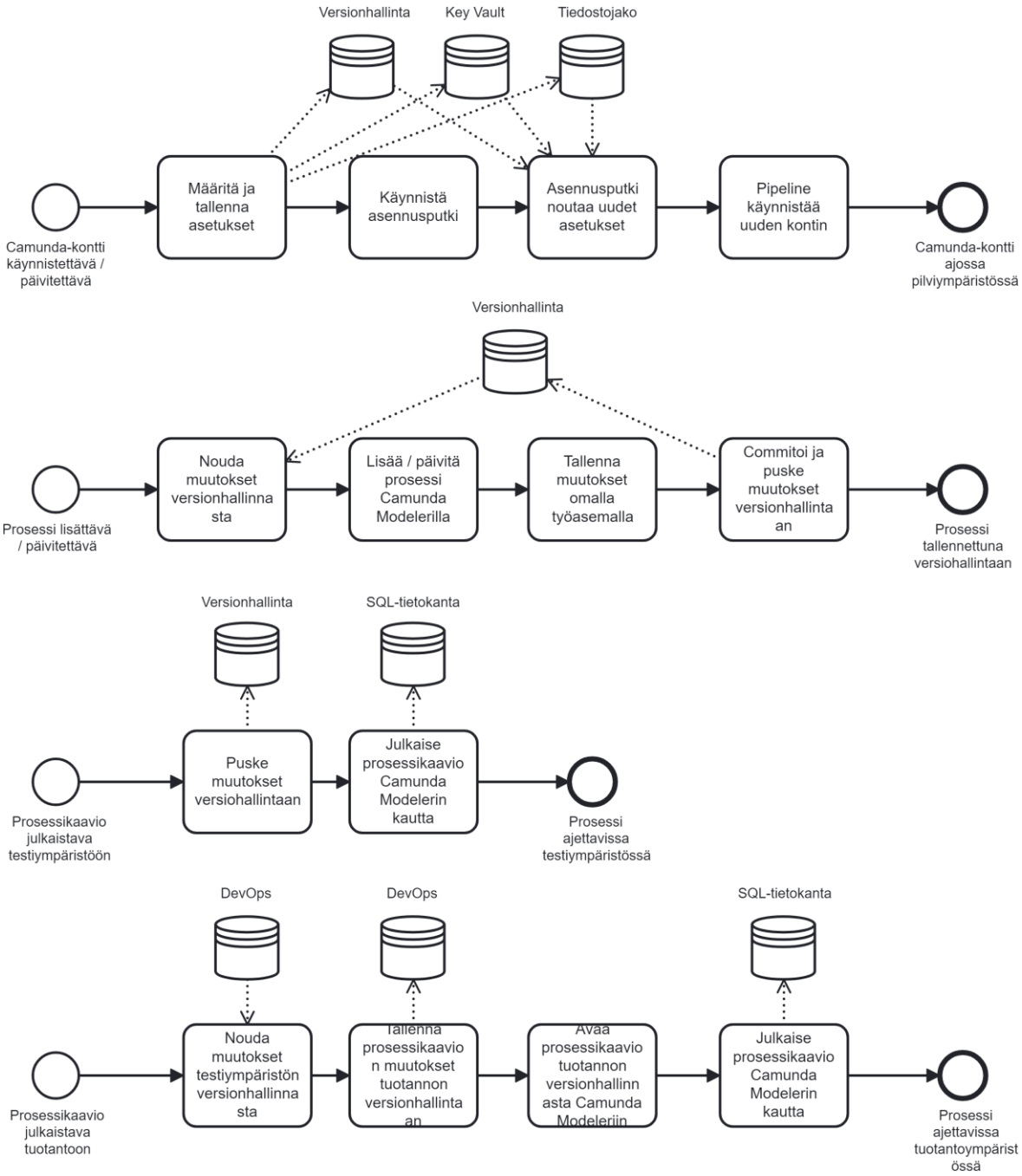
Kuva 5. Kehitysprojektin aikataulusuunnitelma

5.2.2 Esivalmisteluja

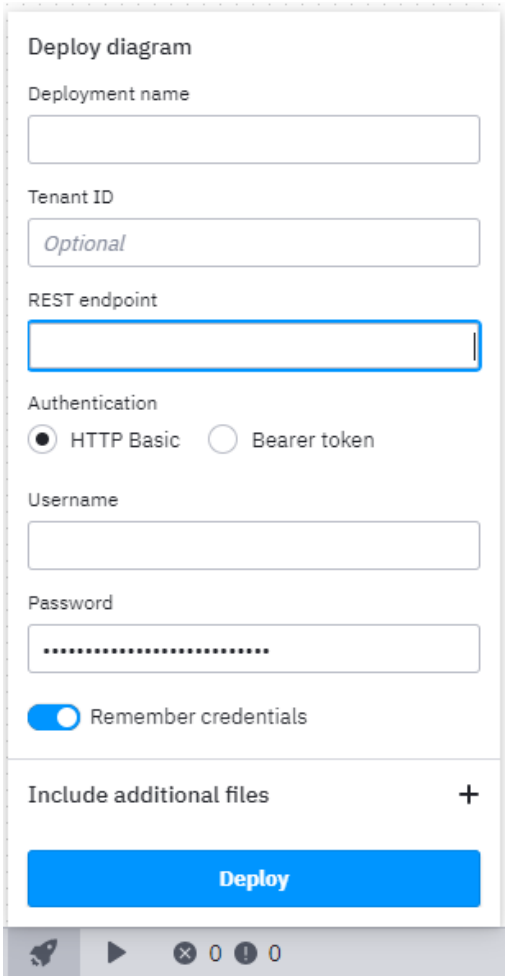
Ensimmäisenä vaiheena tehtiin pikainen testaus pystyttämällä prosessimoottori Camunda 7 Run Azuren pilviympäristöön Container Instances -palveluun ohjeiden mukaan. Camundan 7 Run sekä muita vaihtoehtoisia versioita Camundan prosessimoottorista on saatavilla julkisesti Docker Hub-palvelusta. Run -versioon päädyttiin, koska se mahdollistaa esimerkiksi asetusten muuttamisen helposti ympäristömuuttujilla ja sitä käytettiin myös ohjeissa. Asennus tehtiin manuaalisesti Azuren portaalin kautta ja käyttäen oletusasetuksia, jolloin muun muassa prosessimoottori tallentaa tietokannan kontin sisäiseen levytilaan. Tällöin kontin sammuessa tai käynnistyessä uudelleen kaikki sisältö häviää käyttäjätunnuksineen ja prosessikaavioineen. Ohjeistuksen avustamana tähän ei kuitenkaan mennyt kuin joitain

kymmeniä minuutteja ja samalla varmistettiin, että asennushetkellä uusin versio tulisi todennäköisesti toimimaan halutulla tavalla myös halutulla tavalla konfiguroituna. Container Instance sammutettiin ja poistettiin kun toimivuus oli varmistettu. (Emsbach 2020.)

Tämän jälkeen tarkennettiin teknisen ympäristön rakennetta ja prosessimoottorin käytön työnkulkuja, joita on kuvattu kaaviossa 2. Prosessimoottorin käytössä tunnistettiin vastaavanlaisia työnkulkuja kuin useissa muissakin IT-sovellusten käytössä. Tarvittiin toimintatavat sovelluksen, eli prosessimoottorin, asennukselle ja päivitykselle uudempaan versioon sekä sisällön, eli prosessikaavioiden, lisäämiselle ja päivitykselle prosessimoottoriin. Lisäksi kartoitettiin pakolliset lisäpalvelut, joita prosessimoottori tai työnkulut tarvitsevat toimiakseen. Konttien taustalla olevan imagen muokkaaminen ja muokatun imagen käyttäminen vaatisi käytännössä tallennuspaikan, eli Container Registryn, mutta tässä tapauksessa koettiin, että pitäydytään valmiista imagesta käynnistetyn kontin asetusten muokkauksessa, ellei jotakin todella merkittävää tarvetta nousisi esiin. Prosessikaaviot voi julkaista prosessimoottorille muutamalla eri tavalla. Yksi tapa olisi sisällyttää kaavioiden BPMN-tiedostot sopivaan hakemistoon uuden rakennettavan kontin sisälle. Prosessimoottori osaa tutkia joitakin tiettyjä hakemistoja BPMN-tiedostojen varalta ja julkaisee ne sitten käynnistyksen yhteydessä automaattisesti käyttöön. Tämä tapa olisi vaatinut kuitenkin konttien tallennuspaikan ja imagen rakentamisen uudelleen jokaisen prosessikaavion päivityksen yhteydessä. Toinen tapa julkaista prosessit prosessimoottorille on REST-rajapinnan avulla ja tähän prosessimoottorien käyttöliittymissä on erilaisia keinoja, jotka ovat helpompia kuin kutsun rakentaminen itse. Camunda Modeler -mallinnusohjelma tarjoaa julkaisemiselle helpon tavan suoraan kaavion piirtonäkymästä kuvan 6 mukaisesti. Julkaisemiseen tarvitaan vain prosessimoottorin REST-rajapinnan osoite ja käyttäjätunnukset, jonka jälkeen mallinnusohjelma tekee tarvittavan kutsun taustalla. Toisaalta Flowablen käyttöliittymä mahdollistaa sekä BPMN-, DMN-, että CMMN-muotoisten kaavioiden piirtämisen ja julkaisemisen suoraan omalla käyttöliittymällään muutamaa nappia painamalla tai vaihtoehtoisesti kaavioiden tuonnin jostain toisesta mallinnusohjelmasta. Lopulta tarpeellisiksi palveluiksi koettiin versionhallinta, asennus- ja päivitysputket, tallennuspaikka salasanoille, tiedostojako, sekä erillinen tietokanta. Lisäksi kaavioiden ja päätöstaulujen piirtämiseen valittiin Camunda Modeler -mallinnusohjelma, jonka kautta onnistuu BPMN-muotoisten prosessikaavioiden ja DMN:n mukaisten päätöstaulujen julkaisu.



Kaavio 2. Kehitysprojektissa tarvittavat työnkulut



Kuva 6. Camunda Modelerin prosessikaavion julkaisutoiminto

5.2.3 Kehitysympäristön valmistelu

Kehitysympäristön valmistelu aloitettiin kehitysprojektin alussa, mutta osin rinnakkain testiympäristön rakentamisen kanssa. Kehitysympäristön tavoitteena on olla olennaisilta osiltaan vastaavanlainen kuin testi- ja tuotantoympäristöt, mutta käytettävissä omalla työasemalla. Suunnitelman etuna on se, että prosessin automatisointia päästiin aloittamaan prosessikaavion piirtämisen muodossa ja teknisten yksityiskohtien testaaminen onnistui, vaikkei pilviympäristössä ollut vielä käyttökelpoista alustaa. Kehitysympäristön hallintaan käytettiin Docker Compose -orkestraatiotyökalua. Halutut palvelut ja niiden asetukset määriteltiin YAML-muodossa kuvan 7 mukaiseen compose-tiedostoon, minkä jälkeen sovellukset saa komentorivin avulla käyntiin yhdellä komennolla. Dockerin ja Docker Composen käyttö mahdollistaa pilviympäristöjen tavoin konttien konfiguroinnin ja suorittamisen, mutta omalla työasemalla. Kehitysympäristön tarvitsemat tiedostot tallennettiin Git-versiohallintaan, jonka tässä tapauksessa toteuttaa Azuren DevOps-pilvipalvelu.

```yaml
version: '3.9'
services:
 camunda:
   image: camunda/camunda-bpm-platform:run-7.20.0
   command: ["/bin/bash", "-c", "wget -P /camunda/configuration/userlib/ https://repo1.maven.org/maven2/com/microsoft/sqlserver/mssql-jdbc/12.6.0.jre11/mssql-jdbc-12.6.0.jre11.jar; wget -P /camunda/configuration/userlib/ https://repo1.maven.org/maven2/org/camunda/connect/camunda-connect-connectors-all/1.6.0/camunda-connect-connectors-all-1.6.0.jar; cat /tmp/prodconf/production.yml > /camunda/configuration/production.yml; cp /tmp/prodconf/userlib/camunda-webapp-webjar-7.20.0.jar /camunda/internal/webapps/camunda-webapp-webjar-7.20.0.jar; cp /tmp/prodconf/resources/* /camunda/configuration/resources/; ./start.sh --webapps --rest --production"]
   volumes:
     - ./camunda/configuration/:/tmp/prodconf
   ports:
     - "8080:8080"
     - "8443:8443"
   networks:
     - camunda-network
   depends_on:
     - mssql
 mssql:
   image: mcr.microsoft.com/mssql/server:2019-latest
   environment:
     - ACCEPT_EULA=TRUE
     - MSSQL_SA_PASSWORD=******
   ports:
     - "1433:1433"
   healthcheck:
     test: ["CMD-SHELL", "/opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P ****** -Q 'SELECT 1' || exit 1"]
     interval: 10s
     retries: 10
     start_period: 10s
     timeout: 3s
   networks:
     - camunda-network
 mssql.configurator:
   image: mcr.microsoft.com/mssql/server:2019-latest
   volumes:
     - "./mssql:/tmp/mssql-scripts"
   depends_on:
     mssql:
       condition: service_healthy
   command: >
     bash -c '
     /opt/mssql-tools/bin/sqlcmd -S mssql -U sa -P ****** -i /tmp/mssql-scripts/CreateDb.sql -C;
     echo "Command finished";
     '
   networks:
     - camunda-network
networks:
 camunda-network:
   name: camunda-network
```

Kuva 7. Kehitysympäristön palvelut ja asetuksia

Kehitysympäristössä käytettiin aluksi Camundan 7 Run -imagen päälle rakennettua muokattua imagea, jotta puuttuva MS SQL -tietokanta-ajuri saatiin lisättyä konttiin valmiiksi ennen käynnistystä. Camunda on Java-pohjainen sovellus, joten sopiva ajuri löytyi yleisesti Java-ohjelmoinnissa käytetyn Maven-projektinhallintatyökalun pakettivarastosta. Etsinnässä auttoi myös pakettivarastojen sisältöjä kokoava MVN Repository -verkkosivusto. Ajurin lisääminen konttiin onnistui lataamalla ajuri internetsivustolta paikalliseen hakemistoon ja luomalla Dockerfile, jossa kerrottiin mistä hakemistosta kopioidaan ajuritiedosto luotavan imagen sisällä olevaan hakemistoon. Docker Compose osaa käynnistyskutsun saatuaan

automaattisesti rakentaa uuden kontin Dockerfilen kuvauksesta ja sitten käynnistää kontin käyttämällä luotua imagea. Testiympäristössä ei kuitenkaan ollut container registryä valmiina, mutta sellainen olisi ollut saatavilla Azure Container Registry-palvelun kautta. Tässä piili kuitenkin tulevaisuutta ajatellen ongelma, sillä palvelun ilmaisversiolla julkaistut imaget olisivat julkisia ja todennäköisesti niihin tultaisiin sisällyttämään tuotantokäytön kannalta olennaisia asetuksia. Maksullista versiota ei päätetty ottaa pelkästään tämän kehitysprojektin tarpeisiin, joten täytyi kehittää toinen tapa ajurin saamiseksi kontin sisälle ja Camundan käyttöön. Lopulta ajuri päädyttiin lataamaan kontin sisälle aina käynnistyksessä hyödyntämällä command override -ominaisuutta. Ylikirjoittamalla käynnistyskomennon kontille voidaan antaa valmistelevia komentoja, joiden avulla ajurin lataaminen onnistuu oikeaan hakemistoon ennen Camundan käynnistymistä. Käynnistyskomennon ylikirjoittaminen onnistuu sekä Docker Composella kehitysympäristössä, että testi- ja tuotantoympäristössä Azuren Container Instanceille. Näin ympäristöt pysyivät vastaavina.

