TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Markkus Koodi

Targa kodu automatsioonid loogilise programmeerimisega Home Assistant platvormil

Bakalaureusetöö (9 EAP)

Juhendaja(d): Jakob Mass, MSc

Tartu 2022

Targa kodu automatsioonid loogilise programmeerimisega Home Assistant platvormil

Lühikokkuvõte:

Tuntumate avatud lähtekoodiga targa kodu platvormidel puudub võimekus automatiseerida seadmeid eesmärgipõhiselt. Käesolev bakalaureusetöö loob rakenduse millega on võimalik eesmärgipõhiselt automatiseerida Home Assistant (HA) platvormile integreeritud seadmeid kasutades Prologi. Lisaks analüüsitakse loogilise programmeerimise võimekust automatiseermisel ning uuritakse millistel juhtudel on HA automatiseerimise süsteem parem kui valminud rakendus.

Võtmesõnad:

Kodu automatiseermine, tark kodu, Prolog, loogiline programeerimine, Home Assistant

CERCS: P175 Informaatika

Smart home automation with logic programming on Home Assistant platform

Abstract:

Well known open-source smart home platforms do not provide features to automate their devices using goal-based logic. This bachelor’s thesis will build an application which enables to create goal-based automations in Prolog for devices and services that have been integrated to Home Assistant (HA). In addition this thesis will analyze how capable is the automation system that uses logic programming and in which cases the HA automation system is better than the built application.

Keywords:

Smart home, Home Assistant, home automation, Prolog, logical programming

CERCS: P175 Informatics

Sisukord

1. Sissejuhatus

Interneti üha kasvav laialdane kättesaadavus, tehnoloogia areng ning mikrokiipide hindade langus on tekitanud võimaluse ühendada erinevaid seadmeid võrguga. Näiteks lülitid, telerid, külmikud, robottolmuimejad ja nii edasi. Kasutades võrku ühendatavaid seadmeid ning internetis olevaid teenuseid (näiteks progamm, mis otsib Nord pooli elektrihinnad) on võimalik luua endale tark kodu/nutikodu. Setz Brian [22] jt. kirjeldavad tarka kodu kui elumaju, mille seadmed (lülitid, lambid, kodutehnika jne.) on varustatud tehnoloogiaga mis võimaldavad seadmeid ühendada internetiga.

Käesolevas lõputöös on vaatluse all Home Assistant (HA) platvorm ning selle automatiseerimise süsteem ja selle võimalused. HA platvorm on avatud lähtekoodiga tarkvara kuhu on võimalik integreerida erinevaid targa kodu seadmeid ning neid juhtida või automatiseerida läbi platvormi [1]. Setz Brian [22] jt. artiklist tehtud uuringu järgi on HA platvorm üks tuntumaid avatud lähtekoodiga kodu automatiseerimise platvorme.

HA platvormil automatiseerimine [13] on üles ehitatud reeglipõhise süsteemi järgi. HA platvormil automatsioonid on üles ehitatud kolmes osas: *trigger*, *condition* ja *action*. Näiteks on automatsiooni reegel - kui väljas on pime ning toa aken on lahti, siis pane tuli kustu. Sellise automatsiooni puhul oleks tema *trigger*: “kui väljas on pime”. Kasutades valgussensorit, mis mõõdab väljas valgust, on võimalik seada *trigger*’is sensori numbriline väärtus mis defineerib, et millal on väljas pime. Kui sensor saavutab sellise oleku, siis automatsioon käivitatakse. *Condition* osas kontrollitakse “kas toa aken on lahti”. Selle kontrollimiseks kasutab automatsioon targas kodus seadistatud sensorit, mis jälgib akna hetke olekut. Kui aken on lahti, siis automatsioon liigub *action* staadiumisse, kus kasutaja on automatsiooni seadistanud mingi tegevuse, mida automatsioon tegema peaks. Antud näite puhul lülitatakse toa tuli välja.

Mainitud automatsiooni süsteemi võimekus on piiratud, sest süsteem pole võimeline looma eesmärgipõhist automatsiooni. Dirk E. Mahling [14] jt. kirjutavad, et eesmärgipõhise töövoo idee seisneb selles, et eesmärgi saavutamiseks kaardistatakse kõik vajalikud tingimused ja sammud. Näiteks inimene soovib, et telekat vaadates peab toas valgustus olema minimaalne. Selle täitmiseks hakkab automatsioon näiteks kontrollima seadmeid mis mõjutavad valgust. Kui seadme hetke olek tõstab ruumis valguse taset, siis automatsioon lülitab välja. Pärast seadme väljalülitamist kontrollib automatsioon uuesti, et kas soovitud tase on saavutatud. Kui soovitud taset pole saavutatud, siis automatsioon jätkab selle sama ringiga nii kaua kuni

soovitud eesmärk on saavutatud. Eelnevas näites toodud automatsiooni pole HA-s võimalik ehitada, sest reeglites olevad seadmed peavad olema rangelt defineeritud.

Üks võimalus eesmärgipõhisuse saavutamiseks on loogiline programmeermine ning selle keel Prolog. Genesereth Michael [19] jt. kirjeldavad loogilist programmeermist kui programmeerimise stiili mille kirjutamisel kasutatakse lausearvutust ja tema tehteid. Lõputöös hakatakse kasutama loogilise programmeerimise keelt Prolog. Prolog koosneb kolmest komponendist: faktidest, reeglitest ja nende omavahelistest relatsioonidest. Faktideks nimetatakse lauset, mis väidab midagi mis on tõene. Näiteks lauset "Köögiaken on lahti" võime Prologis faktina kirjutada kui "aken("köögiaken", lahti).". Reegel on lausete kogum, milles defineeritakse faktide vahelisi seoseid. Näiteks on kaks fakti "aken("köögiaken", lahti)" ning "must("köögiaken")". Loome reegli "kas_aken_pesemata(X) :- aken(X, _), must(X)". Antud reegli definitsioon on: "Aken X on pesemata, kui leidub aken X mis on must". Tehes päringu selle reegli pihta, kus väärtuseks anname X, siis Prolog hakkab otsima X-le väärtust millisel juhul reegli eesmärk täidetakse. Ehk selle reegli puhul tuleks X väärtuseks "köögiaken". Seda sellepärast, et eelnevalt defineeritud faktid kirjeldavad, et "köögiaken" on aken ning ta on must.

Antud lõputöö eesmärgiks on valmis ehitada automatiseerimise süsteem millega oleks võimalik HA platvormi seadmeid automatiseerida eesmärgipõhiselt kasutades Prologi, uurida loogilise programmeerimise võimekust targa kodu seadmete automatiseerimisel ja millistes olukordades on Prolog põhine automaatika eelistatud HA põhisele automaatikale. Lõputöö jooksul valmis rakendus, mida on võimalik ühendada HA külge ning kus on võimalik platvormile integreerituid seadmeid automatiseerida Prologiga. Valminud rakenduse abil analüüsiti loogilise programmeerimise võimekust, selle nõrkusi ning otsitakse vastus küsimusele kas loogiline programmeerimine eesmärgipõhisele automatiseerimisele on üldse mõistlik lähenemine.

Lõputöö esimeses peatükis räägitakse lõputöö taustast. Uuritakse kodu automatiseerimise platvorme, Home Assistanti ning lahendusi kus nutikate seadmete automatiseerimiseks on kasutatud loogilist programmeerimist. Teises peatükis on kirjeldatakse lõputöö jooksul valminud rakendust, selle arhitektuuri ja implementatsiooni. Kolmandas peatükis võrreldakse valminud rakenduse automatiseerimise süsteemi Home Assistant süsteemiga ning arutletakse üldiselt mis on valminud rakenduse tugevust, nõrkused jms.

2. Taust

Selles peatükis uuritakse avatud lähtekoodiga targa kodu platvorme, Home Assistanti ning lahendusi, kus nutikate seadmete automatiseerimiseks on kasutatud loogilist programmeerimist.

