Mortti Aittokoski

Tekoäly apuna diabeteksen itsehoidossa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

7.3.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Mortti Aittokoski Tekoäly apuna diabeteksen itsehoidossa 30 sivua 7.3.2018
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	Mobile Solutions
Ohjaaja	Lehtori Peter Hjort

Insinöörityössä tutkittiin tekoälyn hyötyjä terveydenhuollossa ja tavoitteena oli kehittää iOS-sovellus, joka hyödyntää tekoälyä diabeteksen itsehoidossa.

Insinöörityössä selvitettiin, kuinka tekoälyä on hyödynnetty aikaisemmin terveydenhuollossa, ja etsittiin sopivaa ratkaisua iOS-sovellukseen integroitavaksi. Työssä päädyttiin vertailemaan sääntömoottoreita, ja sopivin valittiin integroitavaksi. Sääntömoottorin avulla lääkärin tietämystä voisi siirtää iOS-sovellukseen ja sen avulla käsitellä potilaan tietoja. Tämä mahdollistaa diabeetikon terveyden ja elämäntapojen seurannan, jota kautta autetaan potilasta pysymään tavoitteissa, jotka auttavat vähentämään diabeteksen oireita ja havaitsemaan terveyteen vaikuttavia tapahtumia.

Sääntömoottoriksi valittiin CLIPS, joka integroitiin osaksi sovellusta. Sääntömoottoria käytettiin osana älykkäitä avustajia, jotka sisältävät tarvittavat määritykset tavoitteiden asettamiseen, tiedon käsittelyyn ja sen esittämiseen.

Työtä voidaan hyödyntää tekoälyä käyttävän sovelluksen kehittämiseen ja oppimiseen tekoälyn mahdollisuuksista terveydenhuollossa. Työn lopputuloksena luotiin onnistuneesti sovellus, joka auttaa diabeetikkoa elämään terveellisempää elämää.

Avainsanat	diabetes, iOS, tekoäly, sääntömoottori

Metropolia

Author Title Number of Pages Date	Mortti Aittokoski Augmented Intelligence with Diabetes Self-care 30 pages 7 March 2018
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Mobile Solutions
Instructor	Peter Hjort, Senior Lecturer

Goal of this thesis was to research the benefits of augmented intelligence in healthcare and develop an iOS-application, that uses augmented intelligence in diabetes self-care.

This thesis studies how augmented intelligence has been used previously in healthcare, and how it could be integrated to an iOS-application. Rules engines were compared to create this functionality, and the best suited Rules engine was selected to be integrated in the application. With the rules engine doctor's knowledge could be transferred to the iOS-application and used to analyze patients data. This allows monitoring the lifestyle and health of the patient, and helping the patient to reach the important goals to minimize the negative symptoms of diabetes and to detect events that impact patients health.

CLIPS was selected as the rules engine and was integrated as part of the application. Rules engine was used as part of the intelligent assistants which contains the necessary settings for setting the goal and analyzing and presenting the data.

This thesis can be used for developing an application that uses augmented intelligence, and for learning the possibilities of augmented intelligence in healthcare. As a result an application was successfully developed that helps diabetics to live a healthier life.

Keywords	diabetes, iOS, augmented intelligence, rules engine

Metropolia

Sisällys

1 Johdanto

Opinnäytetyön tarkoituksena on tutkia tekoälyn hyötyjä ja mahdollisuuksia diabeteksen itsehoidossa. Työssä on tavoitteena kehittää sovellus, jonka keskeinen toiminnallisuus on diabeetikon arkea helpottavat tekoälyyn perustuvat toiminnot. Sovellus tehdään jatkokehityksenä yrityksessä. Yksi diabeetikon arkea helpottava toiminto on kaavojen tunnistaminen, esimerkiksi vähäisestä liikunnasta seuraava hypoglykemia. Projektin tavoitteena on luoda sovellukseen tekoälyyn pohjautuvia ominaisuuksia, jotka antavat käyttäjälle, diabeetikolle, hyödyllistä ja ajankohtaista tietoa ja ohjeita, joka auttavat terveellisempään elämää diabeteksen kanssa.

Tekoälyllä tässä yhteydessä tarkoitetaan terveydenhuollon ammattilaisten luomia apureita, jotka sisältävät tiedon kuvioista, joita käyttäjän tiedosta etsitään. Kuvion havaitseminen johtaa toimintaan, jolla pyritään edistämään diabeetikon terveellisempiä elämäntapoja. Englanninkielinen vastine tekoälylle, johon tässä insinöörityössä viitataan, on ”augmented intelligence” eikä ”artificial intelligence”.

Älykäs avustaja rakennetaan käyttämällä CLIPS-työkalua, joka on alun perin NASA:n 1980–1990-luvuilla kehittämä asiantuntijajärjestelmätyökalu. Tämä työkalu sisältää ohjelmointikielen, jonka avulla algoritmit on luotu assistenteille. Tämä tarjoaa alustan tekoälylle sovelluksessa.

Sovellus, johon älykäs avustaja rakennetaan, on Android- ja iOS-alustoille kehitetty mobiilisovellus, jonka nykyinen versio tarjoaa päiväkirjan mm. insuliinin, hiilihydraattien ja verensokerin kirjaamiseen sekä alustan etähoidolle, jossa mobiiliohjelman keräämät tiedot välittyvät hoitohenkilökunnalle. Lisäksi sovelluksen tuoteperheeseen kuuluu terveydenhoidon ammattihenkilöille tarkoitettu web-käyttöliittymä, jolla hoitohenkilökunta voi seurata potilaan tilaa sekä antaa viestintäkanavan kautta hoito-ohjeita ja palautetta potilaalle niitten tietojen pohjalta, joita käyttäjä on sovellukseen syöttänyt.

Diabetes on yleistymässä maailmanlaajuisesti. Taudin hoidosta aiheutuvat kustannukset ovat sen myötä kasvussa. Suomessa diabeteksen sairaanhoidon kustannukset ovat noin 8,8 % terveydenhuollon menoista [1]. Nykyiset hoitomallit eivät ole kustannustehokkaita. Tekoälyä voisi hyödyntää diabeteksen itsehoidossa seuraamalla potilaan elintapoja ja

elintoimintoja ja näitten perusteella luoda automaattisesti käyttäjälle suosituksia ja havaintoja, jotka auttavat potilasta elämään terveellisempää elämää, mikä vähentää terveydenhuollon kustannuksia.

2 Tekoäly terveydenhuollossa

2.1 Ongelma ja ratkaisu

Terveydenhoidon iso ongelma nykypäivänä on tiedon suuri määrä ja ajanpuute. Tutkijat, terveydenhoidon ammattilaiset ja lääketieteelliset laitteet tuottavat valtavia määriä tietoa. Tutkijat ja lääkeyritykset löytävät jatkuvasti uusia tapoja hoitaa sairauksia [2]. Silti lääkäreiden käytettävissä oleva aika tiedon sulattamiseen on edelleen yhtä rajattu. Tähän ongelmaan tekoälystä etsitään ratkaisua. On myös arveltu, että tekoälyn on tuotava tähän ongelmaan ratkaisu, jotta sillä voisi olla merkittävää vaikutusta terveysalaan [3]. Kuvassa 1 on kuvattu prosessi, jossa lääkäri rakentaa potilaalle diagnoosin ja hoitosuunnitelman.

Kuva 1. Prosessi, jossa lääkäri luo diagnoosin ja hoitosuunnitelman potilaalle [2].

Tekoälyn avulla pyritään myös vähentämään sairauksien hoidosta aiheutuvia kustannuksia. Esimerkiksi diabeteksen sairaanhoidon kustannukset Suomessa vuonna 2011 olivat noin 1,5 miljardia euroa, joka vastaa noin 8,8 %:a terveydenhuoltomenoista [1]. Sen sijaan, että potilaan tarvitsisi käydä lääkärin vastaanotolla saadakseen diagnoosin ja hoitosuunnitelman, potilas ja tekoäly voisivat tehdä sen. Lääkärin rooli siirtyisi enemmän tarkkailijan ja valmentajan kaltaiseksi, ja varsinainen hoito tapahtuisi tekoälyn avulla. Potilaan ja tekoälyn rajapinta voisi olla mobiilisovellus, kuten tämän työn aiheena oleva sovellus.

2.2 Tekoälyn määritelmä terveydenhoidossa

Suomen kielen käsitettä tekoäly vastaa englannin kielessä parhaiten käsite artificial intelligence. Toinen aiheeseen liittyvä käsite on augmented intelligence. Lyhenne on englanniksi molemmissa käsitteissä sama, ”AI”, minkä vuoksi lyhenteen merkitys sekoittuu helposti. IBM Watsonia koskevat kirjoitukset käyttävät usein artificial intelligence -termiä kuvaamaan Watsonia. IBM itse käyttää Watsonista termiä augmented intelligence [4]. Myös tämän työn sovellus käyttää augmented intelligencen kaltaista logiikkaa, mutta koska sitä vastaavaa termiä ei ole suomen kielessä, käytetään tässä insinöörityössä tekoäly-termiä kuvaamaan molempia käsitteitä.

Artificial Intelligence pyrkii korvaamaan tai imitoimaan ihmisen toimintaa. Googlen itseajava auto, jossa ihmisellä ei ole mitään kontrollia autoon, on yksi esimerkki tällaisesta toteutuksesta. Augmented Intelligence pyrkii avustamaan ihmisen ajattelua, analysoimista ja suunnittelua, mutta jättää ihmisen edelleen tekemään päätökset. Esimerkiksi IBM Watson Oncology avustaa lääkäriä potilaan tilan analysoimisessa, tiedon keräämisessä ja hoitosuunnitelman laatimisessa. Lääkärin roolia ei korvata, ja lääkäri edelleen tekee päätökset potilaan hoidosta. [5.]