Kun ajuri oli saatu käyttöön, pystyttiin kehitysympäristössä ottamaan käyttöön Microsoftin SQL Server 2019 Developer Edition -tietokantapalvelin konttina Docker Hubin kautta. Developer Editionin lisenssi mahdollistaa palvelimen käytön sovelluskehitykseen ja testaukseen, mutta ei tuotantokäyttöön ilman erillistä lisenssiä. Camundalle tarvittiin tietokantapalvelimelle oma tietokanta ja lisäksi dokumentaatio suosittelee asettamaan tietokannalle yhteensopivuustason. Tietokannan luonti ja asetusten muuttaminen ei onnistu suoraan ympäristömuuttujia asettamalla. Tässä tapauksessa myöskään käynnistyskomennon ylikirjoittamisesta ei ole hyötyä, sillä kyseiset toimenpiteet pitää tehdä vasta sen jälkeen, kun tietokantapalvelin on käynnistynyt. Ongelman kiertämiseksi compose-tiedostoon lisättiin toinen identtinen tietokantapalvelinkontti. SQL Server -kontteihin sisältyy myös komentorivityökalu nimeltä Sqlcmd, jonka avulla voi käskyttää tietokantapalvelinta. Tätä toista konttia kehotettiin odottamaan healthcheck-toiminnon avulla, kunnes varsinainen tietokantapalvelin on käynnistynyt ja sen jälkeen halutut komennot suoritettiin skriptitiedostosta toisen kontin Sqlcmd-työkalun avulla tietokantapalvelimeen. Tämän jälkeen toinen SQL Server -kontti sammuttaa itsensä ja Camundan tietokannan sisältävä kontti jatkaa toimintaansa sisältäen nyt tyhjän tietokannan ja tarvittavat yhteensopivuusasetukset. Käyttäjätunnusta ei tarvitse erikseen luoda tietokannan käyttämiseksi, koska konteissa on valmiina laajoin oikeuksin varustettu käyttäjätili.

Tietokantayhteyteen liittyvät asetukset kuten tietokannan osoite, portti, tietokannan nimi ja tunnukset määritettiin tässä vaiheessa ympäristömuuttujina compose-tiedostoon. Hieman myöhemmin päädyttiin testiympäristön rajoitteiden takia kuitenkin siirtämään asetukset kuvan 8 mukaiseen konfiguraatiotiedostoon, jonka käyttöön Camundan dokumentaatiokin ohjaa tarvittaessa (Camunda b). Camundan 7 Run -kontille voi antaa käynnistysparametrinä

”--production”, jolloin käytetään production.yml-konfiguraatiotiedostoa. Tiedosto tuodaan konttiin liittämällä omalta työasemalta paikallinen hakemisto levyjakona kontin sisään ja käynnistyskomennossa vuorostaan ylikirjoitetaan alkuperäinen konfiguraatiotiedosto. Tässä vaiheessa tiedettiin jo, että todennäköisesti konfiguraatiotiedostoa tarvitaan myös muiden asetusten muokkaamiseksi.

```
server:
# https://docs.camunda.org/manual/latest/user-guide/camunda-bpm-run/#https
# do not use the provided certificate in production
#  ssl:
#    key-store: classpath:keystore.p12
#    key-store-password: camunda
#    key-store-type: pkcs12
#    key-alias: camunda
#    key-password: camunda
#  port: 8443

# datasource configuration is required
# do not use the H2 databse in production
# https://docs.camunda.org/manual/latest/user-guide/camunda-bpm-run/#connect-to-a-database
# https://docs.camunda.org/manual/latest/user-guide/camunda-bpm-run/#database
spring.datasource:
  url: jdbc:sqlserver://mssql:1433;databaseName=camunda;encrypt=false
  driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
  username: sa
  password: ******

# Camunda Run process engine plugin registration
# https://docs.camunda.org/manual/7.20/user-guide/camunda-bpm-run/
camunda.bpm.run.process-engine-plugins:
  - plugin-class: org.camunda.connect.plugin.impl.ConnectProcessEnginePlugin
```

Kuva 8. Kehitysympäristön asetuksia

Asetusten ollessa kunnossa käynnistettiin paikallinen ympäristö komennolla docker-compose up -d, joka käynnistää kontit ja palvelut toimintakuntoon. Parametri -d siirtää suorituksen taustalle, jolloin aktiivinen komentorivisessio ei jää Docker Composen käyttöön. Palvelujen käynnistyttyä tarkistettiin sovelluslokista, että palvelut ovat käynnissä ja odottavat käyttäjän tekemisiä. Seuraavaksi Camundan Cockpit-käyttöliittymän kautta luotiin pääkäyttäjätunnus. Camunda tarjoaisi LDAP-integraatiota, mutta tässä tapauksessa paikallinen käyttäjähallinta on riittävä. Tällöin käyttäjätunnus ja käyttöoikeudet tallentuvat Camundan omaan SQL-kantaan kuten muukin prosessimoottorin käsittelemä data. Data säilyy, vaikka MS SQL-kontti käynnistettäisiin uudelleen, mutta poistuu, mikäli kontti poistetaan ja luodaan uudelleen.

Ennen varsinaista prosessiautomaation kehittämisen aloittamista tiedettiin, että tarpeena tulee olemaan REST-kutsujen tekeminen taustapalvelun rajapintaan. Camundan prosessimoottoriin voi tuoda lisää ominaisuuksia lisäosien muodossa ja Camunda onkin itse tuottanut joitakin esimerkkejä. GitHub-versiohallintapalvelusta ja Camundan dokumentaatiosta löytyi sopiva HTTP Connector-lisäosa, joka sisältyy Camunda Connectors All -pakettiin SOAP-rajapintakutsut mahdollistavan lisäosan kanssa. Camunda Connectors All -paketti lisättiin kuvassa 8 näkyvän konfiguraatiotiedoston listaukseen dokumentaation ohjeiden mukaan. Lisäosa ladataan kontin käynnistyessä kohdehakemistoon vastaavasti kuin tietokanta-ajuri ja tämän jälkeen lisäosan käyttö on mahdollista prosessikaaviossa. Vaihtoehtona olisi esimerkiksi luoda Java-koodilla oma toteutus, mutta lisäosan ominaisuudet nähtiin riittävänä ja nopeana tapana hoitaa asia.

5.2.4 Testiympäristön valmistelu

Testiympäristöä alettiin luoda tammi-helmikuun vaihteessa. Testiympäristö rakennettiin pääosin graafisen Azuren portaalinäkymän kautta. Camunda-kontin pystyttäminen tehtiin samaan tapaan kuin aiemmin ohjeiden mukaan. Eroavaisuutena tällä kerralla tietokanta-asetukset yritettiin lisätä ympäristömuuttujien avulla dokumentaation ohjeiden mukaan. Camunda 7 Run -kontille annettavat ympäristömuuttujat sisältävät isoja kirjaimia ja pisteitä, jotka muodostuivat esteeksi kontin pystyttämiselle. Azuren Container Instanceille ei ole sallittua antaa pisteitä sisältäviä muuttujia. Tämän jälkeen kokeiltiin vaihtoehtoisia ympäristömuuttujia, joissa ei ole pisteitä, mutta myöskään ne eivät näyttäneet toimivan. Tämän jälkeen tietokanta-asetukset vietiin kehitysympäristön tapaan konfiguraatiotiedostoon. Azuren pilviympäristössä voi tehdä vastaavantyyppisen levyjaon liittämisen tallennustilaksi konttiin kuten kehitysympäristössä tehtiin paikallisella hakemistolla. Erona kehitysympäristöön, pilviympäristössä konttiin liitettiin Azuren Storage Accountin alainen tiedostojako. Tiedostojakoon vietiin konfiguraatiotiedosto Azuren portaalinäkymän kautta ja käynnistysparametreihin lisättiin alkuperäisen konfiguraatiotiedoston ylikirjoittaminen kehitysympäristön tapaan. Lisäksi erona kehitysympäristöön, tietokantayhteydelle otettiin käyttöön salaus vaihtamalla encrypt-parametrin arvo.

Testiympäristössä hyödynnettiin Azure SQL -pilvipalvelinta, joka on Microsoft SQL Serverin kanssa yhteensopiva ja siten voitiin käyttää samaa tietokanta-ajuria kuin kehitysympäristössä. Tässä tapauksessa hyödynnettiin olemassa olevaa palvelinta, jolloin tehtäväksi jäi kehitysympäristön tapaan luoda uusi tietokanta palvelimelle, luoda uudet käyttäjätunnukset tietokantaan Camundalle, antaa käyttäjätunnukselle riittävät oikeudet lukea ja kirjoittaa tietokantaan sekä luoda tietokantatauluja. Toisin kuin kehitysympäristössä, tässä

tapauksessa tietokanta ja käyttäjätunnukset luotiin manuaalisesti SQL-lausekkeilla avaamalla tietokantayhteys SQL Server Management Studio -sovelluksen kautta. Käyttäjätunnukset tallennettiin Azure Key Vaultiin, josta ne ovat useamman käyttäjän saatavilla ja tietoturvallisesti tallessa.

Testi- ja tuotantoympäristöön tarvittiin käyttöön salattu HTTPS-yhteys, sillä vaikka prosessimoottorin sekä tietokannan välinen liikenne olikin jo salattu, tarvittiin salaus käyttöön myös rajapintakutsuille taustapalveluihin sekä käyttöliittymän käyttöön. Camunda 7 Run -kontissa on valmiina tuki salaukselle, mutta oletuksena mukana tulee vain kontin paketoijan itse allekirjoittama sertifikaatti. Tällaista sertifikaattia ei tueta oletuksena nykyisissä yleisissä selainohjelmissa eikä myöskään taustapalveluissa, joten edessä oli allekirjoitetun sertifikaatin tilaaminen. Sertifikaatin tilaus vaati CSR-tiedoston eli sertifikaatin allekirjoituspyyntötiedoston luontia. Ensin piti luoda Java KeyStore -tiedosto, jonka luonnissa annetaan tulevaan sertifikaattiin halutut tietokentät. KeyStoren luonti onnistui Java-ohjelmointikielen työkaluihin kuuluvalla Keytool-komentorivityökalulla. CSR-tiedoston pystyi tämän jälkeen luomaan Keytoolilla KeyStoren tietojen pohjalta. Kun sertifikaatti oli tilattu ja myöhemmin vastaanotettu, se tallennettiin KeyStore-tiedostoon. Tässä vaiheessa salattu yhteys ei kuitenkaan vielä toiminut. Asian selvittämisen lopputuloksena näytti, että sertifikaatti oli tallentunut KeyStoreen väärin, TrustedKeyEntry-tyyppisenä. Lopulta selvisi, että KeyStoresta piti ottaa yksityinen avain erilleen omaksi tiedostokseen ja sen jälkeen luoda uusi KeyStore yhdistämällä yksityinen avain sertifikaattiin Keytoolin avulla. Nyt sertifikaatti tallentui PrivateKeyEntrynä KeyStoreen. KeyStorea luotaessa sille annettiin salasana, jota KeyStoren käyttäminen tulee vaatimaan. Salasana ja muut KeyStoren asetukset kirjattiin Camundan konfiguraatiotiedostoon. KeyStore tallennetaan Azuressa tiedostojakoon, kuten konfiguraatiotiedosto aiemmin. Näin KeyStore-tiedosto saadaan kopioitua kontin käynnistyessä oikeaan hakemistoon.

Testiympäristölle luotiin myös automatisoidun päivittämisen tai uudelleen asentamisen mahdollistava asennusputki Azure DevOpsiin. Tässä hyödynnettiin aiempaa toteutusta, johon päivitettiin testiympäristön viimeisimmät manuaalisesti syötetyt asetukset. Näin jatkossa muutosten tekeminen on hallitumpaa ja asetukset ovat tallessa versionhallinnassa. Huhtikuun puolivälissä testiympäristö saatiin teknisesti halutulle tasolle ja sitä voitiin alkaa käyttää prosessin testaamiseen.