2.1 Targa kodu platvormid

Käesoleva lõputöö raames kasutatakse seadmete ja teenuste automatiseerimiseks avatud lähte koodiga Home Assistant platvormi. Peatükk räägib lähemalt populaarsematest avatud lähtekoodiga targa kodu platvormidest. Järgnev lõik tugineb Setz Brian [22] jt. uuringule, kus nad võrdlevad nelja kõige tuntumat avatud lähtekoodiga kodu automatiseerimise platvormi ning kirjutavad nendest lähemalt. Autorid valisid platvormid kasutades viite mõõdustiku:

- Kui palju on platvormi lähtekoodi arendatud.
- Kui populaarne on lähtekoodi repositoorium.
- Millal on tehtud viimane lähtekoodi muudatus.
- Dokumentatsiooni kvaliteet.
- Kui palju inimesi panustavad platvormi arendusse.

Nende viie mõõdustiku tulemusel on neli kõige paremat platvormi Home Assistant, Domoticz[1], openHAB[2] ja ioBroker[3]. Nende mõõdustike järgi oli Home Assistant kõige parem platvorm kolmes kategoorias. Lähtekoodi arendamises on artiklis antud Home Assistant platvormile hinnanguks 4.6, Domoticz'le 4.1, openHAB'le 3.2 ning ioBroker'le 3.2. Repositooriumi populaarsuses on artiklis antud Home Assistant hinnanguks 4.6, Domoticz'le 3.5, openHAB'le 2.7 ning ioBroker'le 2.7. Inimeste arvu kes panustavad platvormi arengusse on hinnatud järgnevalt: Home Assistant 5 punkti, Domoticz 3 punkti, openHAB ja ioBroker 1 punkt. Dokumentatsiooni kvaliteet ning viimane lähtekoodi muudatus oli kõigil neljal täpselt sama.

Domoticz on platvorm, mille esimene versioon avalikustati 2012 aasta lõpus. Sensoreid ja täiturmehhanisme on platvormile integreerida võimalik läbi MQTT. Antud platvormil on seadmeid võimalik automatiseerida kasutades selleks Domoticz enda poolt pakutavat Blockly reeglite ehitajat või näiteks Node-RED. Node-RED on tööriist kus on võimalik automatsioone ehitada graafiliselt [23].

[1] https://domoticz.com/
[2] https://www.openhab.org/
[3] https://www.iobroker.net/

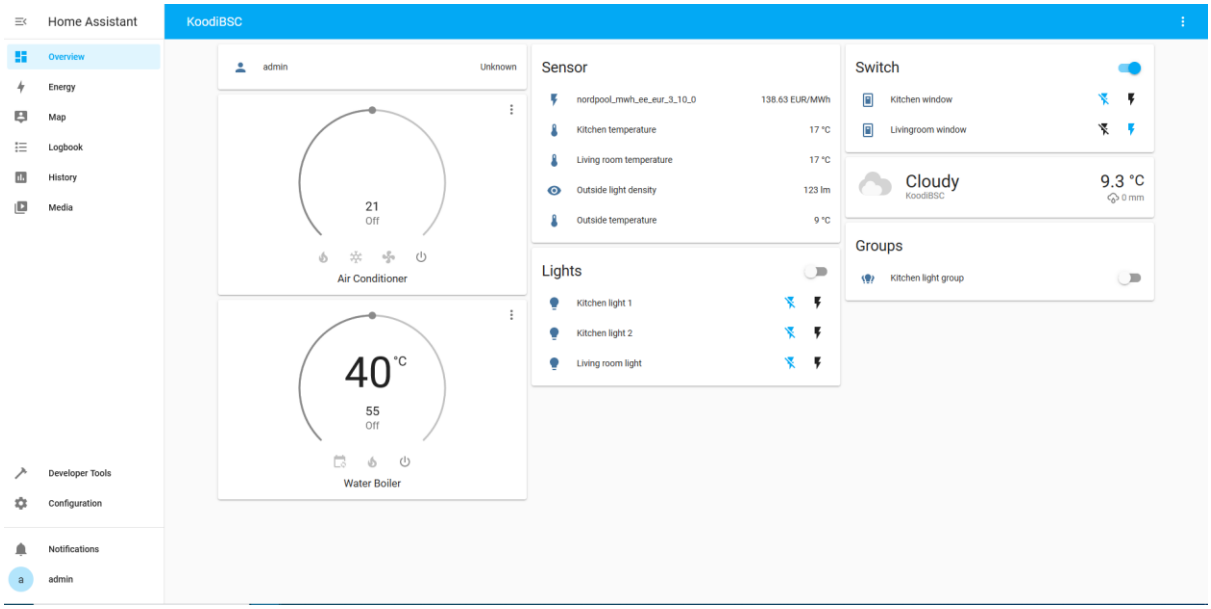
OpenHAB (pikemalt Open Home Automation Bus) on platvorm, mille arendamist alustati 2010. Antud platvormi üheks tugevuseks on, et see toetab palju rohkem seadmeid, teenuseid ja protokolle võrreldes Home Assistant platvormiga. OpenHAB toetab üle 3000 seadme või teenuse [24]. Home Assistant [3] platvorm toetab natuke üle 1850 seadme. Automatsioonide implementeerimine on võrreldes Home-Assistant platvormiga samasugune, mõlemal platvormil on võimalik ehitada automatsioone kas läbi kasutajaliidese või kirjutada automatsiooni fail kasutades selleks imperatiivset programmeerimist [13][25].

Viimane platvorm mida Setz Brian [22] jt. uurisid on ioBroker. Esimene versioon ioBroker platvormist lasti välja 2014 aastal. Kirjutamise ajal toetas platvorm 488 seadet või teenust mis on võrreldes OpenHAB ja Home-Assistant platvormiga väike arv. Platvormile on võimalik integreerida seadmeid ja teenuseid läbi adapterite. Adapterid suhtlevad objektide ja seisude andmebaasiga, kus esimene jätab meelde seadmete/teenuste metaandmed ja sellega kaasnevad konfiguratsioonid ning teine jätab meelde seadmete/teenuste seisud. Platvormil on võimalik automatiseerida seadmeid/teenuseid sarnaselt nagu OpenHAB ja Home-Assistant platvormiga, kuid on võimalik kasutada ka Node-RED ja Blockly süsteemi [23][26].

Antud artikli põhjal on Home Assistant põhjendatud valik, sest kommuun kes seda arendab on suur ja see on võrreldes teiste platvormidega laialdaselt kasutuses. Näiteks Tartu Ülikooli Arvutitehnoloogia Instituudi värkvõrgu laboris, mis IoT (lüh. inglise keelsest sõnast *Internet of Things* ehk asjade internet) mõiste populariseerimisega tegeleb, on kasutusel just Home Assistant platvorm.

2.2 Home Assistant

Home Assistant [1] (HA) on avatud lähtekoodiga kodu automatiseerimise platvorm, mille funktsionaalsus võimaldab tuua kokku kõik seadmed ja teenused, mida on võimalik üle võrgu juhtida, need automatiseerida ning kuvada kasutajaliideses. Projekt sai alguse aastal 2013, kui Paulus Schousten lõi esialgse Pythonis kirjutatud rakenduse ning avalikustas selle Githubis [17, 18]. Platvormi tuuma ehitamisel on kasutatud Pythonit koos modulaarse lähenemisega, mis võimaldab platvormil lähtekoodi muutmata implementeerida seadmete ja teenuste ühilduvust [2]. 10. detsembri seisuga on platvormile implementeeritud ühilduvus 1858 erineva seadme ja teenusega, nende seas tuntumad on näiteks Amazon Alexa, Apple TV ja Google Maps [3].