2.3 IBM Watson

Ensimmäinen enemmän huomiota saanut kokeilu tekoälyn hyödyntämisessä terveydenhoidossa on IBM:n Watson. Sen mahdollisuuksia alettiin tutkia jo 1990-luvulla. Nyt se pystyy tarjoamaan yhtenäisen alustan lääkäreiden ja terveysalan ammattilaisten tueksi. ”IBM Watson for Oncology” on erityisesti onkologeille suunniteltu työkalu syövän hoitoon.

[6.] Se on kehitetty yhteistyössä amerikkalaisen MSK Cancer Centerin kanssa ja on kykenevä analysoimaan sille syötetyn tiedon tarkoituksen ja kontekstin lukuisista eri lähteistä, kuten käsin kirjoitetuista dokumenteista. Watson juontaa tietoa useista eri lähteistä, kuten 290 terveysalan julkaisusta, yli 200 oppikirjasta ja 12 miljoonasta sivullisesta tekstiä. IBM tekee yhteistyötä intialaisen Manipal Hospitalsin kanssa. Watsonia käytetään yli 16 laitoksessa, joissa yli 200 000:ta potilasta hoidetaan vuosittain. Tämä yhteistyö auttaa myös helpottamaan kovaa pulaa onkologeista sairaaloissa. American Cancer Society käyttää IBM Watsonia tiedon louhimiseen internetsivuilta ja asiaan liittyvistä lähteistä tarjotakseen henkilökohtaista hoitoa onkologiapotilaille. Artikkelin mukaan IBM sanoo sairaalan käyttävän Watsonia yli 16 onkologisessa toimessa, joissa DNA:n tietoa hyödynnetään henkilökohtaisen hoidon tarjonnassa potilaille [6].

IBM Watson analysoi potilaan tiedot, pyytää tarvittavia lisätietoja ja esittää yleiskatsauksen potilaan syövän tilasta perustuen analysoituun ja syötettyyn tietoon. Watson esittää yleiskatsausnäkymässä yhteenvedon oikealla ja vasemmalla tiedon alkuperäisen lähteen, jotta lääkäri voi varmistaa, että algoritmi teki oikeat johtopäätelmät sen käytettävissä olevista tiedoista. Lääkärin pitää vahvistaa algoritmin oikeellisuus ja joskus syöttää lisätietoja, jos tiedot eivät löydy potilastiedoista. Watson tarjoaa yhden tai useampia hoitosuunnitelmia jotka voisivat olla soveltuva kyseiselle potilaalle. Se esittää hoitosuunnitelmia luokiteltuna suositeltaviin, harkittaviin ja ei-suositeltaviin. Watsonilla on myös lukuisia muita ominaisuuksia, kuten kyky näyttää tutkimuskirjallisuutta sekä tarjota tietoa lääkkeistä ja niitten sivuvaikutuksista. Lisäksi Watson pystyy tarjoamaan tietoa kliinisistä tutkimuksista. Lääkäri pystyy aina tarkistamaan, mistä Watsonin tarjoama tieto on peräisin, siihen liitetystä linkistä, joka tarjoaa pääsyn aineistoon, jota tiedon luomisessa käytettiin. [6.]

Nykyisin IBM Watson Oncology kykenee analysoimaan tekstiä, mutta tulevaisuudessa se mahdollisesti kykenee analysoimaan myös kuvia ja genomeita IMB Medical Sleeve- ja IBM Genomics -hankkeiden myötä.

2.4 Google DeepMind

Lontoolaiset sairaalat Barnet, Chase Farm ja Royal Free Hospitals tekevät yhteistyötä Googlen tekoälyyn erikoistuneen haaran DeepMindin kanssa jakamalla potilastietoja [6]. Tämän yhteistyön on arveltu säästävän vuosittain paperityötä yli puoli miljoonaa tuntia,

joka voitaisiin käyttää potilaiden hoitamiseen. Kaikki tieto välitetään salattuna DeepMindille, eikä sitä jaeta Googlelle. Analysoitua tietoa hyödynnetään Streams-mobiilisovelluksen kautta. Se toimittaa hälytyksen klinikoille, kun potilailla havaitaan akuutteja alkuvaiheen vahinkoja munuaisissa. Sovellus on ollut tarkoitus ottaa käyttöön vuoden 2017 alussa. Jos se osoittautuu hyödylliseksi, sitä laajennetaan muitten henkeä uhkaavien sairauksien tunnistamiseen, kuten verenmyrkytykseen ja elinten vikoihin. Järjestelmä säästää aikaa koostamalla potilaan historian ja ilmoittamalla huomionarvioisista havainnoista. Kokonaiskuvan luonti veisi lääkäreiltä paljon aikaa ilman järjestelmää. [6.]

2.5 IBM Watson Medical Sleeve ja Watson Avicenna

IBM Watson Medical Sleeve -projekti keskittyy radiologian ja kardiologian alaan. Watson analysoi kuvia ja auttaa lääkäreitä havaitsemaan poikkeamia. Projektia ollaan tällä hetkellä kaupallistamassa Watson Avicenna-nimellä.

3 Asiantuntijajärjestelmät

Asiantuntijajärjestelmä on tietokoneohjelma, joka on kykenee samankaltaiseen ongelmanratkaisuun kuin alan asiantuntija [7]. Asiantuntijajärjestelmät on suunniteltu ratkaisemaan monimutkaisia ongelmia käsittelemällä niille syötettyä tietoa sisäänrakennettujen algoritmien avulla. Nämä algoritmit sisältävät sarjan kysymyksiä, ja järjestelmä kykenee hakemaan vastauksen sille tarjotusta tiedosta. Yksinkertaisimmillaan asiantuntijajärjestelmän kysymykset muodostavat monimutkaisen puurakenteen, ja mitä laajempi puurakenne on, sitä tarkemman ja yksityiskohtaisemman vastauksen järjestelmä kykenee antamaan.

Asiantuntijajärjestelmät käyttävät ihmisten tarjoamaa tietoa ongelman ratkaisuun, joka tyypillisesti vaatisi ihmisen älykkyyttä. Asiantuntijajärjestelmissä tämä tieto voidaan esittää sääntöinä, ja niitä voidaan analysoida ratkaistaessa ongelmaa. Esimerkiksi kirjat sisältävät paljon tietoa, mutta ihmisen tulee lukea ja käsitellä tietoa hyödyntääkseen sitä. Perinteiset tietokoneohjelmat suorittavat tehtäviä käyttäen tyypillistä ongelmanratkaisulogiikkaa, joka sisältää vain algoritmin suorittamiseen tarvittavan suppean määrän tietoa. Tämä ongelmanratkaisulogiikka on yleensä rakennettu osana sovelluskoodia. Jos logiikkaa haluaa muuttaa, tulee sovelluskoodia muuttaa. Muutosten käyttöönotto vaatisi sovelluspäivityksen luomista ja asentamista. Asiantuntijajärjestelmä voi sisältää enemmän

ihmisen tarjoamaa tietoa kuin perinteinen tietokoneohjelma ja hyödyntää sitä useamman eri ongelman ratkaisemisessa.

Asiantuntijajärjestelmiä kehitetään yleensä asiantuntijajärjestelmille tarkoitetuilla sovelluskehitystyökaluilla. Niissä on käyttöliittymä jonka avulla tietoa syötetään työkalun vaatimassa formaatissa. Yleensä mukana on useita toiminallisuuksia listojen, tekstien ja objektien käsittelyyn, ja rajapinnat ulkoisen sovelluksen kanssa kommunikoimiseen ja sen tietokannan käsittelyyn. Näitä sovelluskehitystyökaluja voidaan pitää ohjelmointikielinä joilla on perinteistä ohjelmointikieltä paljon suppeampi soveltuvuus eri käyttötarkoituksiin.

Asiantuntijajärjestelmä koostuu tyypillisesti faktoista, säännöistä ja sääntömoottorista. Sääntömoottori on keskeinen osa asiantuntijajärjestelmän rakentamisessa [8].

Asiantuntijajärjestelmiä käytetään lukuisilla eri liiketoiminnan aloilla kuten rahoituksessa, terveydenhuollossa, jälleenmyynnissä ja teollisuudessa [9, s. 6].

3.1 Sääntömoottorin hyödyntäminen diabeteksen itsehoidossa

Sääntömoottorin avulla voisi tuoda terveydenhuollon ammattilaisen osaamista mobiilisovellukseen. Sääntömoottori tarjoaa alustan, joka skaalautuu isoon määrään sääntöjä, joilla potilaan tilaa voidaan analysoida. Sen avulla voidaan rakentaa sovellus, joka käyttää useita sääntöjä tiedon käsittelyyn, ilman että sovellukseen tarvitsee rakentaa algoritmeja sääntöjen ajoon. Sääntömoottorin käyttö mobiilisovelluksessa mahdollistaa sääntöjen päivittämisen ajon aikana, eikä tämä vaadi käyttäjältä toimia. Sääntöjen ajo mobiilisovelluksessa mahdollistaa välittömän reagoimisen tilaan. Sääntöjen ajo palvelimella mahdollistaisi resurssien skaalaamisen tarpeen mukaan, mutta loisi viiveen tiedon välittämiseen käyttäjälle. Ajettaessa sääntöjä mobiilisovelluksella resursseja ei pysty skaalaamaan, minkä vuoksi on varmistettava, että sääntöjen vaatimat resurssit pysyvät hyväksyttävissä rajoissa. Sääntömoottorin käyttö mobiilisovelluksessa voi tämän vuoksi olla riski tulevaisuuden tarpeita ajatellen. [10.]

3.2 Sääntömoottori

Sääntömoottori on sääntöihin pohjautuva työkalu asiantuntijajärjestelmän toteuttamiseen. Sen tehtävänä on käsitellä suurta määrää sääntöjä ja faktoja. Säännöt ovat ammattilaisten luomia, ja niitten tarkoitus on tuoda alan ammattilaisen tietämys järjestelmään. Faktat ovat asia, jota sääntömoottori käsittelee sääntöjen avulla. Sääntöjen ja faktojen tuloksena syntyy tulos, jota voidaan esittää faktana. Tulos annetaan pääohjelmalle, jonka sisällä sääntömoottoria ajetaan. Tulos kertoo pääohjelmalle, mitä toimintoja sen tulisi suorittaa [8]. Sääntömoottorin tehtävä on ladata faktat ja säännöt muistiin, suorittaa ne käskettäessä ja syöttää ulos tehtävät toiminnot [11].