Kun prosesseja saatiin testattua, heräsi tarve myös nähdä historiatietoja visuaalisesti, sillä onnistumisten ja epäonnistumisten kertyessä olisi hyödyllistä nähdä helposti prosessi-instanssien tietoja esimerkiksi alkamisen, päättymisen ja yksittäisten tehtävien suorituksen ajankohta. Oletuksena Camundan Cockpit-käyttöliittymällä ei näe päättyneitä prosesseja,

mutta käyttöliittymäkomponentteihin on kehitetty lisäosa, Minimal history plugins, jonka avulla edellä mainittuja tietoja pääsee selaamaan. Lisäksi lisäosa näyttää myös päättyneelle prosessi-instanssille tallentuneiden muuttujien arvot. Lisäosan asennus vaati hieman työtä, sillä sen asentaminen tapahtuu tässä tapauksessa purkamalla Camundan kotisivuilta ladatusta asennuspaketista käyttöliittymän toteuttavat tiedostot sisältävä Java-arkisto. Arkiston sisällä olevat alkuperäiset tiedostot korvataan lisäosan GitHub-sivulta ladattavilla tiedostoilla. Sitten päivitetty arkisto tallennettiin Azuren tiedostojakoon ja testiympäristön Camunda-kontin käynnistysparametreihin lisättiin komento, joka ylikirjoittaa kontin sisältämän arkistotiedoston päivitetyllä versiolla. (Minimal "history plugins" for Camunda Cockpit.)

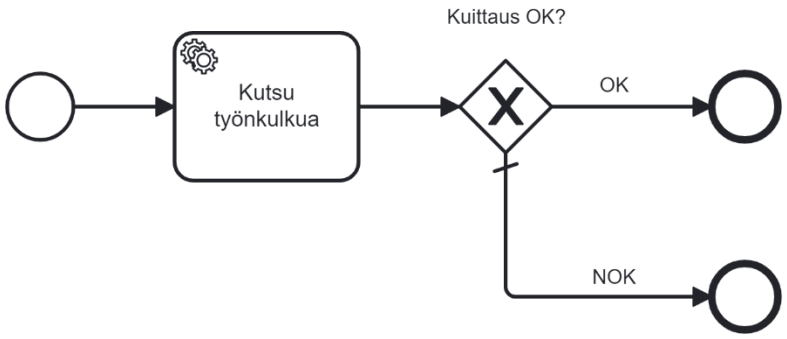
5.2.5 Tuotantoympäristön valmistelu

Tuotantoympäristön valmistelulle varattiin noin kuukausi aikaa ennen käyttöönottoa, joten pari viikkoa testiympäristön toimintakuntoon saattamisen jälkeen toukokuussa aloitettiin jo tuotannon valmistelevat työt. Pääsääntöisesti tuotantoympäristössä tehtiin vastaavat toimenpiteet kuin testiympäristössä. Erona testiympäristöön aloitettiin tekemällä asennusputken kuvaus testiympäristössä käytössä olevien sovellusten ja pilvipalvelujen pohjalta. Palvelujen pystytys tehtiin ensimmäisestä kerrasta lähtien automatisoidusti asennusputken kautta. Käytössä oli muistiinpanot testiympäristön pystytyksestä, joita seuraamalla useimmat asiat hoituivat nopeasti ja pystyttiin varautumaan jo etukäteen mahdollisesti aikaa vieviin osuuksiin. Tuotantoympäristö saatiin vastaavaksi kuin testiympäristö ja teknisesti testattua kesäkuun ensimmäisellä viikolla. Ensimmäiset tuotantoajot viikon kuluttua kesäkuun toisella viikolla. Kehitysprojektin dokumentointia tehtiin useampaan otteeseen kevään aikana toteutustyön ohessa ja dokumentaatio viimeisteltiin tuotantokäytön alussa tuotantoympäristön teknisillä yksityiskohdilla.

5.2.6 Prosessikaavion kehittäminen

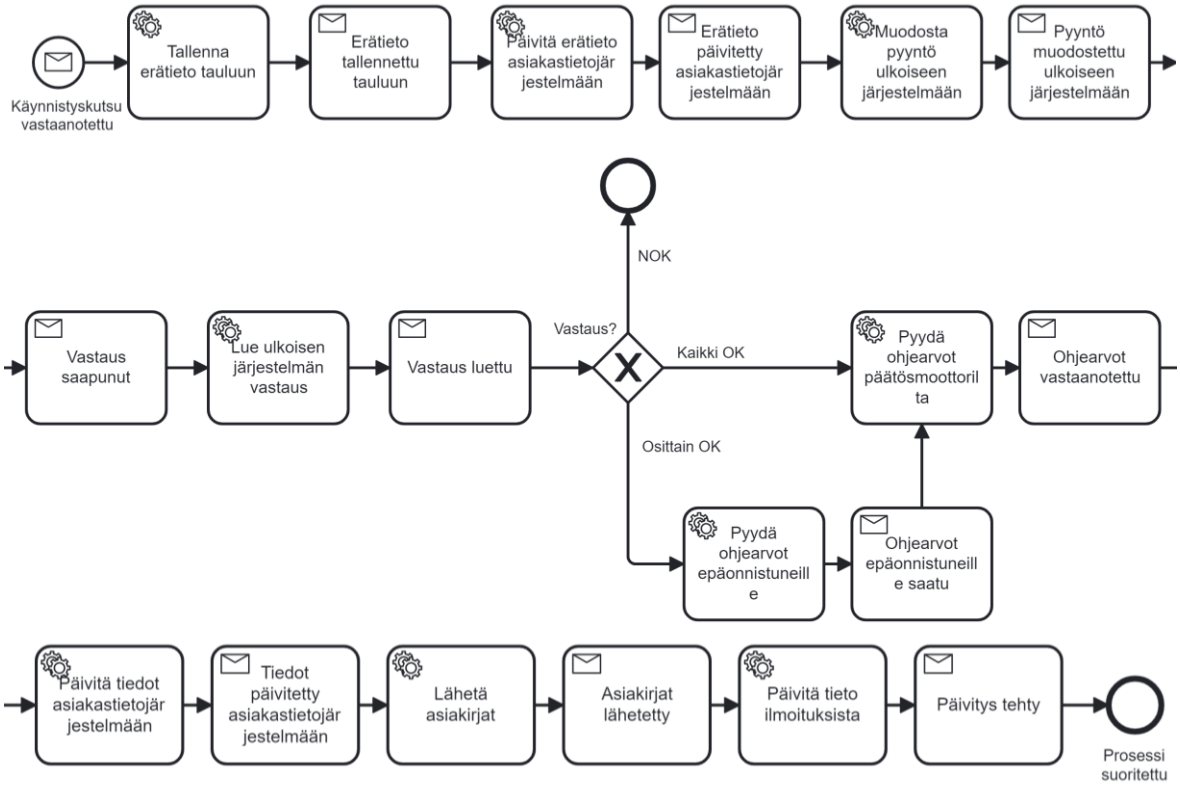
Prosessikaavion kehittäminen alkaa luomalla uusi BPMN-tiedosto, antamalla sille selkokielinen nimi ja tekninen tunnus, eli joidenkin prosessimoottorien termeillä avain. Uudessa prosessikaaviossa on valmiina aloitustapahtuma, jonka perään voidaan aloittaa prosessin tehtävien lisääminen. Tässä tapauksessa prosessin tiedettiin koostuvan ennen kaikkea REST-kutsuista taustapalveluun, joten prosessikaavion kehittämisessä lähdettiin kaavion 3 mukaisesti liikkeelle yksinkertaisesta alkutapahtuman, yhden tehtävän, risteyksen ja lopputapahtuman sisältävästä kokonaisuudesta. Alun ja lopun välissä olevan tehtävän toteuttamiseen käytettiin HTTP Connector-lisäosaa, jonka tarkoituksena on REST-kutsun tekeminen. Tehtävän tyypiksi määritettiin service task, joka on tarkoitettu juuri tämän tyyppisille teknisille tehtäville. Taustapalvelu taas oli asetettu tätä testiä varten vastaamaan yksinkertaisella

JSON-muotoisella kuittaussanomalla: { ”kuittaus”: ”ok” }. Nyt kun prosessi julkaistiin testiympäristöön ja käynnistettiin manuaalisesti käyttöliittymältä, taustapalveluun lähetettiin kutsu, jonka vastaussanoma saatiin tallentumaan prosessi-instanssin muuttujiin. Sen jälkeen prosessissa tarkistettiin vielä vastaussanomasta tallennettu kuittaustieto. Tarkistuksessa hyödynnettiin risteyselementtiä, josta lähtevään OK-haaraan lisättiin ehdoksi, että kuittauksen on oltava ”ok”. NOK-haara taas määritettiin oletussuunnaksi suoritukselle, mikäli kuittaus on jotakin muuta. Tällaisella prosessikaaviolla varmistettiin siis jo kehitysympäristössä, että yhteys taustapalveluun toimii ja myöhemmin voitiin testata tarpeen mukaan esimerkiksi REST-kutsun parametrien vaihtamista.



Kaavio 3. Ensimmäinen versio BPMN-prosessikaaviosta

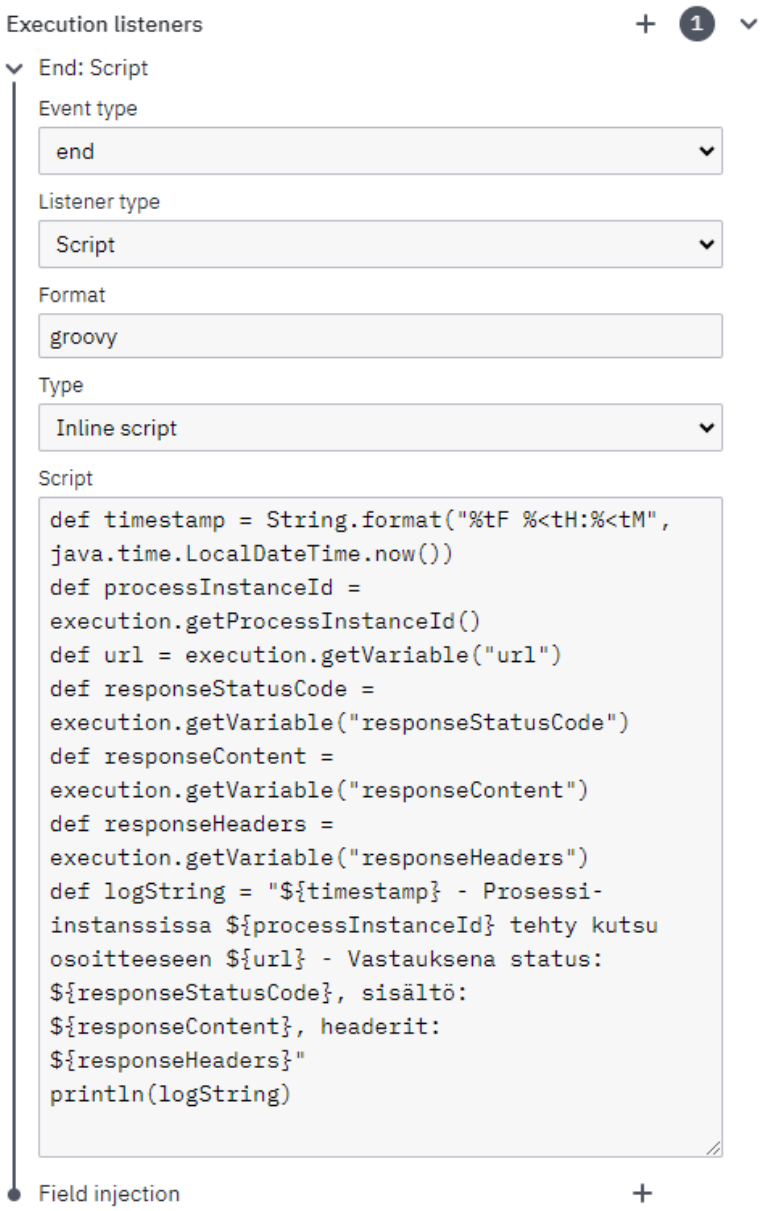
Kun testiympäristö oli saatettu toimintakuntoiseksi, sinne julkaistiin vastaava testiprosessi, mutta prosessin alkutapahtuma muutettiin niin, että prosessi käynnistyisi prosessimoottorin REST-rajapintaan tulevasta ulkoisesta viestistä. Alkutapahtuma vaihdettiin siis none start event -tyyppisestä message start event -tyyppiseksi. Tällä haluttiin testata yhteys testiympäristössä toimivan taustapalvelun ja prosessimoottorin välillä toiseen suuntaan. Prosessimoottorin viestirajapinnan testaaminen oli tärkeää myös siksi, että prosessista hahmoteltiin isoilta osin asynkronista. Se tarkoittaa, että prosessimoottori käynnistää prosessin seuraavan vaiheen kutsumalla taustapalvelua ja sen jälkeen odottelee vaiheen valmistumisesta kertovaa kuittausta saapuvaksi. Kuittauksen saapumisen jälkeen edetään seuraavaan vaiheeseen ja tehdään vastaavat toimenpiteet. Kuittausviestit saapuvat saman REST-rajapinnan kautta kuin prosessin aloittava sanoma. Kun rajapintakutsut ja yhteydet saatiin testattua puolin ja toisin, prosessista mallinnettiin kaavion 4 mukainen, jolloin se sisälsi alkutapahtuman ja lopputapahtuman välissä vuorotellen seuraavan vaiheen käynnistäviä kutsuja ja vaiheen valmistumista odottavia viestin vastaanottotehtäviä, eli receive taskeja.