Joonis 1. Home Assistant kasutajaliides

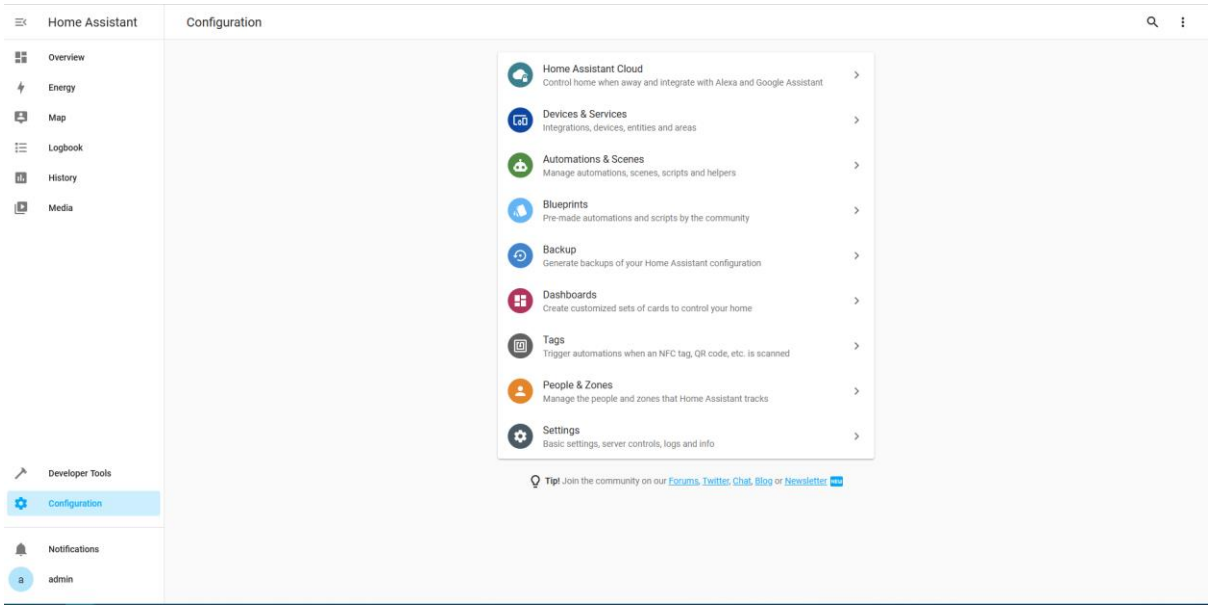
Joonisel 1 on kujutis üles seatud Home Assistant platvormi kasutajaliidesest. Jooniselt on näha erinevaid seadmeid ja teenuseid mis platvormil jooksevad. Joonisel oleva platvormi seadmeteks on õhukonditsioneer, tule lülitid jne. Platvormil integreeritud teenused on hetke ilmaolud ning Nord pooli hetke elektrihind. Järgmises peatükis kirjutatakse HA platvormi üles seadmisest, andmemudelist ning platvormi poolt pakutavatest rakendusliidestest.

2.2.1 Home Assistant andmemudel ja rakendusliides

Käesolev peatükk käsitleb Home Assistant platvormi konfigureerimist, andmemudelit ning räägib lähemalt platvormi rakendusliidestest.

2.2.1.1 Platvormi konfigureerimine

HA platvormi dokumentatsioon kirjutab, et kasutajaliidese kaudu on võimalik üles seada suurem osa platvormist [8]. Kasutajaliideses on sul võimalik integreerida seadmeid ja teenuseid, automatsioone, varundada vanu konfiguratsioone jms.



Joonis 2. Seadistaja kasutajaliideses

Joonisel 2 on kujutis kasutajaliideses olevast vaatest kus on võimalik teostada erinevaid tegevusi seoses platvormi konfigureerimisega. Leidub asju mida näiteks kasutajaliidese kaudu pole võimalik teha. Näiteks osade integratsioonide või seadmete ja teenuste, mille tugi platvormil puudub, üles seadmine. Selle jaoks on vaja seadistada platvorm manuaalselt, muutes selleks platvormi konfiguratsiooni faili nimega “configuration.yaml” [8]. Konfiguratsiooni fail on kirjutatud YAML (lühend *YAML Ain’t Markup Language*) faili formaadis, millega on mugav ka keerulisemaid konfiguratsioone platvormile kirjutada [9].

```
sensor:
  - platform: mqtt
    name: "Outside temperature"
    state_topic: "home-assistant/koodi/outside_temp"
    unit_of_measurement: "°C"
  - platform: mqtt
    name: "Kitchen temperature"
    state_topic: "home-assistant/koodi/kitchen_temp"
    unit_of_measurement: "°C"
  - platform: mqtt
    name: "Living room temperature"
    state_topic: "home-assistant/koodi/livingroom_temp"
    unit_of_measurement: "°C"
  - platform: mqtt
    name: "Outside light density"
    state_topic: "home-assistant/koodi/outside_light_tens"
    unit_of_measurement: "lm"
  - platform: nordpool
    region: "EE"
    precision: 3
    currency: "EUR"
    price_type: MWh
    VAT: False
```

Joonis 3. “configuration.yaml” sisend

Joonisel 3 on üks osa “configuration.yaml” failist ning sellel on kujutatud seadmed ja teenused mis on integreeritud platvormile manuaalselt. Tegemist on sensori domeeni kuulvate

seadmetega millest kolm mõõdavad temperatuuri, üks valgust ja üks jälgib Nord pooli elektrihinda. MQTT teenus on eelnevalt platvormis integreeritud, kuid tegemist on ühte tüüpi integratsiooniga mille üles seadmine ja seadistamine toimub manuaalselt läbi konfiguratsiooni faili. Nord pooli sensor on komponent, mille tugi hetkel platvormil puudub ning mis on platvormile integreeritud manuaalselt. HA dokumentatsioon [8] kirjutab, et konfiguratsiooni failiga on võimalik seadistada seadmete ja teenuste olemeid, seadistada platvormi väliste rakenduste paroole, malle andmete töötluseks ja sõnumite loomiseks jms. Näiteks olemile on võimalik seadistada ajaline intervall, millal platvorm uuendab olemi olekut või mingi kindel sõne mis olemi nimedele juurde liidetakse.

2.2.1.2 Andmemudel

HA dokumentatsioon [4] kirjeldab, et seadmeid ja teenuseid platvormil kutsutakse integratsioonideks. Üheks integratsiooniks võib näiteks olla Philips Hue valgusti. Igal integratsioonil on omad andmepunktid millega on integratsiooni võimalik kontrollida. Neid kutsutakse platvormil *entity*'deks ehk olemiteks. Näiteks olemiks on temperatuuri sensor või lugemislamp. Kõik olemid jaotatakse platvormi poolt domeenidesse. HA [6] kirjeldab domeeni kui iga integratsiooni unikaalset identifikaatorit milleks võib olla näiteks *light*, *sensor*, *switch* jne. Näiteks tule domeeniks on *light* ja temperatuuri mõõtva seadme domeeniks on *sensor*. Platvormi [5] poolt pakutavad domeenid on esitatud lisana (vt. Lisa I).

```
light.living_room_light
sensor.living_room_temperature
switch.living_room_window
```

Joonis 4. HA olemi nimi

Joonisel 4 on kujutatud kolme olemit. Esimeseks olemiks on tule domeeni kuuluv elutoa lamp. Nagu on näha, siis olemi nimes on seadme unikaalseks identifikaatoriks *light* ehk tule domeen. Teiseks on olem mis mõõdab elutoa temperatuuri. Tema unikaalseks identifikaatoriks on *sensor* ehk anduri domeen. Kolmandaks on olem millega on võimalik avada ja kinni panna elutoa aknaid. Tema unikaalseks identifikaatoriks on *switch* ehk lüliti domeen. Leidub juhte kus integratsioonid kasutavad teenuse edasi andmiseks nende endi poolt pakutavat domeeni, et lisada rohkem võimalusi seadmete või teenuste kontrollimiseks. Näiteks Philips Hue pakub lisaks ka nende poolt loodud domeeni, et oleks võimalik kasutada nende poolt pakutavat funktsionaalsust. Lõputöös valminud rakendus kasutab olemite tuvastamiseks olemite nime ning tema atribuute.

Joonis 5. Olemi andmed

Joonisel 5 on kujutis tule domeeni kuulvast olemist, tema olekust ning olemi andmetest mida platvorm meeles hoiab.