Prosessia, jossa faktoja verrataan sääntöjä vasten, kutsutaan kuvion sovittamiseksi, jonka rajapinnan moottori suorittaa. Näissä on käytössä useita eri algoritmeja, joilla kuvion sovittamista tehdään: lineaarinen algoritmi käsittelee listan elementtejä järjestyksessä, kunnes osuma löydetään tai kaikki elementit on käsitelty [12].

Rete-algoritmi on hyvin yleinen sääntömoottoreissa. Siitä on useita laajennettuja implementaatioita, kuten ReteOO, RetePlus ja Rete III [8, 1.1.1]. Rete-algoritmissa jokaisen ehdon arvioinnin tulos pidetään muistissa faktoina, mikä poistaa jatkuvan laskemisen tarpeen. Tämän ansiosta sääntöjä, jotka ovat luonnostaan kuvioiden sovittamista, voidaan käsitellä paljon nopeammin. Faktojen kopiointi on myös täten nopeampaa kuin lineaarisella algoritmilla [8, 1.2.1]. Rete on perinteisesti suoritettu yksisäikeisesti, mutta joissain sääntömoottoreissa sille on tehty tuki monisäikeisyydelle [8, 2.6.1.2] Rete-algoritmin hyötyjä on suuri muistissa pidettävien faktojen määrä, mikä minimoi määrän, jolloin kahta elementtiä tarvitsee verrata keskenään [13, s. 43]. Rete-algoritmin suurin haittapuoli on, että kun elementti poistetaan muistista sen tila puretaan. Tämä usein vaatii samojen toimintojen suorittamisen uudestaan samassa järjestyksessä, kuin elementin tilan luomisessa suoritettiin. Toinen Rete-algoritmin haittapuoli on mahdollinen kombinatorisesti kasvava beta-muistin koko, joka johtuu verkkorakenteiden jakamisesta [13, s. 43]. Kolmas haittapuoli on, että beta-muistin ylläpitämiseksi join-operaatiot on suoritettava tietyssä järjestyksessä. Tämä järjestys määräytyy staattisesti kääntämisen yhteydessä. Neljäs haittapuoli on, että käsitettä aseman (condition) säilyttämistä ei tueta [14, s. 6]. Ylläpitäkseen verkon tilaa algoritmin tulee usein suorittaa raskaita laskutoimituksia säännöille, jotka eivät ole aktiivisia.

Treat-algoritmi perustuu samaan perjaatteeseen kuin Rete-algoritmi. Se on toteutukseltaan monimutkaisempi, mutta tarjoaa yleensä paremman suoritukyvyn [13, s. 46].

Sääntömoottorin käyttö soveltuu sellaisten algoritmien ratkaisemiseen, joihin kuuluu paljon ehtojen käsittelyä ja päätöstentekoa. Päätösten tulisi olla riittävän yksinkertaisia, siten että niistä kykenee kirjoittamaan säännön. Sääntöjä tulisi olla vähintään kaksikymmentä, muuten sääntömoottoriin käytetty panostus ei maksa itseään takaisin [8, 1.2.3.] Ehtoja yhdessä säännössä olisi hyvä olla enemmän kuin kolme, koska pienemmän määrän käsittelyyn sääntömoottori ei välttämättä ole oikea työkalu. Sääntöjen tulisi olla luonteeltaan sellaisia, että ne mahdollisesti muuttuvat ajan myötä. Mikäli ne ovat hyvin staattisia, sääntömoottorista ei välttämättä ole hyötyä. Sääntömoottori soveltuu hyvin, mikäli tuotetta kehitetään jatkuvasti, sillä varsinaisen sovelluksen logiikkaa ei tarvitse muuttaa sääntömoottorin toimintaa muuttaakseen [7]. Toisaalta, jos tuotetta ei ole tarkoitus kehittää jatkossa, sääntömoottori ei välttämättä tarjoa mitään selvää etua. Mikäli suorituskyky ja muistinkäyttö ovat kriittisiä resursseja ajoympäristössä, ei sääntömoottorin käyttö välttämättä kannata. Sääntömoottoria tulisi käyttää ympäristössä, jonka suorituskykyvaatimukset ovat skaalautuvia. Liiketoiminnan kannalta sääntömoottoria käyttävä implementaatio kääntyy voitolliseksi aikaisintaan vuoden päästä tuotteen toimittamisesta asiakkaalle. Tähän vaikuttavat tyypillisesti isot kustannuksen kehittäjien ja asiakkaiden koulutuksessa. Toisaalta pidemmällä aikavälillä sääntömoottorin käyttö kääntyy kannattavaksi joustavuuden ja helpon ylläpidettävyyden kautta. [10.]

Sääntömoottoreissa käytetään kahta eri metodia sääntöjen suorittamiseen: eteenpäin ja taaksepäin ketjuttamista. Sääntömoottori voi tukea joko eteenpäin tai taaksepäin ketjuttamista tai molempia. [8, 1.2.] Eteenpäin ketjutettaessa aloitetaan tiedosta, joita esitetään faktoina, jotka johtavat yhden tai useamman johtopäätöksen täyttymiseen. Taaksepäin ketjuttamisessa käsittely alkaa johtopäätöksestä, jonka sääntömoottori yrittää täyttää. Mikäli johtopäätöstä ei saada täytettyä, sääntömoottori etsii toisen osittaisen johtopäätöksen jonka se voi täyttää. Tätä prosessia jatketaan, kunnes alkuperäinen johtopäätös saadaan täytettyä tai osittaisia johtopäätöksiä ei enää löydy.

3.3 Sääntömoottorin valintakriteerit

Tavoitteena on löytää sopivin sääntömoottori mobiilisovelluksessa suoritettavaksi. Sääntömoottorin tulee olla ajettavissa iOS- ja Android-alustoilla. Sääntömoottorin implementoinnin helpottamiseksi siitä tulisi olla saatavilla valmiiksi käännetty versio kummallekin alustalle. Eteenpäin ketjuttamisen tulee olla tuettuna, jotta kerätyistä tiedoista voidaan tehdä johtopäätöksiä. Taaksepäin ketjuttaminen on eduksi tulevaisuudenkestävyyden

vuoksi, mutta se ei ole välttämätöntä. Suorituskyky on keskeinen tekijä sopivan sääntömoottorin valinnassa, sillä suorituskyvyn skaalautuminen perustuu pääasiassa laitekannan uusiutumiseen, joka tarjoaa parempaa prosessointisuorituskykyä ja suurempaa sovellusmuistia. Sääntömoottoreiden suorituskyky saattaa myös parantua kehityksen myötä esimerkiksi parantamalla sääntöjen käsittelyalgoritmia ja lisäämällä tuen monisäikeiselle ajolle. Sääntökielen tulee mahdollistaa sääntömallien hyödyntäminen, jotta uusia sääntöjä voidaan rakentaa tehokkaasti vanhojen pohjalta. Sääntömoottorin lisenssin tulisi mahdollistaa rajoittamaton ja ilmainen käyttö kaupallisessa suljetun lähdekoodin ohjelmassa. Sen tulisi tarjota Androidalustalla Java-kielinen rajapinta ja iOS-alustalla Swift-kielinen rajapinta. Tämä tekisi integraatiosta mahdollisimman helppoa siten, että integraatio voitaisiin tehdä käyttäen samaa ohjelmointikieltä, kuin pääsovelluksessa on käytetty. Uutta sovelluskehystä, tässä tapauksessa sääntömoottoria, integroitaessa on vaikea ennustaa, kuinka paljon eteen tulee haasteita. Tämän vuoksi hyvä dokumentaatio ja aktiivinen laaja yhteisö ovat tärkeitä tekijöitä sääntömoottoria valittaessa.

Edellä mainitut kriteerit rajaavat sääntömoottorin valintaa huomattavasti. Tähän insinöörityöhön on valittu ne sääntömoottorit, jotka täyttävät kriteerit mahdollisimman hyvin. Sääntömoottorien valinnassa on käytetty lähteinä Chinmoy Mukherjeen Artificial Intelligence: Rules Engines for Mobile Platforms -teosta, joka vertailee Android-alustalla toimivia sääntömoottoreita. Lisäksi Stack overflow -yhteisön ja Google-hakukoneen osumamäärää on käytetty sen arvioimiseen, kuinka paljon käytettyjä eri sääntömoottorit ovat.

3.4 Drools-sääntömoottori

Drools on Javalla kirjoitettu liiketoimintaan suunnattu sääntöjenhallintajärjestelmä (BRMS eli business rule management system). Se käyttää Apache Software Licence 2.0 -lisenssiä, joka mahdollistaa sen kaupallisen käytön ilman lisenssimaksuja. Sen uusin versio 7.5.0 Final on julkaistu joulukuussa 2017. Google Trends -hakutilastoissa Drools on suosituin tässä opinnäytetyössä käsitellyistä sääntömoottoreista. Stack Overflow -yhteisöstä löytyy 580 osumaa hakusanalla ”drools rule engine”. Drools käyttää Rete-algoritmin laajennettua versiota nimeltä ReteOO, joka on suunnattu oliokielten kanssa käytettäväksi. Leaps-algoritmi oli aikaisemmin tuettuna, mutta ylläpidon lakattua sen tuki lopetettiin. Drools tukee eteenpäin ja taaksepäin ketjuttamista. Se tukee monisäikeistä ajoa, mutta se on kokeellisella tasolla. [8, 2.6.1.2.] Sääntöjen kirjoittamiseen Drools käyttää omaa kieltä, jonka tiedostopääte on .drl.