Kaavio 4. Prosessikaavion toinen kehitysversio

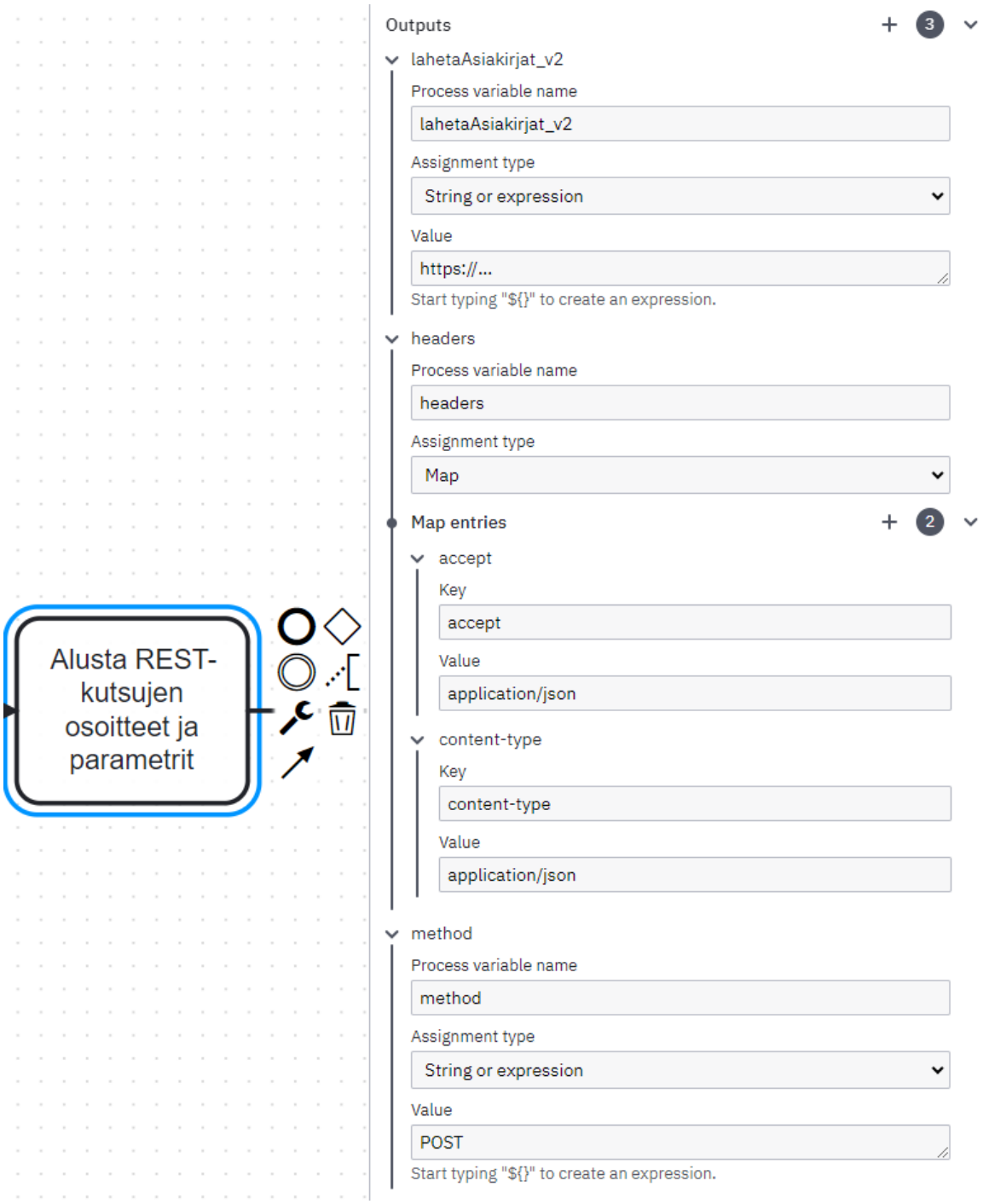
Useamman tehtävän sisältävän prosessin testaaminen nosti esiin tarpeen saada tietoa prosessin kulusta testaamisen aikana ja jälkikäteen. Tiedon saanti onnistuu ainakin kolmella eri tavalla: visuaalisten käyttöliittymien avulla, kirjoittamalla tietoa sovelluslokille ja tekemällä tietokantahakuja. Tietokantahakujen tekeminen on tehokas tapa selvittää tapahtumien kulkua historiataulujen sisältämää tietoa hyödyntäen, mutta koska prosessimoottorin tarkoituksena on olla myös vain pintapuolisesti sovellusta tuntevien käytössä, tarvittiin nopeammin omaksuttava tapa. Camunda tarjoaa mahdollisuuden hyödyntää ja hienosäätää valmista lokituskomponenttia, mutta se vaikutti dokumentaation pohjalta rajoittuvan vain sovelluksen teknisiin tapahtumiin, jolloin liiketoimintaprosessin tapahtumat, esimerkiksi tehtävien onnistuminen tai epäonnistuminen, pois lukien tekniset virheet, eivät sisälly lokitiedostoon. Esimerkiksi Java-kielellä kirjoitetut prosessimoottorin ulkopuoliset tehtävät saisi lokituskomponenttia hyödyntämällä mukaan, mutta kehitysprojektin puitteissa Java-ohjelmointiin ei ryhdytty. Ratkaisuna tilanteeseen hyödynnettiin tehtävien kuuntelijaominaisuutta, joka mahdollistaa pienien skriptien suorittamisen tehtävien aluksi tai lopuksi. Camunda 7 Run -kontti tukee valmiiksi Groovy- ja Javascript -kielisiä skriptejä, joista käytettiin Groovyä sen tuttuuden vuoksi. Prosessikaavion aloitukseen, lopetukseen, jokaiseen tehtävään ja viestikuuntelijaan lisättiin kuvan 9 tapaan yksinkertainen skripti, joka kertoo prosessi-instanssin

tunnisteen ja mikä tehtävä suoritettiin. Tässä vaiheessa yritettiin myös kirjoittaa lokiin tarkempaa tietoa, esimerkiksi vastauksen statuskoodia ja sisältöä, HTTP-kutsuista. Tässä vaiheessa HTTP Connector-lisäosalta ei yrityksistä huolimatta saatu tallennettua mitään tietoja prosessi-instanssiin, joten kuittausten tarkastuksesta luovuttiin toistaiseksi. Lisäksi selvitettiin voisiko virhetilanteita yrittää kaapata ja hallita lisäämällä HTTP-kutsutehtäviin boundary error event-virhetehtävä. Tällainen virhetehtävä kuitenkin osoittautui aktivoituvan BPMNError-tyyppisistä virheistä, jotka eivät ole käytössä HTTP Connectorin lähdekoodissa. (Camunda c; Camunda e.)



Kuva 9. Kuuntelijan lisääminen BPMN-elementtiin

Tarkemmalle tiedolle myös siitä, millaisen kutsun prosessimoottori teki ja mitä taustapalvelu vastasi, olisi kuitenkin hyötyä testatessa, sillä julkaistujen prosessien sisältämien tehtävien asetuksia ei näe käyttöliittymältä, jos asetukset on määritetty tehtävän asetuksiin suoraan teksti- tai numeromuodossa eikä viittauksina prosessi-instanssille tallennettuihin muuttujiin. Käyttöliittymältä nähdään ainoastaan prosessi-instanssin sisältämät muuttujat. Yhtenä vaihtoehtona on mahdollista pitää prosessikaavioeditoria avoinna ja tarkistaa asetukset sieltä. Jos editorissa on kuitenkin prosessikaavion muokkaaminen meneillään, voi olla työlästä perua muutoksia vain tarkistaakseen aiempia asetuksia. Prosessimoottorin käyttämät HTTP-kutsujen asetukset muutettiin kuvan 10 mukaisesti prosessiin muuttujiksi ja ne asetettiin ensimmäisenä tehtävänä prosessin alussa. Tämä mahdollisti myös asetusten muuttamisen prosessin suorituksen aikana Cockpit-käyttöliittymän kautta, jos esimerkiksi syntyisi tarve vaihtaa kutsuttavan palvelun osoitetta jo käynnistyneeseen prosessi-instanssiin.



Kuva 10. Osoitteiden ja kutsuparametrien määrittäminen prosessimuuttujiksi

Joitain viikkoja myöhemmin HTTP Connector-lisäosan esimerkeistä löydettiin keino tallentaa taustapalvelun vastauksen sisältö, statuskoodi ja header-tiedot prosessi-instanssiin käyttäen Connector outputs -kentässä getVariable -metodia (Kuva 11). REST-kutsujen tekemiseen käytettyjen asetuksten sekä saadun vastauksen tallentaminen prosessi-instanssille palveli kahta tarkoitusta. Ensinnäkin tiedot saatiin kirjoitettua lokiin, jolloin sitä kautta

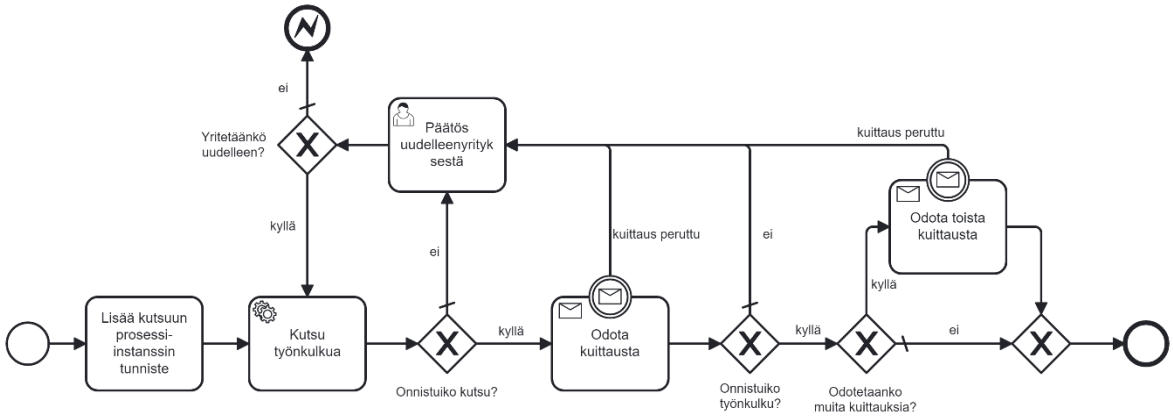
näkee parhaimmillaan yhdellä silmäyksellä koko prosessin kulun tarpeeksi tarkalla tasolla varmistuakseen onnistuneesta suorituksesta tai virhetilanteen ratkaisemiseksi. Toiseksi myös käyttöliittymän näkymä muuttui huomattavasti informatiivisemmaksi, sillä myös sieltä nähtiin nyt kutsuihin käytetyt asetukset ja saadut vastaukset.

Kuva 11. HTTP-kutsun vastauksen sisältämien tietojen tallennus prosessiin

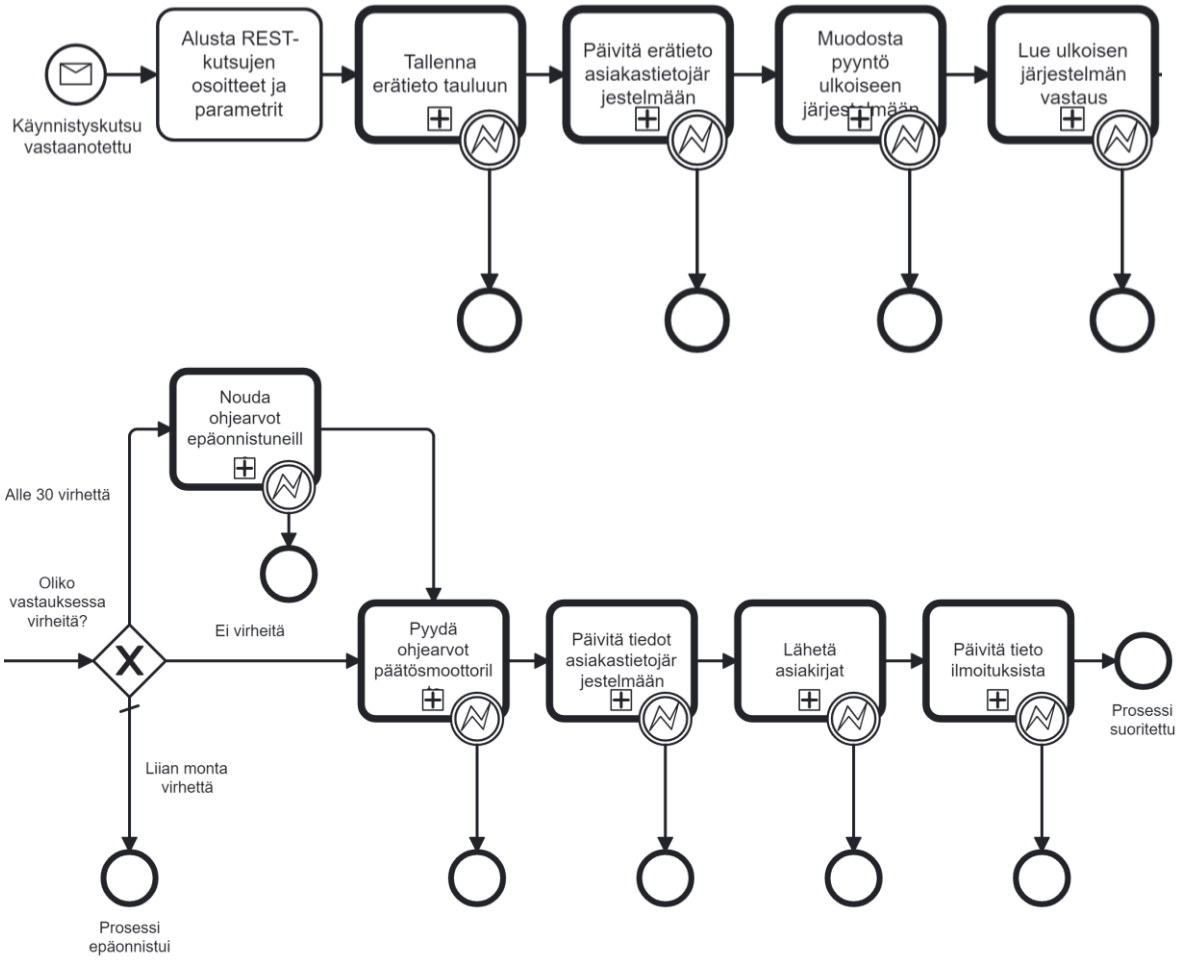
Tuotantoympäristössä testatessa nousi esiin tarve tehdä sama REST-kutsu taustapalveluun uudelleen vaikkapa jonkin taustapalvelussa tapahtuneen virhetilanteen ja sen korjauksen jälkeen. Toisin sanoen jo suoritettu tehtävä pitäisi siis suorittaa uudelleen, mutta nykyisessä prosessikaaviossa suoritus on jo edennyt odottamaan kuittausta tehtävän valmistumisesta. Ratkaisuna prosessikaavioon olisi voinut lisätä esimerkiksi jokaiseen viestin kuuntelutehtävään kylkeen toista viestiä kuuntelevan tapahtuman eli message boundary eventin. Tästä tapahtumasta voidaan vetää nuoli takaisin REST-kutsun tekevään tehtävään. Tällöin prosessin suoritusta voidaan ohjata tarvittaessa lähettämällä manuaalisesti viesti Camundan REST-viestirajapintaan. Tässä kohtaa päätettiin kuitenkin tehdä hiukan isompi muutos, jotta prosessikaaviosta saataisiin vähennettyä identtisiä toistuvia tehtäviä.