Järgnev lõik tugineb Home Assistant *Group* dokumentatsioonile [7] milles kirjeldatakse *Group* integratsiooni, tema käitumist ja kasutamist. Platvormil olevaid olemeid on võimalik grupeerida, kasutades selleks *Group* integratsiooni. *Group* dokumentatsioon kirjutab, et kasutades gruppe on võimalik olemeid kontrollida ning jälgida nende olekuid tervikuna. Gruppidel on neli olekut:

- *On*
- *Off*
- *Unavailable*
- *Unknown*

Gruppe defineerides on kaks võimalust kuidas grupp käituma panna. Esimene võimalus on seadistada grupis olek "*All entities*" valik tõeseks. Seda tüüpi grupis on grupi olek *off*, kui grupis leidub üks olem mille olek on *off*. Kui kõikide seadmete olek on *unavailable*, siis on grupi olekuks *unavailable*. Kui leidub vähemalt üks seade, mis on *unknown* või *unavailable*, siis grupi olek on *unknown.* Kõikidel teistel juhtudel on grupi olekuks *on*. Grupis, kus "*All entities*" on seadistatud väära tõeväärtusega, on grupi olekuks *on* kui leidub vähemalt üks olem mille olekuks on *on*. Grupi olek *unavailable* töötab samamoodi nagu eelmine grupp. Grupi olek *unknown* on siis, kui kõikide olemite väärtusteks on *unknown*. Gruppe on võimalik üles seada käsitsi konfiguratsiooni failis või kasutades selleks kasutajaliidest. *Group* dokumentatsioonis [7] on kirjutatud, et gruppe on võimalik üles seada kasutades kahte versiooni. Uuem versioon

kasutab olemi domeeni ning seob seadmed domeeni siseselt. Vanem versioon kasutab *group* domeeni, kuhu on võimalik lisada erinevatesse domeenidesse kuuluvaid olemeid.

```
light:
  - platform: mqtt
    name: "Kitchen light 1"
    command_topic: "home-assistant/koodi/kitchen_light_1"
  - platform: mqtt
    name: "Kitchen light 2"
    command_topic: "home-assistant/koodi/kitchen_light_2"
  - platform: mqtt
    name: "Living room light"
    command_topic: "home-assistant/koodi/livingroom_light"
  - platform: group
    name: "Kitchen light group"
    entities:
      - light.kitchen_light_1
      - light.kitchen_light_2
    all: true

group:
  living_room:
    name: "Living room"
    entities:
      - light.living_room_light
      - sensor.living_room_temperature
      - switch.living_room_window
```

Joonis 6. Olemite gruppeerimine

Joonisel 6 on kujutatud kaks viisi kuidas gruppe defineerida. Esimene grupp on *light* domeeni all mis seob kokku kõik köögi tuled. Esimesel grupil on seadistatud "*All entities*" väärtus tõeskes. Teine grupp on *group* domeeni all olev "living_room", mis seob kokku kõik seadmed mis on elutoas. Nendeks on elutoa tuli, temperatuuri sensor ning akent lahti ja kinni panev lüliti. Käesoleva lõputöö jooksul valminud rakendus kasutab vanemat gruppide versiooni, sest erinevaid tüüpi seadmeid on parem siduda kui nad on kõik ühe olemi nime all.

2.2.1.3 Rakendusliides

Home Assistant pakub erinevaid rakendusliideseid. Nendeks on REST , WebSocket ja Supervisor rakendusliides. Lõputöö jooksul valminud rakendus kasutab REST rakendusliidest automatsioonide tulemuste saatmiseks. Näiteks on automatsioon mis kustutab tule kui kell on 12 õhtul. Automatsioon salvestab tule oleku ning kasutades salvestatud olekut, saadetakse läbi REST rakendusliidese platvormile soovitud tule olek. Lisaks kasutab rakendus platvormi kõikide seadme andmete sisse lugemiseks WebSocket rakendusliidest.

REST rakendusliidese dokumentatsioon [10] kirjeldab, et HA platvormil on võimalik kasutada rakendusliidest kasutades selleks platvormi kasutajaliidese aadressi ja lisades lõppu "/api". Näiteks oleks õige rakendusliidese aadress: "homeassistant.local:8123/api/". Rakendusliidesega on võimalik suhelda kasutades selleks objekte mille andmete formaadiks on JSON. Rakenduseliidesele päringuid tehes peab päringu päisesse kindlasti kaasa andma *Authorization token* ehk turvavõtme. Seda nõuab platvorm sellepärast, et tundmatud isikud ei saaks platvormi rakendusliidest kuritarvitada. REST rakendusliidese dokumentatsioonis [10] on kirjas kahte tüüpi päringuid: *GET* päringud ja *POST* päringud. *GET* päringut kasutades on võimalik saada platvormilt kätte soovituid andmeid. Näiteks üheks *GET* päringuks on "homeassistant.local:8123/api/states", mis tagastab meile kõikide olemite olekud. *POST* päringut kasutades on võimalik saata platvormile andmeid. Näiteks üheks *POST* päringuks on "homeassistant.local:8123/api/services/light/turn_on" kuhu päringu päisesse on kaasa antud tule olem. Saates sellise päringu, platvorm võtab tule olemi nime ning paneb tule põlema. Näites toodud *POST* päringut kasutab valminud rakendus automatsiooni tulemuste saatmiseks HA platvormile.

WebSocket rakendusliidese dokumentatsioon [11] kirjutab, et WebSocketit on mõeldud andmete voogedastuseks. Voogedastuse all mõeldakse seda, et HA ja rakendus, mis kasutab WebSocketit, loovad silla oma vahel ning hakkavad suhtlema. Lõputöös valminud rakendus kasutab WebSocketit kahel põhjusel. Esiteks on WebSocketi abil rakendusel võimalik kätte saada kõikide seadmete andmed ja olekud. Teine põhjus on see, et kui platvormile integreeritud seadme olek muutub, siis on vajalik, et seadme oleku muutus kajastuks ka rakenduses reaalajas. HA ja rakenduse oma vahelise suhtlemise jaoks kasutatakse JSON formaadis objekte. Iga kord kui luuakse platvormi ja rakenduse vahel ühendus, peab rakendus kasutades turvavõtit ennast platvormile ära tuvastama. Kui tuvastamise staadium on läbi, siis rakendus saab saata platvormile erinevaid käske. Näiteks üheks käsuks on küsida andmeid platvormil olevate seadmete kohta.

Joonis 7. WebSocket vastus saadetud käsule

Joonisel 7 on kujutatud suhtlust rakenduse ja HA vahel. Esimene rida näitab seda, et HA nõuab autentimist enne kui rakendus saab hakata käske saatma. Teine rida annab teada, et autentimine on õnnestunud. Kolmas rida on vastus käsule, milleks oli saada platvormil olevate olemite andmed. Lisaks kasutab lõputöö jooksul valminud rakendus käsku “subscribe_events”. Selle käsuga hakkab rakendus kuulama HA toimuvaid muutusi. Kui olemil muutub olek, siis rakendus näeb seda ning kirjutab rakenduse siseselt eelnevalt sisse loetud olemi oleku üle.

2.2.2 Automatiseerimine

Edasi tutvustatakse Home Assistant automatiseerimist ning kuidas ta toimib. Cambridge inglise keele sõnastikus defineeritakse automatiseerimine järgmiselt: “Automatiseerimiseks nimetatakse tegevust, mille käigus pannakse tehases või kontoris toimuv protsess, mida eelnevalt tegid inimesed, töötama masinate või arvutite abil – eesmärgiga muuta töö tegemist efektiivsemaks või vähendada inimeste töökoormust” [4].

2.2.2.1 Home Assistant automatiseerimise loogika

Kõikide automaatikate põhimõte on järgnev – luuakse mingi kindel sündmus mille täitumist automaatika ootab ning kui sündmus läheb täide siis automaatika teeb vastavalt seadistatud toimingud [12].

Alampeatüki 3.1 alguses sai mainitud, et HA platvormil on ühilduvus üle 1800 erineva teenuse või seadme. Kui platvormile lisatakse seade/teenus, siis andmeid ja informatsiooni kasutades

[4] Automatiseerimise definitsioon: https://dictionary.cambridge.org/dictionary/english/automate

on võimalik luua erinevaid sündmuseid mille puhul automaatika teeb midagi [8]. Näiteks platvormil on seadistatud andur, mis jälgib kas väljas on päev või õhtu ning õhtu saabudes automaatika lülitab sisse tuled.

HA dokumentatsioonis [13] on automaatika reegel jaotatud kolmeks osaks:

- Esimene osa on sündmus, mis kirjeldab sündmust mida automaatika kuulama peab ning mille peale automaatika peab midagi tegema. Näiteks oleks sündmuseks see, kui tehakse lahti välisuks.
- Teine osa on tingimus, mis aitab automaatika reeglit täpsustada, et millised peavad olema sündmuse toimumise ajal tingimused, et automaatika käivitada. Eelnevalt toodud näidet saab täiustada tingimusega – kui välisuks tehakse lahti ning väljas on pime.
- Viimaseks automaatika reegli osaks on tegevus, mida saab kirjeldada kui automaatika tegevust kui mingi sündmus on juhtunud. Täiustades viimast näidet tegevusega saab automaatika reegli – kui välisuks tehakse lahti ning väljas on pime, siis pane esiku ja elutoa tuli põlema.