Drools on saatavilla Android- ja iOS-alustoille. Android-alustalle tuen liittäminen onnistuu lisäämällä se Gradle-riippuvuuksiin. IOS tuki on toteutettu RoboVM-kääntäjällä, jonka tuen Microsoft lakkautti vuonna 2016. Drools-sääntömoottorille on saatavilla hyvin kattava dokumentaatio. Jboss.org tarjoaa 354-sivuisen käyttöoppaan Droolsin käyttöön. Käyttöoppaassa käsitellään esimerkiksi sitä, mikä sääntömoottori on, mitkä sen hyödyt ovat, milloin sääntömoottoria tulisi käyttää, milloin sitä ei tulisi käyttää, teoriaa sääntömoottorin eri komponenteista, teoriaa muun muassa Rete-algoritmista, sääntöjen rakentamista, sääntöjen käyttöönottoa, sääntökieltä, sääntöpohjia, sääntömoottorin Java-rajapintaa, Droolsin käyttöä Eclipsellä sekä esimerkkiprojekteja, joissa on käytetty Droolsia. Esimerkkisäännössä 1 on kolme Droolsin dokumentaatiossa esitettyä sääntöä. Ne kuuluvat kaikki chekout agenda -ryhmään. Ensimmäinen sääntö laskee gross total -summan, mikäli sitä ei ole vielä laskettu, ja lisää sen nykyiseen sessioon ja asettaa sen textArea-muuttujaan. Toinen sääntö vähentää gross total -summasta 5 %, mikäli gross total on 10–20. DiscountedTotal-arvo asetetaan textArea-muuttujaan. Kolmas sääntö vähentää gross total -summasta 10 %, mikäli gross total on vähintään 20. DiscountedTotal-arvo asetetaan textArea-muuttujaan.

```
rule "Gross Total"
    agenda-group "checkout"
    dialect "mvel"
when
   $order : Order( grossTotal == -1)
   Number( total : doubleValue )
       from accumulate( Purchase( $price : product.price ), sum( $price ) )
then
    modify( $order ) { grossTotal = total };
    textArea.append( "\ngross total=" + total + "\n" );
end

rule "Apply 5% Discount"
    agenda-group "checkout"
dialect "mvel"
when
   $order : Order( grossTotal &gt;= 10 &amp;&amp; &lt; 20 )
then
   $order.discountedTotal = $order.grossTotal * 0.95;
   textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );
end


rule "Apply 10% Discount"
    agenda-group "checkout"
    dialect "mvel"
when
   $order : Order( grossTotal &gt;= 20 )
then
   $order.discountedTotal = $order.grossTotal * 0.90;
   textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );
end
```

Esimerkkisääntö 1. Esimerkki Drools-säännöstä.

3.5 JEOPS-sääntömoottori

JEOPS on Javalla kirjoitettu sääntömoottori, ja Android-alusta on tuettuna. Sen uusin versio 2.0 beta 1 on julkaistu vuonna 2000. JEOPS toimii esikääntäjänä, joka kääntää sääntötiedoston Java-tiedostoksi. Java-tiedosto implementoi sääntömoottorin rajapinnan. JEOPSin muistinkäyttö on hyvin pientä: sääntömoottori vie noin 31.5 kilotavua muistia [9, s. 18]. Se käyttää RETE-algoritmia ongelmanratkaisuun. JEOPS käyttää Rete-algoritmia sääntöjen käsittelyyn [9, s. 52]. JEOPS ei tue monisäikeisyyttä. Sen lisenssi on LGPL, joka mahdollistaa käytön suljetun lähdekoodin ohjelmassa ilmaiseksi ilman, että lähdekoodista tulisi tehdä avointa. JEOPSin käyttöopas on vain noin 10 sivua pitkä [15], eivätkä sen kotisivut enää toimi. Stack Overflow -yhteisöstä löytyy ”jeops rule

engine” -hakusanalla vain yksi osuma. JEOPSin dokumentaatiosta lainattu esimerkkisääntö 2 sanoo, että jos myyjä myy tuotetta, jonka asiakas haluaa, hintaan, jonka asiakas voi maksaa, tehdään kaupat [15].

```
rule trade {
  /* In this rule, one agent will buy a product from another agent. */
  declarations
    Salesman s;
    Customer c;
    Product p;
  conditions
    c.needs(p);  // he won't buy junk...
    s.owns(p);
    s.priceAskedFor(p) <= c.getMoney();
  actions
    s.sell(p);
    c.buy(p);
}
```

Esimerkkisääntö 2. Esimerkki JEOPS-säännöstä.

3.6 Jess-sääntömoottori

Jess on Javalla kirjoitettu sääntömoottori. Se käyttää Rete-algoritmia sääntöjen käsittelyyn [16]. Sen ensimmäinen versio julkaistiin vuonna 1995, ja uusin stabiili versio 7.1p2 julkaistiin 2008. Jess kehitettiin alun perin Java-klooniksi CLIPS-sääntömoottorista. Sen käyttö akateemisiin tarkoituksiin on ilmaista, mutta kaupalliseen käyttöön maksullista [16]. Versio 8.0 alpha tukee Android-alustaa, mutta iOS ei ole tuettuna. Sääntöjä voi kirjoittaa joko *Jess rule language* -kielellä tai XML-kielellä. *Jess rule leanguage* on suositeltu kieli [17]. Se pohjautuu Lisp-ohjelmointikieleen. Jess tukee monisäikeistä ajoa [18]. Jess 7.1p2 -versiolle on saatavilla 204-sivuinen käyttöopas, mutta 8.0-versiolle käyttöopasta ei ole saatavilla. Tukiyhteisö ei ole erityisen laaja: hakusanalla ”jess rule engine” Stack Overflow -yhteisöstä löytyy 73 osumaa. Jessin dokumentaatiosta lainatussa esimerkkisäännössä 3 asiakkaalle lähetetään ilmainen CD-RW-levy, jonka tuotenumero on 782321, mikäli asiakas ostaa CD-kirjoittajan ja on säännöllinen asiakas.

```
defrule free-cd-rw-disks
    (CatalogItem (partNumber ?partNumber) (description /CD Wri-
ter/))
    (CatalogItem (partNumber 782321) (price ?price))
    (OrderItem (partNumber ?partNumber))
    (Customer {orderCount > 1})
    =>
    (add (new Offer "Free CD-RW disks" ?price)))
```

Esimerkkisääntö 3. Esimerkki Jess-säännöstä.

3.7 CLIPS-sääntömoottori

CLIPS on tekijänoikeudeton työkalu asiantuntijajärjestelmien rakentamiseen. CLIPS on lyhenne sanoista *C Language Integrated Production System,* eli se on tuotantojärjestelmä johon on integroitu C-ohjelmointikieli. Sen ensimmäinen versio kehitettiin vuonna 1985 NASA-Johnson Space Centerissä. Projektin alkuperäinen nimi oli *NASA's AI Language (NAIL).* Alkuperäisen kehitysryhmän työskentely CLIPS:n parissa päättyi 1990-luvun puolivälissä. Gary Riley oli työskennellyt osana CLIPS:n alkuperäistä kehitysryhmää. Kun kehitys päättyi NASA:ssa, jatkoi Riley CLIPS:n tekijänoikeudettoman version kehitystä itsenäisesti. Hän on ollut mukana teoksen *Expert Systems: Principles and Programming* kirjoittamisessa. CLIPS-työkalun kehityksen lisäksi hän on ollut kehittämässä ja ylläpitämässä kolmea asiantuntijajärjestelmää kymmen vuoden aikajänteellä. [10.]

CLIPS käyttää Rete-algoritmia sääntöjen käsittelyyn [9, s. 9]. Se on hyvin laajasti käytetty, ja tarjoaa erittäin kattavan dokumentaation.

Esimerkkisäännössä 4 on esimerkki CLIPS-työkalulla kirjoitetusta säännöstä. Tässä säännössä todetaan auton käynnistysmoottorin olevan viallinen, mikäli auton avain on käynnistysasennossa eikä auto käynnisty.

```
(deftemplate car_problem
    (slot name)
    (slot status)
)
(deffacts trouble_shooting
    (car_problem (name ignition_key) (status on))
    (car_problem (name engine) (status wont_start))
    (car_problem (name headlights) (status work))
 )
(defrule rule1
    (car_problem (name ignition_key) (status on))
    (car_problem (name engine) (status wont_start))
     =>
    (assert (car_problem (name starter) (status faulty)))
 )
```

Esimerkkisääntö 4. Esimerkki CLIPS-työkalulla kirjoitetusta säännöstä [11].

CLIPS-työkalua kutsutaan asiantuntijajärjestelmätyökaluksi, koska se tarjoaa täydellisen ympäristön asiantuntijajärjestelmien kehittämiseen ja sisältää virheenetsintä-työkalun sekä kehitysympäristön. CLIPS-työkalun runko, eli *shell,* sisältää komponentit, joilla ohjelmia suoritetaan. Shell-ympäristö voidaan jaotella seuraaviin elementteihin:

- Fakta-lista ja ilmentymä-lista. Nämä tarjoavat jaetun muistin tiedolle.
- Tietokanta. Tämä sisältää kaikki säännöt.
- Rajapinnan moottori. Tämä ohjaa sääntöjen ajoa.

CLIPS-ohjelmointikielellä kirjoitettu ohjelma voi koostua säännöistä, faktoista ja objekteista. Rajapinnan moottori ohjaa, mitkä säännöt ajetaan ja milloin. CLIPS-ohjelmointikielellä luotu ohjelman ajo on tietolähtöinen, ja sitä edustavat faktat ja objektit. CLIPS-työkalussa on kolme tapaa esittää tietoa. Säännöt on pääasiassa tarkoitettu ilmaisemaan kokemukseen perustuvaa tapaa ratkaista ongelma. Funktiot ja olio-ohjelmointi on tarkoitettu ilmaisemaan proseduraalista tietoa. Ohjelmia voidaan luoda käyttämällä pelkästään sääntöjä, pelkästään objekteja tai molempien yhdistelmää. CLIPS-työkalu on suunniteltu toimimaan yhdessä muitten ohjelmointikielien kanssa, kuten C ja Java [11]. Sovelluksen iOS-sovellus hyödyntää iOS-alustalle kehitettyä versiota CLIPS-työkalusta, ja se on toteutettu Objective C -ohjelmointikielellä.