Kaavion 5 mukaisen prosessikaavion mallinnus aloitettiin ottamalla pohjaksi aiempi vain yksittäinen REST-rajapintakutsun taustapalveluun tekevä sekä vastausta kuunteleva tehtävä. Tavoitteena oli luoda yleiskäyttöinen aliprosessi, jota voisi käyttää prosessin jokaisessa vaiheessa. Eri vaiheiden erona on yleensä vain rajapintakutsun osoite sekä odotettavan kuittauksen nimi. Aliprosessille voidaan antaa nämä tiedot kutsuvassa prosessissa parametreinä ja tällä tavoin toisteisten muutosten tekeminen kaavioon vähenee. Koska HTTP Connector-lisäosasta oli saatu tässä vaiheessa kaapattua kutsun onnistumisesta kertova tilatieto, pystyttiin yksinkertainen virhehallinta rakentaa myös aliprosessiin. REST-rajapintakutsun tekemisen jälkeen kaaviossa seuraa risteys, jossa tarkistetaan vastauksen tilan kertova muuttuja. Jos muuttuja arvo on myönteinen, aloitetaan kuittauksen odottelu. Taustapalveluun tehdyn pienen uudistuksen myötä kuittauksen mukana saatiin jatkossa myös tieto työnkulun onnistumisesta. Tällekin tilatiedolle lisättiin vastaavasti risteys, jossa tarkistus tehdään. Onnistumisen myötä tarkistetaan, tarvitseeko odottaa vielä muita kuittauksia, koska yhdessä kohdassa prosessia kuittauksia tulee peräkkäin kaksi taustapalvelun sisäisen logiikan takia. Jos kuittauksia ei tule enempää, aliprosessi päättyy ja aliprosessia kutsuneen prosessikaavion suoritus jatkuu. Jos missään edellä mainituista tarkistuksista tilatiedon sisältävän muuttujan arvo puuttuu tai on kielteinen, siirrytään käyttäjätehtävään. Lisäksi jos kuittauksen odotus halutaan keskeyttää käyttäen aiemmin mainittua rajapintakutsua, siirrytään myös kyseiseen käyttäjätehtävään. Käyttäjätehtävä ilmestyy kuvassa 12 näkyvän Camundan Tasklist-tehtävälistaukseen graafiselle käyttöliittymälle ja tehtävään on linkki myös Cockpit-näkymässä. Tehtävään on liitetty yksinkertainen lomake, jolla valitaan, halutaanko taustapalvelussa epäonnistunutta operaatiota yrittää uudelleen. Käyttäjätehtävän suorituksen jälkeen prosessikaaviossa seuraa lomakkeen valinnan tarkastus. Myönteisen vastauksen tapauksessa prosessin suoritus palaa tekemään REST-kutsun taustapalveluun ja siitä eteenpäin vastaavasti kuin ensimmäiselläkin kerralla. Kielteinen vastaus

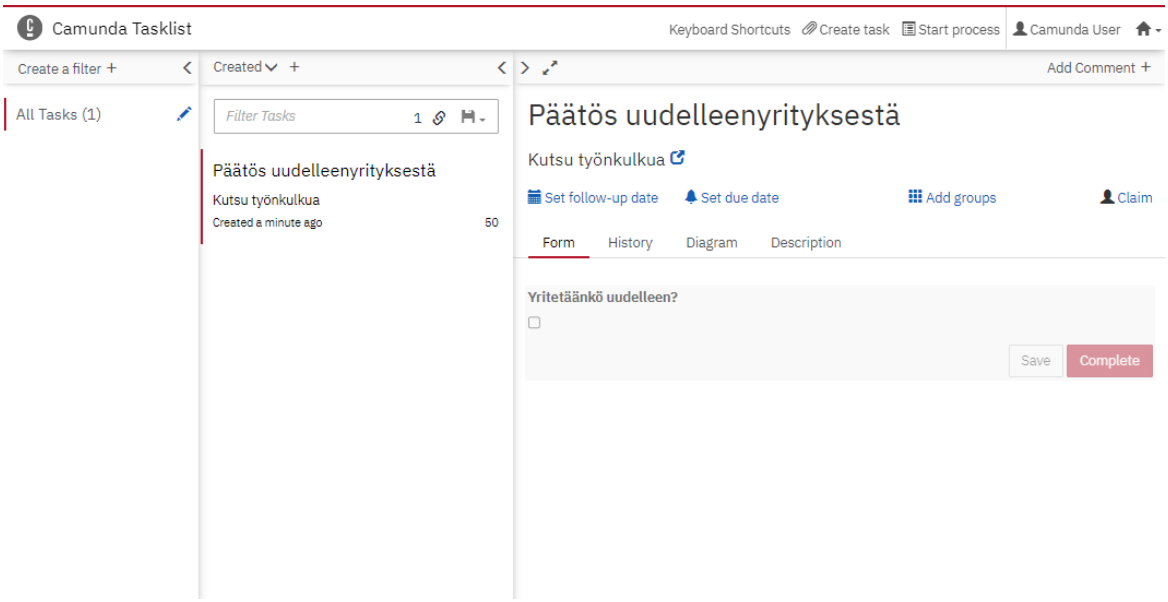
aiheuttaa aliprosessin päättymisen virhetapahtumaan, joka kaapataan kutsuvassa prosessissa aiheuttaen koko prosessi-instanssin päättymisen.



Kaavio 5. Taustapalvelun kutsumiseen käytetty aliprosessi



Kaavio 6. Lopullinen BPMN-prosessikaavio



Kuva 12. Käyttäjätehtävä avoinna Camunda Tasklist-näkymässä

Kun uusi aliprosessi upotettiin varsinaiseen pääprosessikaavioon, saatiin kaavion 6 mukainen lopputulos. Aliprosesseja merkitään tehtävälaatikoilla, joissa on reunustettu plusmerkki. Lisäksi aliprosessien kohdalle lisättiin reunustavat virhetapahtumien kuuntelijat, jotka aktivoituvat silloin kun aliprosessissa päätetään jättää uudelleenyritys tekemättä. Tällöin pääprosessin suoritus etenee kyseisen virhetapahtuman kohdalta suorituksen päättävään lopetustapahtumaan. Aliprosesseihin asetettiin myös parametrit, joilla niitä ohjataan tekemään halutut rajapintakutsut. Tuloksena syntynyt prosessikaavio vastaa ulkonäöltään pitkälti edeltävää versiota, mutta aliprosessien ansiosta jokaisessa vaiheessa on mukana tarkistuksia ja kevyt virhekäsittely. Tarkistuksien ja virhekäsittelyn sisällyttäminen aliprosesseihin säilytti pääprosessikaavion nopeasti hahmotettavana ja keskittyy näyttämään prosessin tärkeimmät vaiheet.

5.3 Vaihtoehtona tapausten hallinta

Vertailun vuoksi edellä kuvatusta automatisoidusta prosessista haluttiin tehdä versiointi käyttäen tapausten hallinnan kuvaustapaa ja toista mallinnustyökalua. Mallinnustyökaluksi valikoitui Flowable, sillä se on profiloitunut molempien standardien puolestapuhujaksi ja siten oli odotettavissa, että sovellus tukisi laajalti sen ominaisuuksia (Zickner 2023). Liiketoimintaprosessien hallintaan käytetyn Camundan käyttöäkin harkittiin, mutta tähän tarkoitukseen sitä ei nähty hyödylliseksi, sillä siitä tapaustenhallinnan tuki on poistumassa (Deehan 2020). Lisäksi Flowable tarjosi mahdollisuuden kuvata tapauksia ohjelmistoon sisältyvällä käyttöliittymällään. Kun työkalu oli valittu, pohdittiin mille tasolle tekninen toteutus olisi

tarkoituksenmukaista viedä. Kehitysprojektin alussa suunniteltiin, että vertailun voisi tehdä testiympäristössä rinnakkain käyttäen samoja taustapalvelun rajapintoja ja tällöin voitaisiin vertailla tuotantoympäristöä vastaavia skenaarioita. Liiketoimintaprosessin automatisoinnin jälkeen kuitenkin havaittiin, että testiympäristön pystyttäminen pilviympäristöön kaikkine tarvittavine konfiguraatioineen ei ole kuitenkaan oleellista vertailun toteuttamiseksi, vaan päätettiin priorisoida mallinnustavan, mallinnustyökalun ja päätösmoottorin käyttöön tutustumista. Tämä tarkoitti lopulta sitä, että tapaustenhallinnan ohjelmistolle päätettiin luoda vain paikallinen kehitysympäristö omalle työasemalle ja taustapalvelun rajapintojen kutsuminen korvattiin yksinkertaisilla käyttäjän kuitattavilla tehtävillä. Näin pystyttäisiin kuitenkin simuloimaan tapauksen kulkua ja eri vaiheiden keskinäistä riippuvuutta. Lisäksi tässä tapauksessa liiketoimintaprosessin tai tapauksen läpimenoaika ja siihen vaikuttava ohjelmistojen suorituskyky ei ollut olennaisessa osassa, joten tällainen yksinkertainen toteutus olisi riittävä vertailuun. Kehitysympäristö luotiin Docker Composen avulla käyttämällä pohjana aiemmin kehitysprojektissa luotua kuvausta ja muokkaamalla se kuvan 13 mukaiseksi Flowablen tarpeisiin.

```yaml
version: '3.9'
services:
  flowable:
    build: ./flowable
    environment:
      - SERVER_PORT=8080
      - SPRING_DATASOURCE_DRIVER-CLASS-NAME=com.microsoft.sqlserver.jdbc.SQLServerDriver
      - SPRING_DATASOURCE_URL=jdbc:sqlserver://mssql:1433;databaseName=flowable;encrypt=false
      - SPRING_DATASOURCE_USERNAME=sa
      - SPRING_DATASOURCE_PASSWORD=Flowable_demo123
    ports:
      - 8080:8080
    depends_on:
      - mssql
    networks:
      - flowable_network

  mssql:
    image: mcr.microsoft.com/mssql/server:2019-latest
    environment:
      - ACCEPT_EULA=Y
      - SA_PASSWORD=Flowable_demo123
    healthcheck:
     test: ["CMD-SHELL", "/opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P Flowable_demo123 -Q 'SELECT 1' || exit 1"]
     interval: 10s
     retries: 10
     start_period: 10s
     timeout: 3s
    networks:
      - flowable_network

  mssql.configurator:
   image: mcr.microsoft.com/mssql/server:2019-latest
   volumes:
     - "./mssql:/tmp/mssql-scripts"
   depends_on:
     mssql:
       condition: service_healthy
   command: >
     bash -c '
     /opt/mssql-tools/bin/sqlcmd -S mssql -U sa -P Flowable_demo123 -i /tmp/mssql-scripts/create-db.sql -C;
     echo "Command finished";
     '
   networks:
     - flowable_network

networks:
  flowable_network:
    driver: bridge
```