HA automatiseerimise süsteem toimib põhimõttel, kus kasutajad annavad ette reeglid, mille puhul automaatika käivitatakse ning konkreetsed tegevused, mida automaatika täitma peab.

2.2.2.2 Seadme või teenuse automatiseerimine

Edasi tutvustatakse lähemalt HA platvormil seadme/teenuse automatiseerimist. HA dokumentatsioon [15] kirjutab Automaatikaid on võimalik ehitada kasutades nii HA platvormi kasutajaliidest kui arvutis olevas programmeerimiskeskkonnad. Automaatikad platvormil arendatakse ning hoiustatakse YAML faili formaadis. Joonisel 8 on toodud näide milline näeb välja automaatika reegel YAML formaadis.

```yaml
automation näide:
  - alias: "Kui välisuks tehakse lahti ning toas on pime, siis pane elutoa ja esiku tuli põlema"
    trigger:
      - entity_id: binary_sensor.front_door
        platform: state
        to: 'on'
        for:
          seconds: 2
    condition:
      - condition: numeric_state
        entity_id: sensor.outside_light_density
        below: 120
    action:
      - service: light.turn_on
        target:
          entity_id:
            - light.living_room_light
            - light.lobby_light
```

Joonis 8. Home Assistant automatsiooni näide

Joonisel 8 on kujutatud automatsioon mis on ehitatud peatükk 2.2.2.1 näite põhjal. Siin on näha, et automatsiooni *trigger* on binaarsensor mis käivitab automatsiooni juhul kui seadme olekuks on "*on*" ning seade on samas olekus vähemalt 2 sekundit. Automatsiooni käivitamisel kontrollib automatsioon, kas väljas on pime. Kui väljas on pime, siis pannakse põlema esiku ja elutoa tuli põlema.

2.2.2.3 Automatiseerimis süsteemi nõrkused

Edasi analüüsitakse automatiseerimis süsteemi nõrkuseid. Platvormil implementeeritud automaatika ei toeta eesmärgipõhiseid automatsioone. Dirk E. Mahling [14] jt kirjutavad, et eesmärgipõhise töövoo idee seisneb selles, et iga eesmärgi jaoks on eelnevalt ette seatud vajalikud tingimused ja sammud, et eesmärki täita. Näiteks pole HA võimeline ehitama järgmist automaatikat: "Kui elutoas telekas töötab, siis vii elutoa valgustase madalaks". Kui selle automaatika tingimus on täidetud, siis automaatika peaks elutoas tulesid järjest kustu panema. Iga kord kui tuli kustu pannakse, siis kontrollitakse kas valgustase on jõudnud ette seatud eesmärgini. Seda ringi tehakse nii kaua kuni soovitud valgustus on toas saavutatud. Joonisel 9 kujutatud tingimusest on näha, et kui automaatika on ainult väga spetsiifiliseks otstarbeks, siis automaatika muutub väga kohmakaks ning selle loetavus on pigem halb.

```yaml
# Example of entry in configuration.yaml
automation my_lights:
  # Turns on lights 1 hour before sunset if people are home
  # and if people get home between 16:00-23:00
  - alias: "Rule 1 Light on in the evening"
    trigger:
      # Prefix the first line of each trigger configuration
      # with a '-' to enter multiple
      - platform: sun
        event: sunset
        offset: "-01:00:00"
      - platform: state
        entity_id: all
        to: "home"
    condition:
      # Prefix the first line of each condition configuration
      # with a '-'' to enter multiple
      - condition: state
        entity_id: all
        state: "home"
      - condition: time
        after: "16:00:00"
        before: "23:00:00"
    action:
      # With a single service call, we don't need a '-' before service - though you can if
you want to
      - service: homeassistant.turn_on
        target:
          entity_id: group.living_room
```

Joonis 9. HA automaatika reegel [16]

Joonisel 9 on kujutatud automatsiooni reegli tööks on käivitada tuled, kui päike on loojumas, kõik inimesed on kodus ja kella aeg jääb kella 4 ja 11 vahele.

2.3 Loogiline programmeerimine

Käesolevas peatükis räägitakse loogilisest programmeerimisest ning tutvustatakse loogilise programmeerimise keelt Prolog. Genesereth Michael [19] jt. kirjeldavad loogilist programmeerimist kui ühte programmeerimise stiili, mille kirjutamisel kasutatakse lausearvutust ja tema tehteid. Loogiline programmeerimine on üks deklaratiivse programmeerimise paradigmadest mille vastandiks on imperatiivne programmeerimine mille stiili järgivad arenduses enimkasutatud keeled(näiteks C++, Java, Python). Imperatiivse programmeerimise stiiliga ehitatud programmid on üles ehitatud nii, et kogu programmi töövoog on programmi arendaja enda poolt välja mõeldud (muutujad, kui-siis laused, tsüklid jne.). Loogilise programmeerimise puhul kirjeldab autor süsteemile andmed, relatsioonid ja eesmärgid. Programm see järel üritab leida viisi kuidas eesmärki saavutada kasutades selleks andmeid ja nende vahelisi relatsioone. Genesereth Michael [19] jt. toovad näite roboti põhjal, kelle ülesandeks on liikuda punktist A punkti B. Imperatiivse stiiliga programmi puhul kirjeldatakse kindel distants kui kaugele näiteks robot peab edasi liikuma, millal ta peab pöörama jne. Deklaratiivse stiiliga programmi puhul kirjeldatakse robotile tema ümbrus, antakse talle algus ja lõpp punkt ning lastakse robotil endal otsustada kuidas ta peaks käituma. Deklaratiivne programm üritab leida viisi kuidas täita programmi eesmärki, võrreldes imperatiivse programmiga mis küsib, et kuidas eesmärki täita.

2.3.1 Prolog

Prolog on käesoleva lõputöös üks kõige olulisemaid osasid sellepärast, et see on programmeerimiskeel millega hakatakse kirjeldama IoT seadmete automatsioone. Bramer Max [27] on avaldanud raamatu milles õpetatakse Prologi. Ta ütleb, et Prolog on üks kõige laialdasemalt kasutatavaid loogilise programmeerimise keel mille esimene implementatsioon loodi 1970-ndate aastate alguses. Kõik Prologi programmid koosnevad kahest komponendi: reeglitest ja faktidest. Faktiks nimetatakse väidet, mis on tõene.

```
temperatuur_olek(elutuba, 25).
temperatuur_olek(magamistuba, 20).
aken_olek(elutuba, lahti).
aken_olek(magamistuba, kinni).
```

Joonis 10. Prolog faktid

Joonisel 10 on toodud näide Prologis kirjutatud faktidest. Näiteks esimest fakti võib tõlkida järgmiselt: “Elutoa temperatuur on 25 kraadi.”. Sarnaselt kolmanda fakti puhul on tõlkeks: “Elutoa on lahti.”. Prologi teiseks komponendiks on reegel mis on kogum faktidest mida soovitakse omavahel seostada. Näiteks meil on reegel mis ütleb, et toas on palav kui aken on kinni ja temperatuur on 20 kraadi või rohkem.(vt Joonis 11).

```
toas_palav(X) :-
    aken_olek(X, kinni),
    temperatuur_olek(X, Y),
    Y >= 20.
```

Joonis 11. Prolog reegel

Faktid mis on programmis ära defineeritud kutsutakse *knowledgebase*’ks. Igat reeglit ja fakti on võimalik pärida Prologis.

```
?- toas_palav(X).
X = magamistuba.
```

Joonis 12. Prolog päring

Joonisel 12 on kujutatud Prologis tehtud päring kasutades joonisel 11 kujutatud Prolog reeglit ning joonisel 10 kujutatud fakte. Tehes päringu reegli pihta ning andes talle väärtuseks X, Prolog hakkab otsima faktide seast väärtusi, millega on võimalik täita reegli eesmärk. Bramer Max [27] kirjeldab sellist tegevust Prologis unifitseerimiseks. Unifitseerimine on tegevus mis üritab leida kahe väärtuse vahel ühist vormi. Näiteks üritatakse leida väärtusele “aken_olek(X, kinni)” väärtust X. Selle leidmiseks Prolog teeb Prolog fakti “aken_olek(elutuba, lahti)” ning võrdleb seda. Unifitseerimine ebaõnnestus, sest “kinni” ja “lahti” pole võimalik ühtsesse vormi viia. Järgmisena võtab Prolog fakti “aken_olek(magamistuba, kinni)”. Unifitseerimine õnnestus, sest väärtust “kinni” ja “kinni” on võimalik ühtsesse vormi viia. Kuna unifitseerimise õnnestus, siis on võimalik väärtus X siduda väärtusega “magamistuba”. Sedasi edasi minnes üritab Prolog päringu eesmärki täita kasutades selleks unifitseerimist.