CLIPS-työkalusta on saatavilla versiot seuraaville ympäristöille:

- .NET
- JNI (Java Native Interface)
- iOS
- CGI (Common Gateway Interface).

3.8 Sääntömoottoreiden vertailu

Suorituskyvyltään paras sääntömoottori on Drools, sillä se tukee parannettua versiota ReteOO-algoritmista ja monisäikeistä ajoa. Muut sääntömoottorit käyttävät alkuperäistä Rete-algoritmia. Jess on ainoa sääntömoottori, joka Droolsin lisäksi tukee monisäikeistä ajoa. CLIPS ja Drools ovat ainoat sääntömoottorit, jotka tukevat sekä iOS- että Android -alustoja, mutta Droolsin tuki on toteutettu lakkautetulla RoboVM-kääntäjällä. Jess on ainoa sääntömoottori, jonka lisenssi ei tarjoa maksutonta käyttöä kaupallisessa suljetun lähdekoodin sovelluksessa. Droolsin tuki ja yhteisö ovat selvästi vertailun parhaita. CLIPS ja Jess tarjoavat hyvän tuen ja yhteisön. JEOPSin kehitys näyttää lakanneen jo vuonna 2000. Sen tuki ja yhteisö ovat erittäin heikkoja. Taulukossa 1 vertaillaan eri sääntömoottoreiden ominaisuuksia.

Taulukko 1. Sääntömoottoreiden ominaisuuksien vertailu.

	Tuetut mobii-lialustat	Raja-pinnan kieli	Algo-ritmi	Moni-säikei-syys	Li-senssi	Tuki	Yhteisö
Clips	iOS, Android	Java, Swift	Rete	ei	Public domain	hyvä	kohta-lainen
Drools	Android	Java	Re-teOO	kyllä	Apache Soft-ware Li-cence 2.0	erin-omai-nen	erin-omai-nen
JEOPS	Android	Java	Rete	ei	LGLP	erittäin huono	erittäin huono
Jess	Android	Java	Rete	kyllä	maksul-linen	hyvä	hyvä

4 Älykkäät avustajat diabeteksen hoidon apuna

Älykkäät avustajat ovat insinöörityössä käsiteltävän sovelluksen keskeisin ominaisuus. Ne tarjoavat potilaalle ja terveydenhuollon ammattilaiselle eri osa-alueisiin jaoteltua tietoa potilaan terveydentilasta sekä elämäntavoista, jotka vaikuttavat diabeetikon terveyteen. Näiden tietojen avulla terveydenhuollon ammattilainen kykenee määrittämään konkreettisia ja ymmärrettäviä hoitotavoitteita läpinäkyvästi potilaan kanssa kullekin hoidon osa-alueelle. Älykkäät avustajat tarjoavat niin pitkän aikavälin kuin lyhyen aikavälin tietoa ja mahdollistavat keskittymisen tiettyihin osa-alueisiin. Näin ne tehostavat jatkuvaa kehitystä. Potilaan näkökannalta älykkäät avustajat tarjoavat terveydenhuollon ammattilaisen ohjaamaa, ymmärrettävää ja toiminnallista ohjausta diabeteksen hoitoon. Älykkäät avustajat näyttävät potilaalle, kuinka hän suoriutuu tavoitteisiin nähden pitkällä ja lyhyellä aikavälillä. Älykkäät avustajat ovat diabeteslääkäreiden kehittämiä.

4.1 Älykkäiden avustajien luokat

Ensimmäinen älykkäiden avustajien luokista on kliiniset tavoitteet. Niillä pyritään integroimaan perinteinen hoito sovellukseen. Ne seuraavat hyvin tunnettuja tietoja, jotka vaikuttavat suoraan diabeetikon terveyteen, kuten hypoglykemioitten määrä. Nämä älykkäät avustajat keventävät terveydenhoidon ammattilaisen työkuormaa näiden arvojen seurannassa siirtämällä monitoroinnin ja tietojen analysoinnin sovellukselle. Potilaalle tämä tarjoaa terveydenhuollon ammattilaisen asettamia mukautettuja tavoitteita. Älykkäät avustajat tarjoavat oikea-aikaista apua asetetuissa tavoitteissa pysymiseen.

Toinen älykkäiden avustajien luokista on elämäntapatavoitteet. Ne auttavat potilasta seuraamaan elämäntapoja, jotka auttavat pitämään diabeteksen kurissa. Esimerkiksi säännöllinen verensokerin mittaaminen ja liikunta ovat tällaisia tavoitteita. Elämäntapatavoitteet määritetään yhdessä terveydenhuollon ammattilaisen kanssa, ja älykäs avustaja auttaa pysymään näissä tavoitteissa. Ne tarjoavat ammattilaiselle tavan ohjata potilasta elämään terveellisempää arkea ja mahdollistavat elämäntapojen ohjaamisen osana kokonaisvaltaista hoitoa. Älykkäät avustajat siirtävät jatkuvan monitoroinnin taakan sovellukselle. Potilaalle tämä mahdollistaa terveydenhuollon ammattilaisen asettamia mukautettuja elämäntapatavoitteita ja oikea-aikaista apua niissä pysymisessä niin pidemmällä kuin lyhyellä aikavälillä.

Kolmas älykkäiden avustajien luokista on terveyskatsaukset. Ne auttavat käsittelemään suurta tiedon määrää ja tunnistamaan toistuvat haitalliset tapahtumat, kuten hypoglykemiat liikunnan jälkeen. Terveyskatsaukset tarjoavat mahdollisuuden terveydenhuollon ammattilaiselle auttaa potilasta välttämään havaitut haitalliset tapahtumat, ja ne antavat ammattilaisille tavan havaita näitä tapahtumia, jotka aikaisemmin saattoivat jäädä tunnistamatta. Terveyskatsaukset vähentävät säännöllisten henkilökohtaisten tapaamisten tarvetta ja vapatuttavat aikaa henkilökohtaista hoitoa varten. Potilaalle terveyskatsaukset tarjoavat apua terveydelle haitallisten tapahtumien välttämiseksi ilmoittamalla niitten tapahtuessa ja tarjoamalla neuvoja, kuinka ne voi jatkossa välttää ja mikä niitten aiheuttaja oli. Terveyskatsaukset näyttävät pidemmän aikavälin kehityksen siitä, kuinka potilas on onnistunut välttämään haitallisia tapahtumia.

4.2 Älykkäiden avustajien rakenne

Älykäs avustaja ladataan iOS-sovellukseen JSON-muodossa. Määritys on jaettu kolmeen osioon: yleiset määritykset, lokalisoidut resurssit ja asiakkaan asetukset. Yleiset määritykset kertovat, kuinka älykäs avustaja tunnistetaan, käsitellään ja esitetään iOS-sovelluksessa. Suoritusindikaattoreita käytetään käyttäjän suoriutumisen seurantaan. Lokalisoidut resurssit sisältävät eri kielillä käännetyt tekstit, joita näytetään käyttäjälle syötteessä, ilmoituksissa ja älykkään avustajan tiedoissa. Pakolliset elementit ovat title ja description. Title on muotoilematonta tekstiä ja esittää älykkään avustajan otsikkoa. Description on markdown-muodossa olevaa tekstiä, sillä markdown tarjoaa mahdollisuuden muotoilla tekstiä älykkään avustajan luonnin yhteydessä. Se tarjoaa dynaamisen tavan esittää tekstiä iOS-sovelluksessa. Description kertoo älykkään avustajan toiminnasta ja käyttötarkoituksesta. Esimerkkikoodissa 1 on ote lokalisoidun määrityksen JSON-rakenteesta.

```
{
  /* other definition elements */
  ...

  /* Localized resources */
  "localizations": {
    "en": {
      "title": "Otsikko avustajalle",
      "description": "Avustajan tiedot"
    },
    "sv": {
      "title": "Samma på svenska",
      "description": "..."
    }
```

```
  }

  /* other definition elements */
  ...
}
```

Esimerkkikoodi 1. Ote lokalisoidun määrityksen JSON-rakenteesta.

Asiakkaan asetukset määrittävät kunkin älykkään avustajan käyttäjäkohtaisia määrityksiä. Asetukset vaikuttavat älykkään avustajan toimintaan ja ulkoasuun. Jokainen asetus noudattaa samaa rakennetta. Rakenteeseen kuuluu asetuksen nimi, joka on määritetty merkkijonomuodossa ja on pakollinen jokaiselle asetukselle. Mandatory-asetus kertoo, onko asetus pakollinen avustajalle, ja se on tyypiltään Boolean. ValueType-asetus kertoo, mikä asetuksen arvon tyyppi on, ja se on pakollinen määritys asetukselle. DefaultValue-asetus kertoo asetuksen oletusarvon, ja se on pakollinen. Validation-asetus kertoo käytettävät validointisäännöt asetuksen arvolle, ja se on tyypiltään lista validointisääntöjä. Unit-asetus kertoo asetuksen arvon yksikön. UserChangeable-asetus kertoo, pystyykö potilas vaihtamaan asetuksen arvoa, ja se on tyypiltään Boolean. ProChangeable-asetus kertoo pystyykö terveydenhuollon ammattilainen vaihtamaan asetuksen arvoa, ja se on tyypiltään Boolean. Esimerkkikoodissa 2 on ote asetuksen JSON-rakenteesta.

```
{
 "setting": "enabled",
 "valueType": "BOOLEAN",
 "defaultValue": false,
 "userChangeable": false,
 "proChangeable": true
}

{
 "setting": "target",
 "key": "<custom key>",
 "mandatory": true,
 "valueType": "INTEGER",
 "defaultValue": "<value>",
 "validation": [
   { "unit": "min", "min": 1, "max": 20}
 ],
 "userChangeable": false,
 "proChangeable": true
}
```

Esimerkkikoodi 2. Ote asetuksen JSON-rakenteesta.