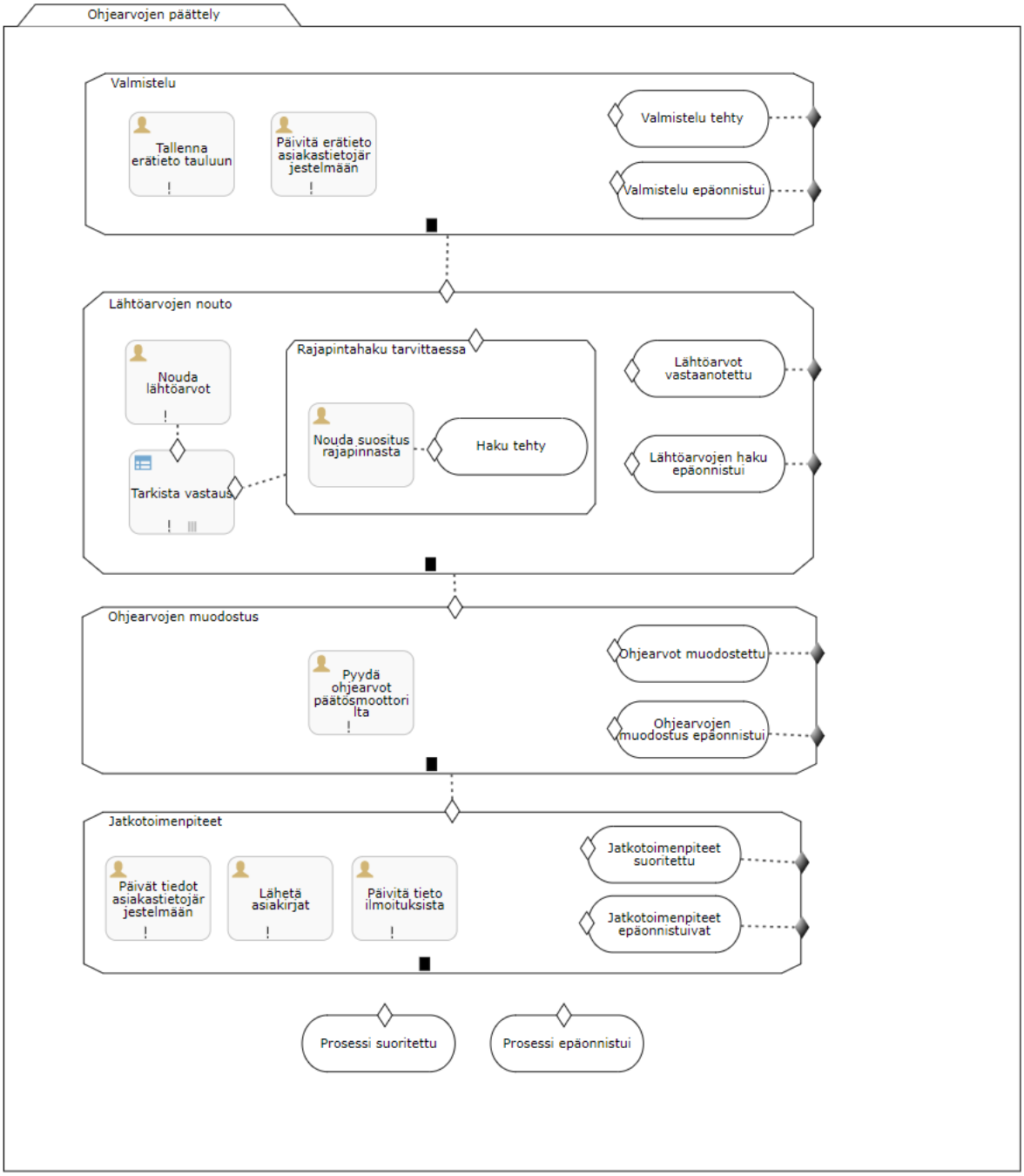
Kuva 13. Flowablen kehitysympäristön palvelut ja asetuksia

Tapausten hallinnan termistössä ei varsinaisesti ole prosesseja vaan tapauksia, joten mallinnus aloitettiin luomalla uusi tapaus. Tyhjää tapausta kuvataan salkkumaisella kuviolla, jonka sisään tehtäviä ja muuta sisältöä aletaan lisätä. Salkun sisään lisättiin tyhjä käyttäjätehtävä, jonka tarkoituksena oli kuvastaa taustapalveluun tehtävää rajapintakutsua. Kun käyttäjä kuittaa tehtävän, ikään kuin oletetaan että rajapintakutsu ja sen taustalla oleva operaatio on suoritettu. Käyttäjätehtävään lisättiin lomake, jonka kautta tehtävää kuitatessaan käyttäjä voi valita onnistuiko operaatio vai ei. Tällä tavalla pystyttiin kokeilemaan myös mitä tapahtuisi, jos tapaukselle palautuisikin virheilmoitus. Edellä kuvatun kaltaisia käyttäjätehtäviä lisättiin tapauksen sisään kaikille prosessiin sisältyneille operaatioille ja tässä vaiheessa olisi tuntunut loogiselta lisätä ensimmäistä tehtävää lukuun ottamatta kaikkiin

tehtäviin vahdit, eli entry sentryt, ja ketjuttaa tehtävät assosiaatioviivoilla. Tällöin tapaus olisi ollut käytännössä identtinen aiemmin rakennetun BPMN-muotoisen kaavion kanssa ja tapausten hallinnan hyödyt jäisivät saamatta. Toisaalta tässä tapauksessa ratkaisu olisi voinut teknisesti toimia.

Kuvaustavan opiskelun ja pohdinnan jälkeen tapauksen sisältämät tehtävät jaettiin vaiheisiin niiden välisen riippuvuuden pohjalta. Vaiheita kuvataan CMMN-kuvaustavassa stage-elementillä, jonka sisään voi lisätä vaiheeseen kuuluvat tehtävät. Osa tehtävistä pitää suorittaa ennen kuin tapauksessa voidaan edetä ja toisaalta jotkin tehtävät voi joutua suorittamaan useampaan kertaan. Tapaukselle syntyi neljä eri vaihetta. Valmisteluvaiheessa on kaksi tehtävää, jotka kuittaamalla tapaukselle tallentuu tehtävän lomakkeelta muuttujat, jotka kertovat onnistuivatko tehtävät. Vaiheeseen sisältyy myös milestone-elementit, jotka kertovat onnistuiko vai epäonnistuiko vaihe. Niiden vahdeissa eli entry sentryissä tarkistetaan tehtävien muuttujat ja sen mukaan joko onnistumisen tai epäonnistumisen milestone aktivoituu ja tällöin tapaukselle tallentuu tieto sen aktivoitumisesta. Seuraavan vaiheen aktivoituminen riippuu edellisen vaiheen onnistumisen milestonen aktivoitumisesta. Mikäli epäonnistumista kuvaava milestone aktivoitui, ohitetaan loput vaiheet ja siirrytään tapauksen vaiheiden ulkopuoliseen milestoneen, joka kertoo koko tapauksen epäonnistuneen. Jos taas onnistumisen milestone aktivoitui, siirrytään seuraavaan vaiheeseen. Toisessa vaiheessa on jälleen taustapalveluun tehtävää rajapintakutsua kuvastava käyttäjätehtävä. Tehtävässä on kuitenkin poikkeava lomake, jonka kautta voi antaa haluamansa numeroarvot ikään kuin taustapalvelun palauttamina arvoina. Nämä tulokset tarkistetaan seuraavassa vaiheessa päätösmoottorin avulla. Lyhyesti kuvattuna päätösmoottori antaa annettujen arvojen pohjalta jonkin neljästä eri muuttujan arvosta, jotka kertovat olivatko arvot ok, lähes ok, ei ok tai yritetäänkö arvoja hakea uudelleen. Jos arvoja täytyy noutaa uudelleen, aktivoituu vaiheen sisällä vielä sisäkkäinen vaihe, joka kuvastaa valinnaista vaihetta. Tämän alivaiheen sisällä on edeltävän kanssa identtinen käyttäjätehtävä, jonka lomakkeella voi antaa uudet numeroarvot. Kun tehtävä on kuitattu, alivaihe päättyy ja arvot tarkastetaan uudelleen päätösmoottorilla. Päätösmoottorin päättelytehtävä voidaan siis suorittaa useampaan kertaan tapauksen suorituksen aikana. Mikäli arvot ovat nyt hyväksytyissä rajoissa, vaiheen onnistumista kuvastava milestone aktivoituu. Jos taas arvot eivät olleet hyväksyttäviä, epäonnistumista kuvastava milestone aktivoituu ja tapaus päättyy. Onnistumisen tapahtuessa tapauksessa aktivoituu vielä peräkkäin kaksi eri vaihetta, jotka ovat samanlaisia kuin ensimmäinen vaihe. Kolmannessa vaiheessa on yksi tehtävä, joka pitää tehdä ennen neljättä vaihetta. Kun neljännen vaiheen onnistumista kuvastava milestone aktivoituu, koko tapauksen onnistumista kuvastava milestone aktivoituu ja tapaus on valmis. Koko tapauksen onnistumista ja epäonnistumista kuvastavien milestone-elementtien olemassaolo ei ole

välttämätöntä, mutta liiketoimintaprosessin piirtämiseen tottuneelle on helpompaa hahmottaa tapauksen lopputulos nopeammin niiden ollessa tapauksen sisällä. Tapauksesta rakentui kaavion 7 mukainen tapauskaavio.



Kaavio 7. Lopullinen tapaussuunnitelma

5.4 Päätösmoottorin hyödyntäminen

Päätösmoottori sisältyy molempiin valittuihin ohjelmistoihin, ja kuten aiemmin todettua, toimii niissä molemmissa DMN-kuvauskielellä toteutettujen päätöstaulujen suoritusvälineenä. Varsinaisessa automatisoidussa liiketoimintaprosessissa ei päädytty vielä hyödyntämään DMN:ää tai päätöstauluja vaan työssä tutustuttiin DMN:n kuvauskieleen ja siihen, miten kuvauskielen käyttö on toteutettu eri sovelluksissa. Päätöstauluja voi kuvata sekä Flowablen käyttöliittymän työkaluilla, että Camundan tuottamalla Camunda Modeler -mallinnusohjelmalla. Molemmat toimivat pitkälti samaan tapaan ja erot ovat paljolti kosmeettisia. Kehitysprojektissa rakennettiin toimivat esimerkit kahdesta eri päättelystä.

Ensimmäisessä esimerkissä haluttiin kokeilla yksinkertaisempaa päättelyä: Miten Camundalla automatisoidussa liiketoimintaprosessissa yksinkertaisena risteyksenä toteutettu rajapintakutsun palauttaman kokonaisrivimäärän ja epäonnistuneiden tapausten rivimäärän tarkastus toimisi päätöstauluna. Tämä päättely tehtiin Flowablella tapaustenhallinnan esimerkkiin päätöstauluna. Päätöstaululle tuodaan kokonaisrivimäärä sekä epäonnistuneiden tapausten rivimäärä sisään tulevina muuttujina ja ensimmäisen osuman valintaa käyttämällä valitaan tulosmuuttujan arvo. Mahdollisia yhdistelmiä on määritelty neljä kappaletta. Kuvan 14 päätöstaulu integroituu tapaukseen lisäämällä päätöstehtävä tapauksen sisään ja asettamalla viittaus luotuun päätöstauluun. Päättelyn jälkeen tulosmuuttujan arvo on käyttävissä tapauksen kontekstissa. Lopputuloksena syntyi aiemmin rakennettuun kaaviossa 7 näkyvään tapauskaavioon toimiva esimerkki myös päätösmoottorin hyödyntämisestä.

tarkistaVirheidenMaara

F	rowCountTotal rowCountTotal *number*		rowCountFailed rowCountFailed *number*		vastausStatus vastausStatus *string*
1	>	0	==	0	ok
2	>	0	<=	30	osittainOk
3	>	0	>	30	eiOk
4	==	0	==	0	eiOk

Kuva 14. Tilan päättelyä päätöstaulun avulla

Toisena esimerkkinä haluttiin kokeilla voisiko, ja kuinka helppoa, taustapalvelulta olisi siirtää päättelylogiikkaa päätösmoottorille. Tämä esimerkki toteutettiin ensin Flowablella. Erona ensimmäiseen esimerkkiin tässä tapauksessa sisään tulevaa lähtöarvoa täytyisi muuttaa kertomalla ja summaamalla päättelyn eri vaihtoehtoja tarkastaessa. Yksi

vaihtoehto olisi tehdä kyseiset laskutoimenpiteet etukäteen ja luomalla jokaisen laskutoimituksen tuloksesta erillinen sisään tuleva muuttuja, mutta se nähtiin melko sekavana ja monimutkaisena tapana. Lopulta päättelyn eri vaihtoehdot toteutettiin hyödyntämällä Flowablen päätösmoottorin tukemaa skriptikieltä hyväksikäyttäen. Tällöin menetetään hieman taulukon luettavuudesta, sillä sisään tulevan muuttujan nimen ei enää tarvitse täsmätä varsinaisessa päättelyssä käytettyjen muuttujien nimeä ja lisäksi rivien ulkonäkö muuttuu hieman. Lopputuloksena saatiin kuitenkin toimiva päättely. Samaa esimerkkiä haluttiin kokeilla vielä myös Camundan Modeler-piirtotyökalulla luotuna. Siinä halutut laskutoimenpiteet on mahdollista tehdä sisään tulevan muuttujan otsikkokentän ohjelmakoodikentässä, jolloin laskutoimitukset ovat nähtävillä taulukossa ja taulukon ulkonäkö pysyy hieman selkeämpänä kuin Flowablella luodussa esimerkissä. Samalla huomattiin, että Flowable tukee JUEL-skriptikieltä ja Camunda sen lisäksi muitakin, mikä on yksi huomioon otettava seikka esimerkiksi siirrettäessä päätöstauluja suoritettavaksi sovelluksesta toiseen. Molemmat esimerkit testattiin teknisesti toimiviksi paikallisessa kehitysympäristössä syöttämällä erilaisia lähtöarvoja ja tulostamalla lopputulos lokiin.

6 Yhteenveto

Opinnäytetyössä pyrittiin selvittämään ensinnäkin, millaisia hyötyjä prosessiautomaation välineistä on liiketoimintaprosessin automatisoinnissa. Tässä tapauksessa osa prosessin automatisoinnin taustatyöstä oli valmiiksi tehty, sillä tarpeelliset vaiheet, ja niiden järjestys, oli selvitetty ja määritetty jo edellisenä vuonna. Erilaisten vaiheiden tarpeellisuus ja niiden keskinäisten riippuvuuksien selvittäminen voi aiheuttaa merkittävästi työtä ennen varsinaista teknistä toteutusta. Usein tällainen määritys- ja selvitystyö voi jo itsessään olla hyvin arvokasta, sillä se saattaa nostaa esiin tavoitteen kannalta turhia vaiheita tai odottelua aiheuttavia riippuvuuksia, joita Lean-ajattelussa kuvataan termillä hukka. Tässä kehitysprojektissa voitiin oikeastaan teknisen kehitysympäristön valmistelun jälkeen aloittaa saman tien suoritettavan prosessikaavion piirtäminen. Selkein ja konkreettisin hyöty prosessin automatisoinnista on manuaalisen työn väheneminen. Aiempaan verrattuna yksittäisten vaiheiden jälkeen ei tarvitse manuaalisesti jatkaa prosessia, jolloin käynnissä olevaa prosessia seuranta poistuu ja vapauttaa aikaa ja huomiota muihin tehtäviin. Prosessin automatisointi heijastui myös taustapalvelujen kehitykseen, sillä prosessin kuvaaminen ja siirtäminen prosessimoottorin suoritettavaksi herätti tarpeen muutoksille taustapalvelun toiminnassa. Kun ihminen ei ole enää jokaisessa prosessin vaiheessa tarkistamassa prosessin vaiheita, tarvittiin virheiden käsittelyyn ja suoritettujen toimenpiteiden kirjanpitoon parannuksia.