Järgnev lõik tugineb Bramer Max õpikule [27] kus kirjeldatakse Prologi väärtusi. Prologis kirjutatakse faktide väärtusi kas numbriliselt, aatomitena või sõnedena. Prologis on aatomeid kolme tüüpi. Üks variant aatomeid defineerida on väikese algustähega ning mille pikkus on vähemalt üks või rohkem tähemärki. Teine variant on defineerida kasutades ülakomasid ning kolmas variant on jada erilistest sümbolitest mille pikkuseks on vähemalt üks sümbol. Nendeks

sümboliteks on: “+ - * / > < = & # @ ”. Sõnesid on võimalik defineerida kasutades jutumärke. Prologis on võimalik muutujaid kasutada kui tehakse päring kas reegli või fakti pihta. Kõik muutujad Prologis algavad suure algustähega ning nende pikkus peab olema üks või rohkem tähemärki. Muutujad nimetatakse Prologis terminiteks, mille väärtust alles otsitakse. Kui see väärtus leitakse, siis pakutakse seda lahendina.

2.4 IoT süsteemid ja loogiline programmeermine

Käesolev peatükk räägib IoT süsteemidest (näiteks targa kodu süsteem), mis kasutavad loogilist programmeerimist seadmete automatiseerimiseks. Lisaks uuritakse mille poolest erinevad ja sarnanevad kirjutatud süsteemid lõputöös valminud rakendusega.

Bisicchia Giuseppe jt [20] avaldasid artikli, kus nad kirjeldasid nutikate seadmete automatiseerimise raamistiku, mida on võimalik kasutada nutikate seadmete poolt mõjutatavates keskkondades. Nende artikkel kirjeldab peamiselt kahte asja:

1. Deklaratiivset raamistikku, mis võimaldab luua leppe poliitika konfiguratsioone, et lahendada kasutajate poolt loodud automatsioonide konflikte.
2. Prototüüpi nimega Solomon, mis on REST teenus ning mis kasutab eelmises punktis kirjeldatud raamistiku. Raamistiku implementeerimiseks on kasutatud Prologi.

Solomon, mida Bisicchia G. jt [20] oma artiklis kirjeldavad, on tööriist nutikate seadmete automatiseerimiseks, mis on vahelüliks nutikate seadmete keskkonnal ning tavakasutajate/süsteemi administraatorite vahel. Solomoni kasutamiseks on vajalik eelnevalt ära kirjeldada keskkond, mida soovitakse automatiseerida kasutades selleks Prologis kirjeldatud reegleid ja fakte. Keskkonnas olevad seadmed esialgu kirjeldatakse faktidena. Hiljem grupeeritakse seadmed kasutades tsoon fakte (näiteks köök või elutuba) ning seadmete isendi klasse. Isendi klass on fakt, mis aitab siduda rakendusel tsoonid, seadme domeeni, seadmed ning mida seade keskkonnas mõjutab (näiteks valgust või temperatuuri). Kui keskkond on üles seatud, siis süsteemi administraatoritel on võimalik luua konfiguratsioone reeglite konfliktide lahendamiseks. Rakendus kasutab neid konfiguratsioone, et leida konflikti lahend ning siis teeb keskkonnas vastavad muudatused. Tavakasutajatel on võimalik Solomonis seada IoT seadmete eesmärgid, mida seadmed täitma peavad. Autorid tõid enda töös näite targa kodu lahendusest, kus on mitmel kasutajal võimalik seadistada endale sobilik

toatemperatuur. Kui kasutajate poolt seadistatud toa temperatuurid erinevad, siis raamistik leiab eelnevalt seadistatud vahenduspoliitika abil kompromissi.

Järgnev lõik tugineb D'Urso Fabio jt. artiklile [21] kus kirjeldatakse nutikate seadmete keskkonda, mida on võimalik muuta intelligentseks kasutades deklaratiivset Pythoni tööriista. Autorid kasutavad artiklis nende poolt ehitatud tööriista nimega PHIDAS. PHIDAS on raamistik mis võimaldab kasutajal üles seada Prologi sarnast faktide kogumit seadmetest, protseduurilisi ja reageerivaid reegleid. Reageerivaks reegliks nimetatakse reeglit, mis mingi kindla sündmuse korral käivitatakse. Näiteks süsteemil on reageeriv reegel mis käivitatakse siis, kui mingi sensori väärtus muutub. Kui reegel käivitub, siis ta võtab uue sensori väärtuse ja asendab selle vanaga faktide kogumis. Protseduuriliseks reegliks nimetatakse reeglit, mis käivitatakse ainult siis, kui nende kohta spetsiifiliselt päritakse. Näiteks on automaatika, kus vaadatakse kas toas on pime. Kui toas on pime, siis kutsume välja reegli, mis paneks tule põlema. Antud raamistik võimaldab kasutajal luua süsteem, mida on võimalik manipuleerida loogiliste/deklaratiivsete konstruktoritega. PHIDAS-t kasutades, D'Urso Fabio [21] jt. implementeerisid intelligentse IoT keskkonna, kus kõik seadmed jooksevad mikrokontrollerite peal ning millega saab luua erinevaid automatsioone kasutades loogilisi konstruktoreid. Seadmed kuulavad üksteist kasutades selleks sõnumi lüüsi (ingl. k *message gateway*), mis kasutab Wifi moodulit soklina (ingl.k *socket*), et seadmed saaksid üksteisele saata ja kuulata sõnumeid.

Solomoni rakenduse automatiseerimise süsteem võrreldes valminud rakenduse süsteemiga järgivad sama loogikat. Seadmed esmalt kirjeldatakse faktidena, faktid grupeeritakse ning antakse tegevustele domeenid, et rakenduses saaks kindlaks teha mida soovitakse mõjutada. Erinevusi nende kahe rakenduse vahel siiski leidub. Näiteks pole mõeldud eesmärgipõhisele automatiseerimisele ning antud rakendus pole ühendatud targakodu platvormi külge. Suurim erinevus Solomoni ja valminud süsteemi vahel on see, et Solomonil on võimekus lahendada kasutaja poolt seatud reeglite konflikte, mida valminud rakendus ei suuda teha. PHIDAS on natukene teist sorti automatiseerimise süsteem, kuid idee on nii Solomoni kui ka valminud rakendusega võrreldes sama. Pythoni tööriist on võrreldes Prologiga keerulisem, kuid see eest see toimib põhimõtteliselt nutikodu platvormina mille seadmed on ühendatud mitmete mikrokontrollerite külge mis on oma vahel pandud suhtlema.

3. Valminud rakendus

Selles peatükis kirjeldatakse Home Assistant (HA) platvormile rakenduse arendamist. Peatüki jooksul kirjutatakse rakenduse funktsionaalsetest nõuetest, rakenduse arhitektuurist ning kuidas arhitektuur on rakendusena valmis ehitatud.

3.1 Funktsionaalsed nõuded

Käesolev peatükk räägib lähemalt valminud süsteemi funktsionaalsetest nõuetest. Rakenduse funktsonaalsed nõuded kujunesid küsimuste põhjal: kuidas rakendus HA-st seadme andmed kätte saab, kuidas seadmeid automatiseerida Prologis, kuidas luua eesmärgipõhiseid automaatikaid ja kuidas rakendus edastab automatsiooni tulemused HA-le. Tabelis 1 on kujutatud rakenduse funktsionaalseid nõudeid.