Enabled-asetus on tyypiltään Boolean, ja se kertoo, onko älykäs avustaja otettuu käyttäjälle käyttöön terveydenhuollon ammattilaisen toimesta. Notification.feed-määritys on tyypiltään Boolean, ja se kertoo, näytetäänkö älykkään avustajan luomia viestejä iOS-

sovelluksen syötenäkymässä (kuva 3 s. 24). Notification.system on tyypiltään Boolean ja se kertoo, näytetäänkö älykkään avustajan luomia viestejä käyttöjärjestelmäilmoituksina (kuva 2 s. 23). Molemmat notification-aseutkset voivat olla asetettuna pois. Notification.system-asetuksen ollessa päällä myös notification.feed-asetuksen tulee olla päällä. Notification.system-asetus voi olla pois päältä notification.feed-asetuksen ollessa päällä. Käyttäjä pystyy muuttamaan notification-asetuksia älykkään avustajan omasta näkymästä (kuva 5 s. 27). Esimerkkikoodi 3 on ote asiakkaan asetusten JSON-rakenteesta.

```
{
 /* other definition elements */
 ...

 /* Settings */
 "settings": [
   {
     "setting": "enabled",
     ...
   }
 ]

 /* other definition elements */
 ...
}
```

Esimerkkikoodi 3. Ote asiakkaan asetusten JSON-rakenteesta.

Elämäntapa- ja kliininen tavoite -luokkaan kuuluvien älykkäitten avustajien tulee määrittää yleisten pakollisten asetuksien lisäksi tavoitearvo, jota vasten suoritusindikaattoria voidaan mitata. Esimerkkikoodi 4 on ote tavoite-asetuksen JSON-rakenteesta.

```
{
 "setting": "target",
 "key": "<custom key>",
 "mandatory": true,
 "valueType": "INTEGER",
 "defaultValue": "2700000",
 "validation": [
   { "unit": "min", "min": 1, "max": 20}
 ],
 "userChangeable": false,
 "proChangeable": true
}
```

Esimerkkikoodi 4. Ote tavoite-asetuksen JSON-rakenteesta.

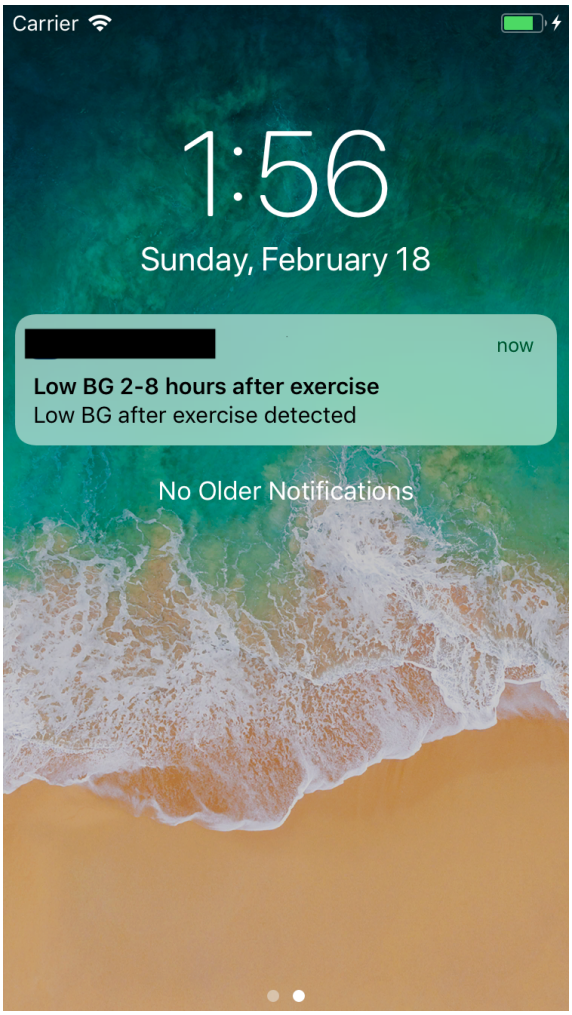
Älykkäät avustajat ladataan sovellukseen palvelimelta. Jokaisella älykkäällä avustajalla on omistajana palveluntarjoaja. Palveluntarjoaja on tyypillisesti lääkäriasema. Kaikki palveluntarjoajan omistamat älykkäät avustajat ladataan kaikille sen asiakkaille. Älykkäät avustajat alkavat tuottaa tietoja omien määritystensä mukaan käyttäjän toiminnasta. Nämä tiedot ovat joko kertoja, jolloin tietty tapahtuma on esiintynyt, tai monipuolisempia suoriutumisindikaattoreita. Tapahtuman kertoihin perustuva tieto on esimerkiksi, kuinka monta kertaa käyttäjä on mitannut verensokerinsa. Monipuolisempi suoriutumisindikaattori kertoo esimerkiksi, kuinka monta prosenttia käyttäjä on suorittanut tavoitteestaan, kun tavoite on viisi verensokerimittausta päivässä.

Älykkäät avustajat eivät näy käyttäjälle, ennen kuin terveydenhuollon ammattilainen ottaa älykkään avustajan käyttäjälle käyttöön. Ammattilainen näkee älykkäitten avustajien tuottamat tiedot, joiden perusteella hän tekee päätöksen tarvitaanko älykäs avustaja potilaalle käyttöön. Kun se otetaan käyttöön, näkee myös potilas älykkään avustajan tuottamat tiedot. Tämän käyttöönottotavan tarkoituksena on, että potilas kehittyisi niissä asioissa, joissa ammattilaisen mielestä potilaan tulisi kehittyä pärjätäkseen paremmin diabeteksen kanssa.

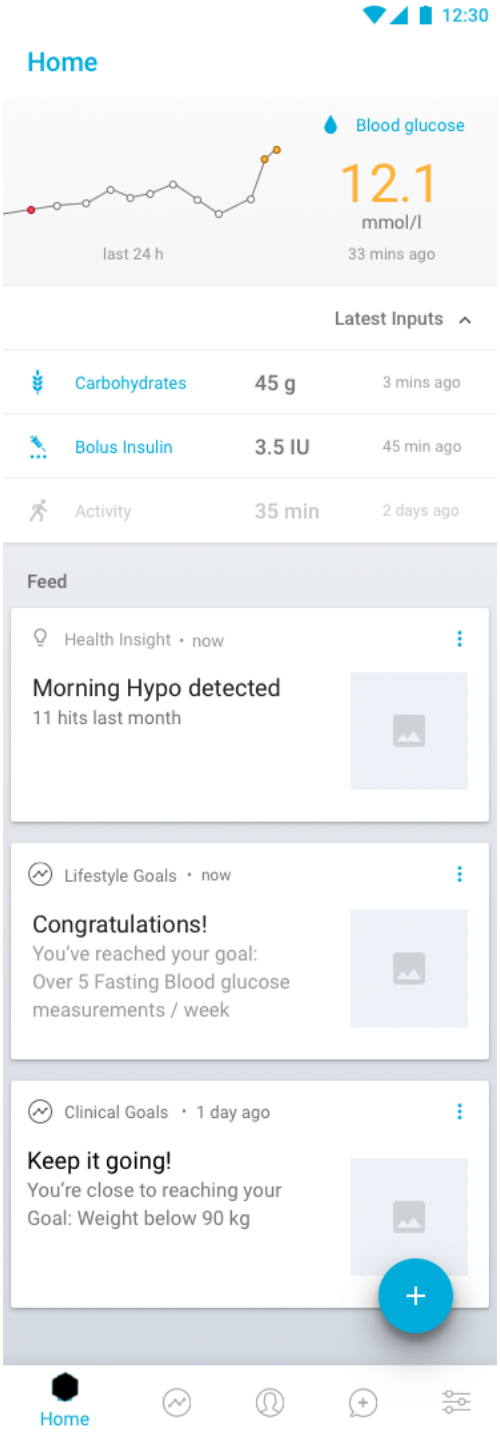
4.3 Älykkäiden avustajien esittäminen

Älykkäiden avustajien tuottama tieto esitetään käyttäjälle syötteessä, järjestelmäilmoituksina, yhteenvetoina hoitosuunnitelmassa ja älykkään avustajan omassa näkymässä. Käyttäjä saa syötteeseen ja järjestelmäilmoituksina tilapäivityksiä älykkäästä avustajasta. Käyttäjän sovellukseen syöttämät mittaukset analysoidaan heti, kun käyttäjä syöttää tietoja sovelluksen syöttö-näkymästä tai esimerkiksi siirtämällä verensokerimittauksia tuetusta Bluetooth-verensokerimittarista. Hoitosuunnitelmanäkymä ja älykkään avustajan oma näkymä tarjoavat välittömästi ajantasaista tietoa potilaan tilan muutoksista. Käyttäjä saa samalla tilapäivityksiä edetessään kohti ja saavuttaessaan tavoitteita sekä terveyskatsausten havaitessa terveydelle haitallisia tapahtumia käyttäjän tiedoista. Käyttäjän syöte on osa kotinäkymää. Kotinäkymän luokka perii UIViewController-luokalta. Kotinäkymän luokalle asetetaan UITableView-alanäkymä, joka peittää koko kotinäkymän. UITableView-alanäkymä täytetään hakemalla tietokannasta syötteeseen tilapäivitykset, joilla alustetaan tilapäivitys-luokat. Tilapäivitys-luokka noudattaa protokollaa, joka sisältää funktion, jolla kotinäkymän luokka saa haettua UITableView-alanäkymälle tilapäivitys-oliosta UITableViewCell-tyyppisen instanssin. Kuvassa 2 on älykkään avustajan

luoma järjestelmäilmoitus, ja kuvassa 3 on sovelluksen kotinäkymä, jossa käyttäjän syöte sijaitsee.