Toisena tutkimuskysymyksenä selvitettiin, mitä tuotantokäyttöön soveltuvan prosessimoottorin käyttöönotto vaatii. Teknisen ympäristön muutamista haasteista huolimatta prosessimoottorin valmistelussa onnistuttiin ja tarvittavat vaiheet saatiin suoritettua ja dokumentoitua. Lisäksi tuotantokäytön alkamisen ajankohdan siirtyminen parilla viikolla antoi aikaa suorittaa BPMN-kuvaustavalla mallinnettu liiketoimintaprosessille testaus myös tuotantoympäristössä.

Kolmantena tutkimuskysymyksenä tutkittiin millaisia ominaisuuksia BPMN-standardille vaihtoehtoinen CMMN-standardi tuo automatisoitavaan prosessiin. Tutkimukseen käytettiin CMMN-standardin kokeiluun paremmin soveltuvaa ohjelmistoa. Tapauksenhallinnan kuvaustavan perusasiat tulivat tutuiksi ja esimerkkitapauksesta saatiin toteutettua toimiva, melko yksinkertainen kehitysversio. Toteutuksen aikana alettiin nähdä jo piirteitä siitä millaisissa kohteissa liiketoimintaprosessien automaatio ja BPMN toimii kenties paremmin kuin CMMN ja toisinpäin. BPMN vaikuttaa loistavan selkeästi etukäteen määritettävissä olevien prosessien automatisoinnissa, kun taas CMMN:n hyödyt tulevat esiin epämääräisemmissä tapauksissa, joissa ihmisten välinen vuorovaikutus on isossa osassa ja tehtävien järjestystä ei aina tunneta. Esimerkiksi asiakaspalvelun tai myynnin prosessien kuvaamisessa CMMN voisi toimia BPMN:ää paremmin. Toteutettua tapausta on mahdollista jatkokehittää ja

toisaalta käyttää toiminnallisena esimerkkinä, kun esimerkiksi pohditaan jatkossa erilaisten prosessien automatisaatiota.

Neljäntenä tutkimuskysymyksenä selvitettiin millaisia hyötyjä päätösten hallinta DMN-standardin ja kuvaustavan avulla voisi tuoda liiketoimintaprosesseihin tai tapausten hallintaan. Kehitysprojektiin valituissa prosessiautomaation ohjelmistoissa on DMN-standardia hyödyntävä päätösmoottori sisäänrakennettuna. Kehitysprojektissa toteutettiin kahdesta päättelystä päätöstaulut. Ensimmäinen tapaus oli harjoituksenomainen yksinkertainen rivimäärien pohjalta päättely, jonka toteutus onnistuu toisaalta hyvin myös BPMN- tai CMMN-kuvauskielien omilla työkaluilla. Tällaisen tapauksen toteuttaminen erikseen päätöstauluna ei siten todellisessa tilanteessa ole kovin tarkoituksenmukainen, mutta toimi tässä tapauksessa hyvänä harjoituksena. Monimutkaisemmat laskutoimitukset vaativat enemmän tutkimusta ja perehtymistä esimerkiksi ohjelmistojen skriptikielitukeen, sillä muuttujien arvojen dynaaminen muokkaaminen päätöstaulun sisällä vaati monimutkaisempaa toteutusta. Tässä ohjelmistojen erot tulivat hyvin esiin sekä teknisesti että visuaalisesti. Flowable tukee vain JUEL-skriptikieltä, kun taas Camunda useita eri vaihtoehtoja pysyen kuitenkin visuaalisesti selkeämpänä. DMN-kuvaustavan tutkimusta ei työn aikana otettu osaksi tuotantokäyttöön julkaistua prosessiautomaatiota, sillä automatisoituun osuuteen ei sisältynyt päättelyitä, jotka olisivat hyötyneet päätösten hallinnan käytöstä. Kuvaustapaa ja päätösten automatisoinnin sisällyttämistä prosesseihin tutkittiin siis teoriatasolla jatkokehitystä ajatellen.

Kaiken kaikkiaan kokeilluista sovelluksista kumpikin soveltuu prosessien automatisointiin ja automatisoituun päätöstentekoon. Flowable taas tarjoaa lisäksi tuen tapausten hallinnalle. Tulevaisuutta ajatellen kuitenkin täytyy ottaa huomioon useita muitakin seikkoja, kuten tietoturva- ja ominaisuuspäivitysten saatavuus, maksullisen lisenssin hinnoittelu sekä sen sisältämät ominaisuudet ja rajoitteet.

7 Pohdinta

Lopputuloksena syntyneestä artefaktista saatiin konkreettista hyötyä automatisoidun prosessin toimiessa tuotantoympäristössä. Tällainen automatisoitu prosessi on oppikirjamainen esimerkki Lean-ajattelun hukan poistosta. Kehitysprojektin aikana tuotantoympäristössä ehdittiin ajamaan pienempien testiajojen lisäksi neljä suorituskertaa siten että taustapalvelut käsittelivät vastaavankokoista aineistoa kuin aiemmin manuaalisesti prosessia suoritettaessa. Haastattelun perusteella tällaisten ajojen suorittaminen oli kestänyt noin puolitoista työpäivää, kun vaiheita oli käynnistetty ja tuloksia tarkastettu muiden työtehtävien ohessa. Automatisoidulla prosessilla vastaava toimenpidesarja kesti viidestä kuuteen tuntia suorituskertaa kohti, jonka jälkeen päästiin tarkastamaan tulokset. Ajoittain suorituksessa saattaa toki syntyä virhetilanteita, jotka aiheuttavat prosessiin manuaalisesti tehtävää selvitystyötä ja korjausta. Näitä pyrittiin ehkäisemään sekä testaamalla toimintaa testiympäristössä, että ottamalla taustapalveluille tuotantoympäristössä pienempiä aineistoja käsittelyyn ensimmäisillä suorituskerroilla.

Tapausten hallinta ja CMMN vaikuttivat toimivilta vaihtoehdolta BPMN:lle tietyissä tilanteissa, mutta kehitysprojektin pitkälti lineaarinen prosessi ja tekijän aiempi kokemus vain BPMN:stä ei saanut niiden kaikkia vahvuuksia esiin. Vaihtoehtoisten kuvaustapojen ja standardien opiskelu vie aikaa sekä prosessin tai tapauksen automatisoinnin tekijältä, että myös muilta sidosryhmiltä, ja lisäksi BPMN:llä on mahdollista luoda teknisesti vastaavia ratkaisuja kuin CMMN:llä. Samat argumentit toimivat toki myös toiseen suuntaan, koska mielestäni prosessin automatisointi olisi onnistunut myös valituilla tapausten hallinnan työkaluilla. Olennaista onkin kartoittaa, millaisia työkaluja ja osaamista automaation kehittämiseen ja ylläpitoon organisaatiossa on jo olemassa, millaisia prosesseja ja tapauksia pitäisi tulevaisuudessa automatisoida, ja sen pohjalta valita käytettävät standardit ja ohjelmistot.

Päätöstaulut ja DMN vaikutti toimivan molemmissa kehitysprojektissa käytetyissä ohjelmistoissa muutamin pienin eroavaisuuksin visuaalisessa esitystavassa ja esimerkiksi monimutkaisempien päättelyjen tekoon käytettävien ohjelmointikielien valikoimassa. Päätöstaulujen luominen on teknisesti nopea prosessi ja lopputulos on todennäköisesti helpommin ymmärrettävä laajemmalle joukolle kehitysprojektin sidosryhmästä kuin jollakin ohjelmointikielellä toteutettu vastaava logiikka. Monimutkaisempien päättelyjen kuvaamiseen päätöstaulua ja -moottoria pidettiin varteenotettavana vaihtoehtona prosessin jatkokehitystä ajatellen.

7.1 Haasteet

Tapaustenhallinnan osalta uuden tapauksen luonti vaikutti helpolta, sillä tapauksen sisältyvät tehtävät olivat samoja mitä aiemmin kuvattiin lineaariseen liiketoimintaprosessiin.

Toisaalta taas haasteita alkuun toi yritys oppia pois lineaarisesta ajattelusta ja päästä hyödyntämään tapaustenhallinnan ominaisuuksia. Uuden kuvaustavan ja ohjelmiston teknisten yksityiskohtien opettelu käytännössä vei aikaa. Lisäksi haastetta toi pilviympäristön käyttö organisaatioympäristössä, sillä parhaiden käytäntöjen mukainen uusien palvelujen käyttöönotto vaatii osaltaan riittävän määrän suunnittelua, määritystä, testausta ja dokumentointia.

Testattujen ohjelmistojen käyttö etenkin jatkokehityksen kannalta voi aiheuttaa haasteita tai ainakin vaatii lisää tarpeiden kartoittamista lisensointimallien takia. Molemmista testatuista ohjelmistoista kehitysprojektissa käytetyt versiot ovat käytettävissä jopa kaupalliseen käyttöön, mutta päivitystukea Camundan ilmaisversiolle on jäljellä enää suhteellisen lyhyen aikaa syksyyn 2025 ja Flowablen tuki taas riippuu kehittäjäyhteisöstä (Holmes-Higgin 2020; Thengvall 2024). Molemmista on myös julkaistu uudempi versio: Camundasta versio 8 ja Flowablesta versio 7, joihin uudet ominaisuudet suunnataan. Tuoreemmissa versioissa ohjelmistojen käyttöä on pyritty eri tavoin ohjaamaan siten että kaupallisessa käytössä maksullisesta versiosta tulisi houkuttelevampi, mikä myös tekee vaihtoehtojen vertailusta hankalaa. Camundan tuoreinta versiota voi käyttää omassa ympäristössä (self-managed) kehitys- ja testikäytössä ilmaiseksi ja lisäksi selainpohjaisella prosessikaavioiden mallinnustyökalulla voi luoda kaavioita ilman kustannuksia. Kaupallinen käyttö itse ylläpidetyllä prosessimoottorilla tai minkäänlainen prosessien suorittaminen Camundan pilvipalvelun kautta vaatii maksullisen tilauksen. Avointa lähdekoodia versioon 7 asti edustanut prosessimoottori ei Camundan omien sanojensa mukaan myöskään enää versiossa 8 ole varsinaisesti avointa lähdekoodia, vaikka lähdekoodi onkin saatavilla. Flowable taas on tuoreimmassa versiossaan 7 edelleen lisensoitu avoimena lähdekoodina. Kehitysprojektissa käytetyn version 6 mukana tulleet graafiset prosessikaavioiden, tapausten ja päätöstaulujen mallinnustyökalut sekä muut käyttöliittymäkomponentit on poistettu uudemmasta versiosta ja vain prosessien, tapausten ja päätösten moottorit tarjotaan avoimesti ja ilmaiseksi. Käyttäjälle jää siis pohdittavaksi rakentaako itse käyttöliittymän rajapintojen päälle, voisiko pärjätä ilman, vai hankkiako maksullinen lisenssi mukana tulevine graafisine työkaluineen. Myös jonkinlainen ”sekoitus” voisi olla mahdollinen, jossa prosessikaaviot piirrettäisiin esimerkiksi Camunda Modeler-editorilla, mutta suoritus tapahtuisi Flowablen prosessimoottorilla. Yhteisen standardin myötä tämä skenaario olisi mahdollinen, mutta eri ohjelmistojen eroavaisuudet esimerkiksi tuettujen BPMN- tai CMMN-elementtien, skriptikielien tai ohjelmistokohtaisten laajennusten osalta saattavat aiheuttaa hankaluuksia. BPMN 2.0 -standardi määrittelee, että sitä noudattelevien kaavioiden visuaalisen ja liiketoimintalogiikan sisältävän sisällön voi tallentaa XML-muodossa ja siirtää järjestelmästä toiseen (Object Management Group 2013). Kuitenkin standardinmukaisuus vaihtelee ohjelmistoista toiseen ja esimerkiksi

Camundan ja Flowablen dokumentaatiosta löytyy sivut, jotka luettelevat miltä osin ne BPMN 2.0 -standardia noudattavat. Camunda Modeler -editorissa ei myöskään ole jatkossa tukea CMMN:lle, joten se rajaisi tapausten hallinnan pois.

7.2 Jatkokehitys

Kuten aiemmin mainittiin, kehitysprojektissa käytettiin sovellusten tuettuja, mutta ei tuoreimpia versioita. Tulevaisuudessa täytyy ottaa huomioon siirtymä uuteen sovellusversioon tai mahdollisesti kokonaan toiseen sovellukseen. Tällöin täytyy punnita ainakin ovatko maksullisen lisenssin hyödyt hinnan arvoisia. Vaikka sovellukset toteuttavatkin yhteisen standardin määrittelemiä prosessikaavioita, tapauksia ja päätöstauluja, eri sovellukset eivät kuitenkaan todennäköisesti toimi ristiin siten että tietylle sovellukselle luotuja kaavioita voisi suorittaa toisella sovelluksella täysin ilman muutoksia. Sovellusten toimittajilla onkin ohjeistuksia ja jopa valmiita työkaluja esimerkiksi prosessikaavioiden tai jopa prosessien suoritushistorian muuntamiseen heidän omille tuotteilleen. (Pappas 2020; Flowable b.)