Tabel 1. Rakenduse funktsionaalsed nõuded

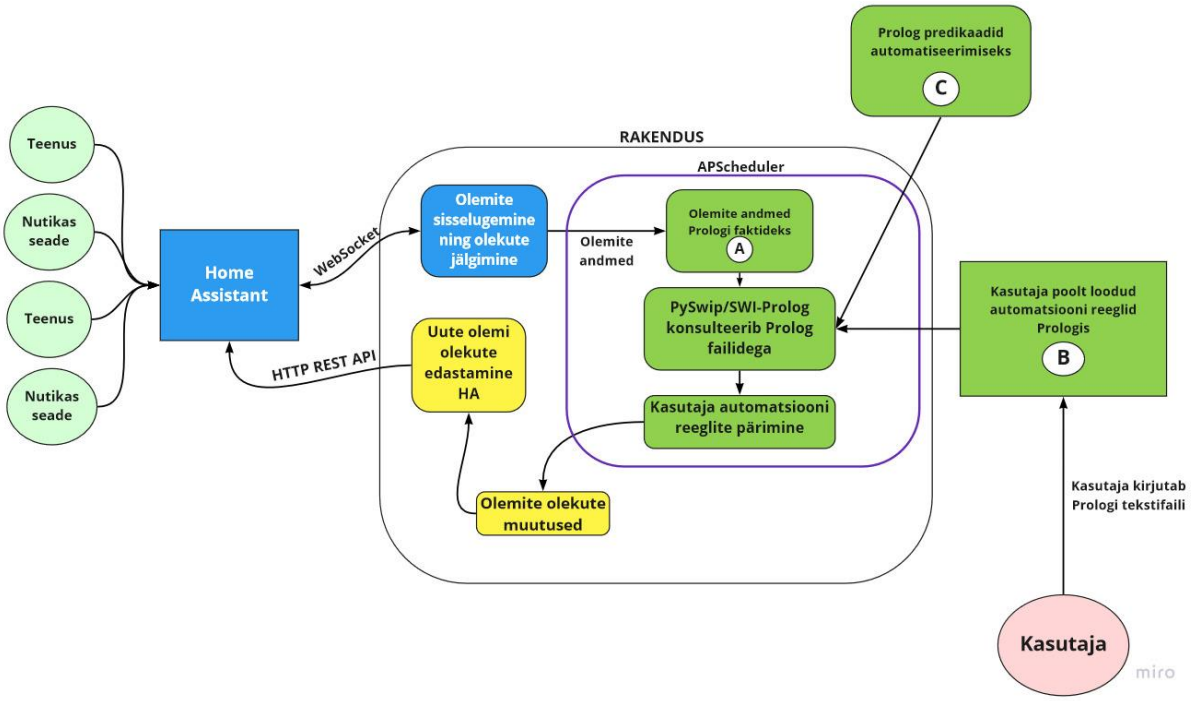
Nõude nr.	**Funktsionaalse nõude kirjeldus**
FN-1	Rakendus peab oskama sisselugeda HA platvormi seadmete ja teenuste olemid ning nende andmed ja uuendada olemite andmeid, kui platvormil toimub mingi muudatus.
FN-2	Rakendus peab olema võimeline looma Prologi faktid andmetest mis platvormilt saadud on.
FN-3	Kasutajal on võimalik luua automatsioone Prologi süntaksiga teksti failis.
FN-4	Rakendus peab pakkuma kasutajale Prolog reegleid, mille abil on võimalik salvestada aktuaatorite uus olek.
FN-5	Rakendus peab sisaldama erinevaid olemasolevaid Prolog predikaate, mis võimaldavad kasutajal mugavat eesmärgipõhist automatsioonide kirjeldamist
FN-5-1	Rakendus sisaldab Prolog reeglit, mis leiab seadmeid ja teenuseid grupi põhiselt
FN-5-2	Rakendus sisaldab Prolog reeglit, mis leiab seadmeid ja teenuseid domeeni põhiselt
FN-5-3	Rakendus sisaldab Prolog reeglit, mis leiab skoobist ühe aktuaatori.
FN-6	Rakendus peab pakkuma kasutajale olemasolevaid Prolog reegleid, millega on võimalik eesmärgipõhiselt seadme olekut muuta.
FN-7	Rakendus peab oskama pärida Prologis tehtud automatsioone ning saadab tulemused platvormile.
FN-8	Rakendus peab oskama pärida automatsioone iga teatud intervalli tagant.

Esimene funktsionaalne nõue on selleks, et rakendusel oleks ülevaade seadme ja teenuse olemitest ja nende olekutest mis on HA platvormilt kätte saadavad. Teine nõue on selleks, et seadmeid automatiseerides oleks vajalikud faktid seadmete olemite kohta olemas. Kolmas nõue on selleks, et kasutajatel oleks võimalik rakenduses seadmeid automatiseerida kasutades selleks Prologi. Neljas nõue on mõeldud automatsioonide poolt muudetud aktuaatorite olekute salvestamiseks. Selle abil on võimalik rakendusel muudetud aktuaatorite olekud edasi saata HA platvormile. Viies nõue koos oma alamnõuetega on mõeldud selle jaoks, et kasutajal oleks võimalik luua eesmärgipõhiseid automatsioone. Esimene ja teine alamnõue võimaldab teha nimekirja seadme ja teenuste olemitest. Näiteks platvormil on tehtud grupp elutoa seadmete olemitest. Grupipõhine otsimine võimaldab platvormi kõikide olemite seast leida gruppi kuuluvad olemid. Samamoodi toimib ka domeeni põhine olemite otsimine. Kolmas alamnõue võimaldab olemite nimekirjast leida ühe aktuaatori ette antud eesmärgiga. Kuues funktsionaalne nõue võimaldab rakenduses kasutajatel kasutada reegleid, millega on võimalik automatsioon muuta eesmärgipõhiseks. Näiteks kasutaja soovib elutoas valgustaset alla viia. Selleks, et sellist eesmärki täita otsitakse kõik elutoas olevad valgust mõjutavad seadmed mida hakatakse ükshaaval välja lülitama. Seadmeid lülitatakse nii kaua välja kuni elutoa valgustase on saavutanud soovitud taseme. Seitsmes nõue rakenduses võimaldab kasutaja poolt loodud automatsioonide reegleid pärida. Iga päringu tulemusel kontrollitakse kasutaja automatsioone ning vajaduse korral saadetakse automatsiooni tulemus platvormile. Rakenduse kaheksas nõue on kahel põhjusel. HA platvormil on seadmete ja teenuste olekud pidevas muutuses. Iga muutuse korral pole mõistlik automatsioone jooksutada, sest rakendus ei pruugi enda tööd lõpetada enne kui uue oleku muutus saabub. Teine põhjus on eesmärgipõhisuse saavutamine. Valminud rakendus salvestab Prologis päritud automatsioonide tulemused ning pärast Prologi jooksutamist saadetakse uued olemi olekud platvormile. Näiteks automatsioon, mille eesmärgiks on valgustaseme alandamine lülitades järjest tulesid välja. Valminud rakendus kontrollib, kas valgustase on seatud eesmärgi saavutanud. Kui eesmärk ei ole saavutatud, siis lülitatakse üks tuli välja. Iga iteratsiooniga mis rakendus automatsioonide pärimisel teeb, lülitatakse üks tuli välja kuni on saavutatud soovitud valgustase.

3.2 Arhitektuur

Antud peatükis kirjeldatakse ehitatud rakenduse arhitektuuri ning mida on kasutatud selle arhitektuuri implementeerimiseks. Rakenduse implementeerimiseks on kasutatud Pythonit ning selle teeke. Valminud rakendus on kätte saadav:

https://github.com/markkuskoodi/prologHA. Rakenduse arhitektuuri ja tema töövoogu on kujutatud joonisel 13.



Joonis 13. Rakenduse arhitektuur

Joonisel 13. on välja toodud helesinisega Home Assistant (HA) platvorm, mille külge on integreeritud nutikad seadmed ja teenused. Lisaks on helesinisega märgitud ka rakenduse üks osa, mille abil suhtleb rakendus HA-ga, et nutikodu seadmete ja teenuste olemite kohta andmeid koguda. Rohelisega märgitud kastid kujutavad rakendusse implementeeritud Prologi ja selle töövoogu. Kollasega on märgitud kastid näitavad kuidas rakendus pärast Prologi jooksutamist saadud automatsiooni tulemused HA-le edastab. Lisaks on joonisel märgitud ka kasutaja ning mida rakendus kasutaja poolt ootab. Järgnevates peatükkides kirjutatakse põhjalikumalt kuidas rakendus on implementeeritud ja kuidas ta käitub.