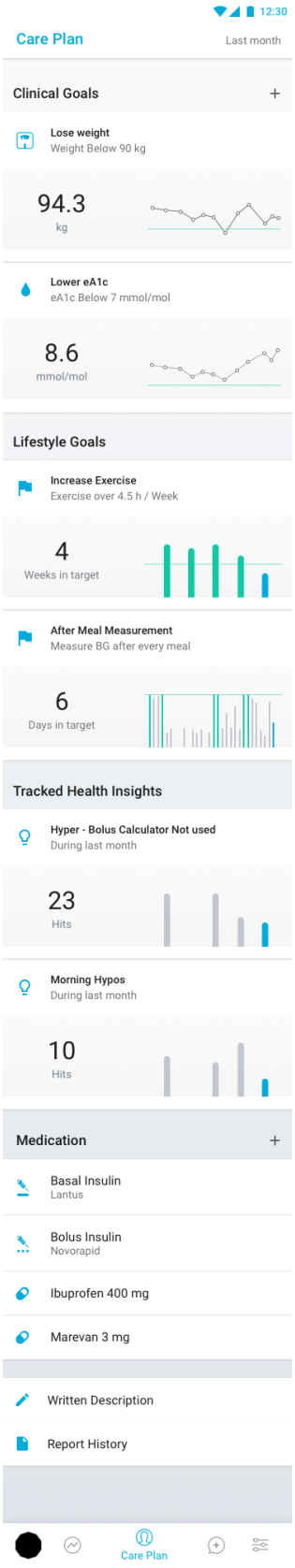
Kuva 2. Älykkään avustajan luoma järjestelmäilmoitus iOS-käyttöjärjestelmän lukitusnäytöllä.



Kuva 3. Käyttäjän kotinäkymä, jossa syöte sijaitsee (Feed-otsikon alla).

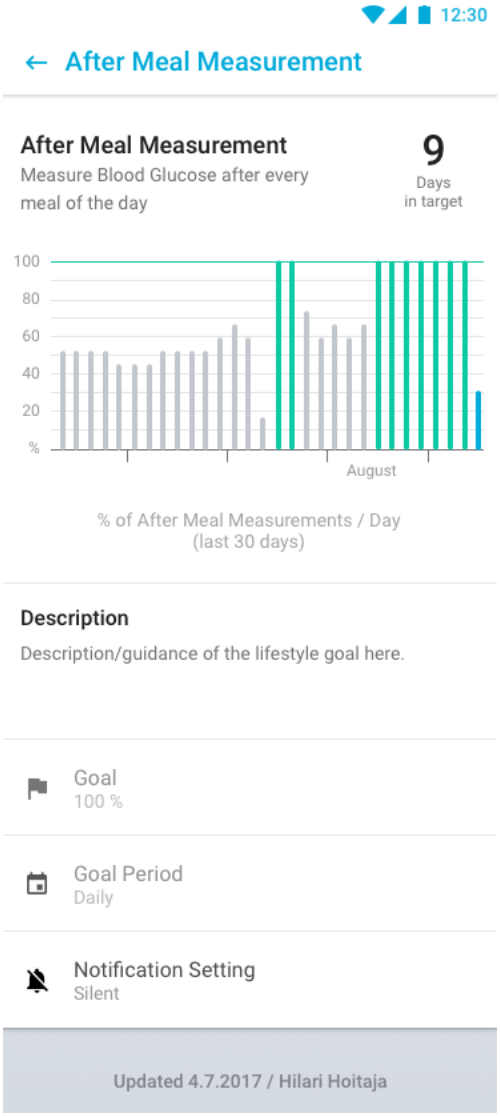
Hoitosuunnitelmanäkymä näyttää yhteenvedon käytössä olevista älykkäistä avustajista, ja ne on jaoteltu omiin ryhmiinsä älykkään avustajan luokan mukaan. Yhteenvedossa näytetään älykkään avustajan nimi, mahdollinen tavoite ja nykyinen tila numeraalisena arvona sekä diagrammina. Hoitosuunnitelmanäkymän luokka perii UIViewController-luo-

kalta. Hoitosuunnitelmanäkymän luokalle asetetaan UITableView-alanäkymä, joka peittää koko hoitosuunnitelmanäkymän. UITableView-alanäkymä täytetään hakemalla tietokannasta käytössä olevat älykkäät avustajat, joilla alustetaan älykkään avustajan hoitosuunnitelmanäkymän luokat. Se noudattaa protokollaa, joka sisältää funktion, jolla hoitosuunnitelmanäkymän luokka saa haettua UITableView-alanäkymälle avustajan hoitosuunnitelmanäkymä-oliosta UITableViewCell-tyyppisen instanssin. Lisäksi protokolla sisältää funktion, joka tarjoaa hoitosuunnitelmanäkymän luokalle UIViewController-luokan, johon siirrytään, kun käyttäjä painaa älykkään avustajan solua. Hoitosuunnitelmanäkymässä älykkään avustajan solun koskettaminen vie älykkään avustajan omaan näkymään. Kuvassa 4 on hoitosuunnitelmanäkymä.



Kuva 4. Käyttäjän hoitosuunnitelma, jossa näytetään yhteenveto käytössä olevista älykkäistä avustajista.

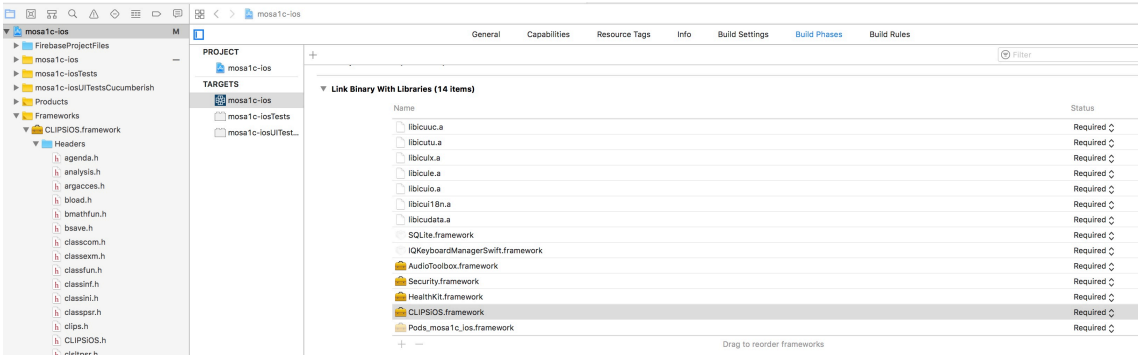
Älykkään avustajan omassa näkymässä näytetään yhteenvedon tietojen lisäksi älykkään avustajan kuvaus, tavoitteen tarkasteluväli ja ilmoitusasetus. Ilmoitusastus kontrolloi, näytetäänkö älykkään avustajien luomia viestejä syötteessä ja järjestelmäilmoituksina. Potilas voi muuttaa ilmoitusasetusta. Näkymän luokka perii UIViewController-luokalta. Näkymän luokka alustetaan viittauksilla, jotka tarjoavat tiedot diagrammia, käännöksiä, älykkään avustajan tilaa, asetuksia ja kuvausta varten. Näkymän luokalle on asetettu ScrollView-tyyppinen alanäkymä, joka peittää koko näkymän. ScrollView-tyyppiselle alanäkymälle on asetettu StackView-tyyppinen alanäkymä, joka peittää koko Scroll-View-alanäkymän. Kuvassa 5 on älykkään avustajan oma näkymä.



Kuva 5. Älykkään avustajan oma näkymä.

4.4 CLIPS-ympäristön integroiminen iOS-sovellukseen

Ensimmäinen vaihe CLIPS-moottorin integroimisessa iOS-sovellukseen on CLIPSIOS-paketin lataaminen osoitteesta http://www.clipsrules.net/?q=Downloaads/CLIPSiOS. Tämä paketti tarjoaa kehyksen, englanniksi *framework*, ohjeet ja kolme esimerkki-iOS-sovellusta, jotka käyttävät CLIPS-moottoria. Seuraavaksi kehys lisätään XCode-projektiin valitsemalla *target*-valikosta sovellus, minkä jälkeen valitaan *Build Phases* -välilehti. Sitten valitaan *Link Binary With Libraries* -osien alta plus-painikkeella .framework-tiedosto ladatusta paketista. Kuvassa 6 on integroitu CLIPS-kehys iOS-projektiin.



Kuva 6. CLIPS-kehys integroituna iOS-projektiin XCodessa.

Seuraava vaihe on ympäristön luominen. Se tehdään kutsumalla CreateEnvironment-funktiota. Se palauttaa osoittimen ympäristöön. Tätä osoitinta käytetään jatkossa kutsujen tekemiseen. Ympäristö voidaan tuhota kutsumalla DestroyEnvironment-funktiota ja sen muisti voidaan tyhjentää kutsumalla Clear-funkitota [19]. Kun ympäristö on luotu, Load-funktiolla ladataan sääntöjä ympäristöön. Parametrina annetaan tiedostopolku merkkijonomuodossa, joka osoittaa ASCII- tai UTF-8 -tekstitiedostoon. Merkkijonomuodossa oleva sääntö voidaan tallentaa UTF-8 tekstitiedostoksi String-luokan write-funktiolla. Load-funktio palauttaa tiedon onnistuiko tiedoston lataaminen ilman virheitä vai oliko tiedoston jäsentelyssä tai avaamisessa ongelma. DestroyEnvironment- , Clear- ja Load -funktiot ottavat parametrina aikaisemmin luodun osoittimen ympäristöön.