Yhtenä teknisenä jatkokehityskohteena voisi olla sovelluslokituksen tallennus pysyvämmin siten, ettei se häviä kontin käynnistyessä uudelleen ja jotta se olisi käytettävissä myöhempänä ajankohtana. Lisäksi testiympäristön palvelut voisi kuvata muiltakin osin, kuin vain prosessimoottorin kontin osalta, asennusputken määritystiedostoihin vastaavaan tapaan kuin tuotantoympäristö. Näin varmistettaisiin testiympäristön nopea uudelleenasennus toimivaksi kokonaisuudeksi ongelmatilanteissa. Missään ympäristössä ei kehitysprojektissa otettu myöskään käyttöön varmistuksia, kuten esimerkiksi varmuuskopiointia. Tässä tapauksessa prosessit ovat suhteellisen lyhyitä, muutamien tuntien mittaisia, ja niiden yksityiskohtaista historiaa ei todennäköisesti tulla tarvitsemaan laajalti tulevaisuudessa. Tulevissa kehitysprojekteissa tällaiset seikat pitää kuitenkin arvioida ja ottaa huomioon. Kehitysprojektin lopuksi havaittiin sovelluksessa hieman päällekkäisyyttä, sillä prosessimoottorin käyttöliittymän näkymät ja sovelluslokitus tuottavat hyvin samanlaista informaatiota prosessin suorituksen tapahtumista eri esitystavoilla visuaalisesti tai tekstimuodossa. Jatkossa lokituksen voisi keskittää prosessimoottorin sisäisten teknisten tapahtumien kirjaamiseen teknisten virhetilanteiden ratkaisun avuksi ja käyttöliittymän näkymät liiketoiminnallisten tapahtumien visualisointiin.

Liiketoiminnallisesti ajatellen yksi kehityskohteista olisi laajentaa näkökulmaa BPA:sta BPM:n suuntaan ja selvittää, voisiko prosessin automatisointia laajentaa alku- tai loppupäästä. Liiketoiminnallisen prosessin suoritusaikaa voisi mitata sekä prosessimoottorin tuottaman datan avulla, että muilla keinoin automatisoidun osuuden ulkopuolella. Prosessin kuvaaminen nykyistä laajemmin, ja tilastotietojen kerääminen, voisi paljastaa pullonkauloja ja auttaa osaltaan poistamaan hukkaa kokonaisuudesta. Lisäksi liiketoiminnallisen

prosessin päättelyitä voisi olla mahdollista siirtää päätösmoottorille ja siten automatisoida myös erilaisia tarkistuksia ja kaavamaisia päätöksiä. Tällöin pitäisi selvittää, saadaanko kaikki päättelyyn tarvittavat lähtötiedot tuotua riittävän helposti päätösmoottorin käyttöön. Jos tiedot saataisiin tuotua, päätökset voitaisiin suorittaa automatisoidun prosessin osana eikä niitä tarvitsisi tehdä ihmisvoimin, tai ohjelmoimalla niitä taustapalveluihin.

Lähteet

Alfresco. 2017. Getting Started with Alfresco Process Services. Viitattu 10.9.2024. Saatavissa https://hub.alfresco.com/t5/alfresco-process-services/getting-started-with-alfresco-process-services/ba-p/290016

Anicas, M. 2014. Java Keytool Essentials: Working with Java Keystores. DigitalOcean. Viitattu 9.9.2024. Saatavissa https://www.digitalocean.com/community/tutorials/java-keytool-essentials-working-with-java-keystores

Athuraliya, A. 2023. What is BPMN? The Easy Guide to Business Process Modeling Notation. Creately. Viitattu 9.9.2024. Saatavissa https://creately.com/guides/what-is-bpmn/

Athuraliya, A. 2024. The Complete List of BPMN Symbols and Their Meanings. Creately. Viitattu 29.9.2024. Saatavissa https://creately.com/guides/bpmn-symbols/

Belcic, I. & Stryker, C. 2024. What is business process modeling and notation (BPMN). IBM. Viitattu 9.9.2024. Saatavissa https://www.ibm.com/blog/bpmn/

Brown, L. 2022. Introducing IBM Process Automation Manager Open Edition. IBM. Open Editions -blogi. Viitattu 9.9.2024. Saatavissa https://community.ibm.com/community/user/automation/blogs/lori-brown/2022/07/27/introducing-ibm-process-automation-manager-open-ed

Camunda a. BPMN Primer. Viitattu 29.9.2024. Saatavissa https://docs.camunda.io/docs/components/modeler/bpmn/bpmn-primer/

Camunda b. Camunda 7 Run. Viitattu 10.9.2024. Saatavissa https://docs.camunda.org/manual/7.20/user-guide/camunda-bpm-run/

Camunda c. Error Events. Viitattu 10.9.2024. Saatavissa https://docs.camunda.org/manual/7.20/reference/bpmn20/events/error-events/

Camunda d. Introduction, Viitattu 10.9.2024. Saatavissa https://docs.camunda.org/manual/7.21/introduction/

Camunda e. Logging. Viitattu 10.9.2024. Saatavissa https://docs.camunda.org/manual/7.20/user-guide/logging/

Camunda f. What is Camunda 8. Viitattu 10.9.2024. Saatavissa https://docs.camunda.io/docs/components/concepts/what-is-camunda-8/

Camunda g. DMN Tutorial. Viitattu 29.9.2024. Saatavissa https://camunda.com/dmn/

Codecademy. What is REST. Viitattu 10.9.2024. Saatavissa https://www.codecademy.com/article/what-is-rest

Deehan, N. 2020. How CMMN never lived up to its potential. Camunda-blogi. Viitattu 10.9.2024. Saatavissa https://camunda.com/blog/2020/08/how-cmmn-never-lived-up-to-its-potential/

Docker a. Docker Compose overview. Viitattu 9.9.2024. Saatavissa https://docs.docker.com/compose/

Docker b. Dockerfile reference. Viitattu 9.9.2024. Saatavissa https://docs.docker.com/reference/dockerfile/

Docker c. How Compose works. Viitattu 9.9.2024. Saatavissa https://docs.docker.com/compose/compose-application-model/

Docker d. Use containers to Build, Share and Run your applications. Viitattu 9.9.2024. saatavissa https://www.docker.com/resources/what-container/

Docker e. What is an image. Viitattu 9.9.2024. Saatavissa https://docs.docker.com/guides/docker-concepts/the-basics/what-is-an-image/

Eisner, M. 2020. Intro to Case Management Model and Notation (CMMN). Viitattu 10.9.2024. Saatavissa https://www.processmaker.com/blog/intro-to-case-management-model-and-notation-cmmn/

Emsbach, R. 2020. Anyone can run Camunda BPM on Azure in 10 Minutes. Viitattu 10.9.2024. Saatavissa https://camunda.com/blog/2020/05/anyone-can-run-camunda-bpm-on-azure-in-ten-minutes/

Flowable a. CMMN 1.1. Viitattu 29.9.2024. Saatavissa https://www.flowable.com/open-source/docs/cmmn/ch06-cmmn/

Flowable b. Flowable LEAP. Viitattu 15.9.2024. Saatavissa https://www.flowable.com/leap-bpm-migration

Flowable c. Flowable Pricing Plans + Support. Viitattu 10.9.2024. Saatavissa https://www.flowable.com/pricing

Flowable d. The Flowable Project. Viitattu 10.9.2024. Saatavissa https://www.flowable.com/open-source

Freund, J. & Rücker, B. 2019. Real-Life BPMN. 4th Edition.

GitHub. Features. Viitattu 9.9.2024. Saatavissa https://github.com/features

Goodwin, M. & Khan, T. 2023. What is YAML. Viitattu 10.9.2024. Saatavissa https://www.ibm.com/topics/yaml

Herzberg, G., Panikkar, R., Sahu, A. & Whiteman R. 2020. The imperatives for automation success. McKinsey & Company. Viitattu 15.9.2024. Saatavissa https://www.mckinsey.com/capabilities/operations/our-insights/the-imperatives-for-automation-success

Holmes-Higgin, P. 2016. Flowable and Activiti: What the Fork. Flowable-blogi. Viitattu 10.9.2024. Saatavissa https://www.flowable.com/blog/engineering/flowable-and-activiti-what-the-fork

Holmes-Higgin, P. 2020. Release frequency & future upgrade transition. Foorumiviesti. Viitattu 10.9.2024. Saatavissa https://forum.flowable.org/t/release-frequency-future-upgrade-transition/5313/2

IBM. What is business process management (BPM). Viitattu 9.9.2024. Saatavissa https://www.ibm.com/topics/business-process-management

IBM. 2021. IBM Business Process Manager overview. Viitattu 9.9.2024. Saatavissa https://www.ibm.com/docs/en/bpm/8.6.0?topic=manager-business-process-overview

Introducing JSON. Viitattu 9.9.2024. Saatavissa https://www.json.org/json-en.html

Johannesson, P. & Perjons, E. 2021. An Introduction to Design Science. E-kirja. Cham: Springer Nature Switzerland AG. Saatavissa rajoitetusti https://primo.aalto.fi

Meyer, D. 2016. Camunda Engine Evolution since Activiti Fork. Camunda-blogi. Viitattu 9.9.2024. Saatavissa https://camunda.com/blog/2016/10/camunda-engine-since-activiti-fork/

Microsoft a. Clearly visualize business processes. Viitattu 9.9.2024. Saatavissa https://www.microsoft.com/en-us/microsoft-365/visio/business-process-modeling-notation

Microsoft b. Open source on Azure. Viitattu 9.9.2024. Saatavissa https://azure.microsoft.com/en-us/solutions/open-source

Microsoft c. What is Azure. Viitattu 9.9.2024. Saatavissa https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/

Microsoft. 2023. What is the Windows Subsystem for Linux. Viitattu 10.9.2024. Saatavissa https://learn.microsoft.com/en-us/windows/wsl/about

Microsoft. 2024a. Azure Key Vault basic concepts. Viitattu 9.9.2024. Saatavissa https://learn.microsoft.com/en-us/azure/key-vault/general/basic-concepts

Microsoft. 2024b. Monitor performance by using the Query Store. Viitattu 9.9.2024. Saatavissa https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store

Microsoft. 2024c. Storage account overview. Viitattu 9.9.2024. Saatavissa https://learn.microsoft.com/en-us/azure/storage/common/storage-account-overview

Microsoft. 2024d. What is Azure Container Instances. Viitattu 9.9.2024. Saatavissa https://learn.microsoft.com/en-us/azure/container-instances/container-instances-overview

Microsoft. 2024e. What is Azure DevOps. Viitattu 9.9.2024. Saatavissa https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops

Microsoft. 2024f. What is Azure SQL Database. Viitattu 9.9.2024. Saatavissa https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview

Minimal "history plugins" for Camunda Cockpit. Käyttöohje. Viitattu 10.9.2024. Saatavissa https://github.com/datakurre/camunda-cockpit-plugins/blob/master/README.md

Mozilla. 2024a. JSON. Viitattu 9.9.2024. Saatavissa https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

Mozilla. 2024b. XML introduction. Viitattu 10.9.2024. Saatavissa https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction

Mucci, T. & Stryker, C. 2024. What is business process automation. IBM. Viitattu 9.9.2024 Saatavissa https://www.ibm.com/topics/business-process-automation

Object Management Group. 2013. Business Process Model and Notation (BPMN), Version 2.0.2. Viitattu 10.9.2024. Saatavissa https://www.omg.org/spec/BPMN/2.0.2/PDF

Object Management Group. BPMN, CMMN and DMN Specifications at OMG. Esite. Viitattu 10.9.2024. Saatavissa https://www.omg.org/intro/TripleCrown.pdf

Oracle. 2024. Keytool. Viitattu 9.9.2024. Saatavissa https://docs.oracle.com/en/java/javase/11/tools/keytool.html

Pappas, J. 2020. Migrating processes from other vendors to Camunda. Camunda-blogi. Viitattu 15.9.2024. Saatavissa https://camunda.com/blog/2020/03/migrating-processes-from-other-vendors-to-camunda/

Piirainen, J. 2023. Prosessiautomaation haasteet ja mahdollisuudet vuonna 2023. Twoday-blogi. Viitattu 9.9.2024. Saatavissa https://www.twoday.fi/blogi/blogi/prosessiautomaation-haasteet-ja-mahdollisuudet

Red Hat. 2021. What is Kogito. Viitattu 9.9.2024. Saatavissa https://www.red-hat.com/en/topics/automation/what-is-kogito

Red Hat. 2022. What is business process automation. Viitattu 9.9.2024. Saatavissa https://www.redhat.com/en/topics/automation/what-is-business-process-automation

Red Hat. Chapter 1. About Red Hat Process Automation Manager. Viitattu 9.9.2024. Saatavissa https://docs.redhat.com/en/documentation/red_hat_process_automation_manager/7.3/html/planning_a_red_hat_process_automation_manager_installation/about_red_hat_process_automation_manager

SAP Signavio. SAP Signavio Process Manager. Viitattu 10.9.2024. Saatavissa https://www.signavio.com/products/process-manager/

Thengvall, M. 2024. Important Update: Camunda 7 Community Edition End of Life Announced. Foorumiviesti. Viitattu 10.9.2024. Saatavissa https://forum.camunda.io/t/important-update-camunda-7-community-edition-end-of-life-announced/50921

Trisotech a. Digital Automation Suite. Viitattu 10.9.2024. Saatavissa https://www.trisotech.com/digital-automation-suite/

Trisotech b. Digital Modeling Suite. Viitattu 10.9.2024. Saatavissa https://www.trisotech.com/digital-modeling-suite/

Visual Paradigm a. BPMN Notation Overview. Viitattu 9.9.2024. Saatavissa https://www.visual-paradigm.com/guide/bpmn/bpmn-notation-overview/

Visual Paradigm b. Process Design Tool. Viitattu 1.10.2024. Saatavissa https://www.visual-paradigm.com/features/process-design-tool/

Visual Paradigm c. What is Case Management Model and Notation (CMMN). Viitattu 10.9.2024. Saatavissa https://www.visual-paradigm.com/guide/cmmn/what-is-cmmn/

Zickner, V. 2023. Low-code, High Impact: Flowable & CMMN for Complex Use Cases. Flowable-blogi. Viitattu 10.9.2024. Saatavissa https://www.flowable.com/blog/engineering/low-code-high-impact