Kun ympäristö on luotu ja säännöt ladattu, seuraava vaihe on faktojen ja instanssien luominen. AssertString-funktiolla asetetaan faktoja, ja se ottaa parametreina ympäristön

ja character array -muodossa olevan faktatekstin. Merkkijonomuodossa oleva teksti voidaan muuttaa character array -muotoon String-luokan cString-funktiolla. Fakta voi olla joko järjestetty tai deftemplate-muodossa. Erimerkki järjestetystä faktasta on ”(värit punainen vihreä sininen)” ja deftemplate-muodossa olevasta (henkilö (nimi Matti Meikäläinen)(ikä 40))” [19, 3.3.1]. Mikäli fakta luodaan onnistuneesti tai on jo luotuna, palauttaa AssertString-funktio viittauksen faktaan. Viittausta voidaan myöhemmin käyttää faktan poistamiseen muistista. Tallennettaessa viittausta tulee RetainFact-funktiota kutsua, jotta viittaus pysyy voimassa faktan poistamisenkin jälkeen. MakeInstance-funktiolla asetetaan instansseja, ja se ottaa parametrina ympäristön sekä character array -muodossa olevan instanssitekstin. Esimerkki instanssitekstistä on ”(piste15 of PISTE (x 13) (y 14))”. Mikäli instanssi luodaan onnistuneesti, palauttaa MakeInstance-funktio viittauksen instanssiin. Viittausta voidaan myöhemmin käyttää instanssin poistamiseen muistista. Tallennettaessa instanssia tulee RetainInstance-funktiota kutsua, jotta viittaus pysyy voimassa instanssin poistamisenkin jälkeen. UnmakeInstance-funktiolla voi poistaa instanssin, ja Retract-funktiolla voi poistaa faktan muistista.

Kun faktat ja instanssit on luotu, seuraava vaihe on sääntöjen suorittaminen. Sääntöjä suoritetaan Run-funktiolla. Sille annetaan parametreina ympäristö ja long-muodossa oleva numeraalinen raja, joka määrittää, montako sääntöä ajetaan. Jos raja on negatiivinen, suoritetaan kaikki suoritettavat säännöt. Funktio palauttaa sääntöjen suoritettua niitten lukumäärän.

Kun säännöt on suoritettu, seuraava vaihe on tulosten evaluointi. EnvEval-funktiolle [19, 4.1.9] annetaan ensimmäisenä parametrina ympäristö, joka luotiin aikaisemmin. Toisena parametrina annetaan evaluoitava ilmaisu. Ilmaisua "(find-all-facts ((?f action)) TRUE)" käytetään löytämään uudet faktat säännöistä, joita käytetään sääntömoottorin tuloksena. Kolmantena parametrina annetaan viittaus CLIPS-ohjelmistokehyksessä määritettyyn DATA_OBJECT-rakenteen omaavaan olioon. Tämä voidaan luoda Swiftillä UnsafeMutablePointer<dataObject>.allocate(capacity: 1) -komennolla. Kutsumalla EnvEval-funktiota se palauttaa DATA_OBJECT-olioon löydetyt faktat. EnvEval-funktio palauttaa arvon 1, mikäli evaluointi onnistui, ja arvon 0, mikäli evaluointi epäonnistui [19, 4.1.10].

Kun tulokset on evaluoitu, seuraava vaihe on uusien faktojen käsittely. DATA_OBJECT-olion value-kenttä sisältää uudet faktat multifield-muodossa. Multifield on CLIPS-ohjelmistokehyksen rakenne, joka sisältää tiedon sen pituudesta ja arvot. UnsafeMutablePointer-muotoon tallennetaan viittaus value-kentän arvoihin. Seuraavaksi luodaan uusi UnsafeMutablePointer DATA_OBJECT-rakenteella. EnvGetFactSlot-funktiolle annetaan

parametreina viittaus ympäristöön ja faktaan, haettavan kentän nimi ja uusi DATA_OBJECT-olio, jonne tiedot tallennetaan. Faktat saadaan merkkijonomuotoon kutsumalla UnsafeMutablePointer-instanssin .pointee.toString-funktiota. Tämän jälkeen merkkijonomuodossa olevat faktat käsitellään pääsovelluksessa.

5 Yhteenveto

Insinöörityössä selvitettiin, kuinka tekoälyä on hyödynnetty terveydenhuollossa ja kuinka tekoälyä voisi hyödyntää diabeteksen itsehoidossa. Työssä havaittiin, että tekoälyllä on paljon mahdollisuuksia edistää terveydenhoitoa. Sovellutuksia on jo olemassa useita, kuten IBM Watson ja Google DeepMind. Ne ovat osoittaneet hyödyllisyytensä kustannusten alentamisessa, hoidon parantamisessa ja lääkärin työn tehostamisessa ja helpottamisessa.

Sääntömoottoria päätettiin hyödyntää diabeteksen itsehoidossa. Tavoitteena oli löytää sopiva sääntömoottori, jonka avulla terveydenhuollon ammattilaisen osaamista voisi tuoda mobiilisovellukseen ja integroida se osaksi tuotekokonaisuutta. Ratkaisuksi päätettiin valita Android- ja iOS-alustoille yhteensopiva sääntömoottori. Insinöörityössä vertailtiin eri vaihtoehtoja, päädyttiin CLIPS-sääntömoottoriin ja selvitettiin, kuinka CLIPS-sääntömoottori saadaan osaksi iOS-sovellusta.

Työssä jäi selvittämättä, millaiseen suorituskykyyn CLIPS kykenee siinä suoritettavilla säännöillä. Lisäksi jäi vertailematta, miten palvelimella suoritettava sääntömoottori olisi suoriutunut mobiililaitteessa suoritettavaan verrattuna ja kuinka palvelimella suoritettavan sääntömoottorin tuloksia voitaisiin synkronoida reaaliajassa mobiililaitteeseen. CLIPS-integraation kuvaus jäi myös rajalliseksi osittain siksi, että yrityksen salassa pidettävää tietoa ei insinöörityöhön voinut sisällyttää.

Lähteet

1 Koski, Sari; Kurkela, Olli; Ilanne-Parikka, Pirjo; Rissanen, Pekka. 2017. Diabeteksen kustannukset Suomessa 2002-2011 . Verkkoaineisto. < https://www.diabetes.fi/files/9237/Diabetes_lukuina_2017_flyer.pdf> Luettu 20.10.2017.

2 Hansen, Thomas. 2016. The role of AI in Healthcare – an in-depth guide. Verkkoaineisto. < https://www.linkedin.com/pulse/role-ai-healthcare-in-depth-guide-thomas-riisgaard-hansen/> 6.9.2016. Luettu 21.10.2017.

3 Sweeney, Evan. 2017. AI's success relies on solving healthcare's data problem. Verkkoaineisto. < http://www.fiercehealthcare.com/analytics/ai-s-success-healthcare-relies-solving-a-significant-data-problem > Luettu 21.10.2017.

4 Afzal, Athar. 2017. Augmented Intelligence, NOT Artificial Intelligence. Verkkoaineisto. < https://www.ibm.com/blogs/social-business/2017/01/31/augmented-intelligence-not-artificial-intelligence/ > Luettu 21.10.2017.

5 Lavenda, David. 2016. Artificial Intelligence vs. Intelligence Augmentation. Verkkoaineisto. < https://www.networkworld.com/article/3104909/software/artificial-intelligence-vs-intelligence-augmentation.html > Luettu 21.10.2017.

6 Bloch-Budzier, Sarah. 2016. NHS using Google technology to treat patients. Verkkoaineisto. < http://www.bbc.com/news/health-38055509> 22.11.2016. Luettu 20.10.2017.

7 Stella, Nwibgo; Chuks, Agbo Okechuku. Expert System: A Catalyst In Education Development in Nigeria. Verkkoaineisto. < http://www.hrmars.com/admin/pics/261.pdf> Luettu 31.1.2018.

8 The Rule Engine. 2011. Chapter 1. Verkkoaineisto. The JBoss Drools team. < https://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/html/ch01.html> Luettu 31.1.2018.

9 Mukherjee, Chinmoy. 2015. Artificial Intelligence: Rules Engines for Mobile Platforms. E-kirja. Lulu Press.

10 Rudolph, George. 2008. Some Guidelines For Deciding Whether To Use A Rules Engine. Verkkoaineisto. < http://www.jessrules.com/guidelines.shtml> Luettu 15.1.2018.

11 Giarratano, Joseph. CLIPS User's Guide V.6.30 . Verkkoaineisto. < http://clipsrules.sourceforge.net/documentation/v630/ug.pdf > Luettu 20.10.2017.

12 Knuth, Donald. 1968. The Art of Computer Programming. Yhdysvallat: Adisson-Wesley.

13 Miranker, Daniel P. 1987. TREAT: A Better Mate lgorith for AI Production Systems. Verkkoaineisto. < https://www.aaai.org/Papers/AAAI/1987/AAAI87-008.pdf> Luettu 11.1.2018.

14 Miranker, Daniel P. 1987. A Better Mate lgorith for AI Production Systems; Long Version. Verkkoaineisto. <ftp://ftp.cs.utexas.edu/pub/AI-Lab/tech-reports/UT-AI-TR-87-58.pdf> . Luettu 11.1.2018.

15 Filho, Carlos Figueira. 2000. JEOPS - The Java Embedded Object Production System – User's Manual. Verkkoaineisto. <https://www.cin.ufpe.br/~jeops/v20/usermanV2.html>. Luettu 10.1.2018.

16 The Jess FAQ. Verkkoaineisto. Sandia National Laboratories. < http://www.jess-rules.com/jess/FAQ.shtml> Luettu 15.1.2018.

17 Jess Language Basics. Verkkoaineisto. Sandia National Laboratories. < http://www.jessrules.com/jess/docs/71/basics.html> Luettu 15.1.2018.

18 Embedding Jess in a Java Application. Verkkoaineisto. Sandia National Laboratories. < http://www.jessrules.com/jess/docs/71/embedding.html> Luettu 15.1.2018.

19 Giarratano, Joseph. CLIPS Advanced Programming Guide V.6.30 . Verkkoaineisto. < http://clipsrules.sourceforge.net/documentation/v630/apg.pdfd> Luettu 20.10.2017.