3.2.1 Rakenduse ja Home Assistant platvormi integreermine

Tabelis 1 kirjeldatud FN-1 implementeerimiseks kasutab rakendus WebSocket lahendust. Rakenduses on WebSocket implementeeritud websocket-client[5] teegi abil mis võimaldab luua silla mille kaudu suheldakse HA-ga. Rakendus kasutab HA WebSocket rakendusliideses kahte

[5] https://websocket-client.readthedocs.io/en/latest/

käsku: “get_status” ning “subscribe_events”. Käsk “get_status” tagastab kõikide integreeritud seadmete ja teenuste olekud.

```
{"type":"auth_required","ha_version":"2022.4.2"}
{"type":"auth_ok","ha_version":"2022.4.2"}
{
    "id": 2,
    "type": "result",
    "success": true,
    "result": [
        {
            "entity_id": "person.admin",
            "state": "unknown",
            "attributes": {
                "editable": false,
                "id": "admin",
                "user_id": "57e0e182bf95477fb19a9bf09c695a7a",
                "friendly_name": "admin"
            },
            "last_changed": "2022-05-04T06:50:55.401015+00:00",
            "last_updated": "2022-05-04T06:50:57.543512+00:00",
            "context": {
                "id": "6067e6163f6ff1747f83644c340d94fb",
                "parent_id": null,
                "user_id": null
            }
        },
```


Joonis 14. “get_status” käsu tagastus

Joonisel 14 on kujutatud WebSocket kasutajaliidese tagastus käsule “get_status”. Jooniselt on näha, et käsu tulemuseks on üks HA platvormile integreeritud teenuse olemitest. Kuna seadmete ja teenusete olekud muutuvad koguaeg, siis nende muutuste kuulamiseks me kasutame käsku “subscribe_events”. Kõik seadmete ja teenusete olemid, mis WebSocket vahendusel saadetakse kaardistatakse Pythoni *dictionary* (sõnastikku). Sõnastikus on võtmeks seadme identifikaator ning tema väärtusteks on seadme olek ning seadme atribuudid.

```
light.kitchen_light_1
[
    "off",
    {
        "supported_color_modes": [
            "onoff"
        ],
        "assumed_state": true,
        "friendly_name": "Kitchen light 1",
        "supported_features": 0
    }
]
```


Joonis 15. Olemitest koosnev sõnastiku üks väärtustest

Joonisel 15 on kujutatud rakenduses olev sõnastiku üks olemitest. Joonisel on näha, et olem nimega “light.kitchen_light_group” on sõnastikus võtmeks. Andes sõnastikule see võti tagastatakse joonisel kujutatud olemi olek ning olemi atribuudid. Kuna seadme ja teenuse olemeid on palju, siis käesoleva lõputöö raames ehitatud rakendus toetab järgmised HA domeene:

- *Light* domeeni kuuluvad seadmed.
- *Sensor* domeeni kuuluvad seadmed.
- *Climate* domeeni kuuluvad seadmed.
- *Switch* domeeni kuuluvad seadmed.
- *Person* domeeni kuuluvad seadmed (tegemist on HA platvormil *Person* integratsiooniga).
- *Group* domeeni kuuluvad seadmed (tegemist on HA platvormil *Group* integratsiooniga).

Kui seadmete ja teenuste olemid on rakenduses sõnastikku sisse loetud, siis hakkab rakendus kuulama HA platvormil toimuvaid muudatusi. Rakendus jälgib kas HA platvormi poolt saadud sõnum on seotud ühe olemiga sõnastikus. Kui tegemist on sõnastikus oleva olemiga, siis rakendus uuendab olemi väärtusi.

3.2.2 Seadmete ja teenuste automatiseerimine rakenduses

Enne kasutajate poolt tehtud automatsiooni reeglite pärimist käib rakendus üle eelmises peatükis kirjeldatud sõnastiku, mille abil tehakse seadme ja teenuse olemitest Prolog faktid (vt Joonis 13(A)). Rakenduse FN-2 täitmiseks kasutatakse eelmises peatükis kirjeldatud olemitest koosnevat sõnastikku. Sõnastik käiakse rakenduse poolt üle ning igast olemist Prolog faktid. Rakendus kontrollib millisesse domeeni kuulub olem ning loob vastavad faktid. Kui tegemist on aktuaatori tüüpi olemiga, siis luuakse kolm fakti ning kui ei ole siis luuakse kaks fakti. Näiteks joonisel 14 kujutatud olemi puhul teeb rakendus kolm fakti.

```
actuator("light.kitchen_light_1", off).
domain("light.kitchen_light_1", "light").
actuator_type("light.kitchen_light_1", light).
```

Joonis 16. Joonisel 15 kujutatud olemi faktid Prologis

Joonisel 16 on kujutatud joonisel 15 näidatud olemi faktid Prologis. Rakendus lõi kolm fakti selle olemi kohta. Kuna HA platvormil on olemi domeeniks (ehk olemi nimes enne punkti kirjutatud) *light*, siis tegemist on tulega/lambiga mis Prologis defineeritakse aktuaatoriks.

Esimene fakt on “actuator” ehk aktuaator, mis hoiab sõnena meeles olemi nime ning aatomina olemi olekut. Teine fakt on domeen mis kirjeldab ära millisesse domeeni kuulub joonisel kujutatud olem. Antud juhul kuulub olem “light” ehk valgus domeeni, sest tegemist on olemiga millega on võimalik valgust mõjutada. Kolmas fakt on “actuator_type” ehk aktuaatori tüüp, mis hoiab muutujateks olemi nime ja aktuaatori tüüpi ehk *light*.

```
light_sensor_status("sensor.outside_light_density", low).
domain("sensor.outside_light_density", "light").
```

Joonis 17. Valgust mõõtva sensori olem fakti kujul

Joonisel 17 on kujutatud valgust mõõtva sensori olem Prologis faktide kujul. Kuna tegemist on olemiga mille olekut pole võimalik platvormil kontrollida, siis rakendus loob talle ainult kaks fakti. Esimeseks faktiks on “light_sensor_status” ehk valgussensori olek kus tal on muutujate väärtuseks olemi nimi ning sensori väärtus ehk madal. Teiseks faktiks on domeen, kus on defineeritud, et sensor kuulub valgus domeeni. Seda sellepärast, et sensor mõõdab valgustaset. Kõikide sensor tüüpi olemitega rakendus kasutab numbriliste väärtuste asemel kolme klassi (vt. tabel 2).

Tabel 2. Sensorite numbriliste väärtuste klassid

Sensori tüüp	***low***	***medium***	***high***
Valgust mõõtev	Väärtus < 130	130 <= Väärtus <= 170	Väärtus > 170
Temperatuuri mõõtev	Väärtus < 10	10 <= Väärtus <= 22	Väärtus > 22

Nutikodu automatiseerimisel on väga tähtis ka gruppide olemasolu, sest selle abil on võimalik platvormil olemeid grupeerida. Näiteks võib gruppide loomise aluseks olla toad. Gruppide kasutamine rakenduses on võimalik siis kui kasutatakse HA gruppide vanemat implementatsiooni, sest see võimaldab meil gruppidesse panna eri tüüpi seadmeid. Rakendus kirjutab grupis olevad seadmed eraldi faktidena, et neid oleks hiljem võimalik automatsioonides pärida (vt Joonis 18).

```
group("light.table_lamp", "group.living_room").
group("light.sofa_lamp", "group.living_room").
group("sensor.motion_sensor_light_level", "group.living_room").
```

Joonis 18. Grupi faktid

Joonisel 15 on kujutatud rakenduse poolt tehtud faktid grupi olemist. Esimesel real on fakt mis defineerib, et olem “light.table_lamp” kuulub gruppi “group.living_room” ehk elutoa olemite gruppi. Samamoodi on joonisel defineeritud ka järgmised olemid mis kuuluvad elutoa gruppi. Kõik olemite faktid kirjutatakse rakenduse poolt faili “automation_final.pl”, mida hiljem rakenduses hakatakse Prologi poolt kasutama.

Rakenduse nõude FN-3 täitmiseks on rakendusele tehtud fail nimega “automation_rules.pl” (vt Joonis12(B)). Sinna saab rakenduse kasutaja kirjutada Prologis automatsiooni reegleid kasutades selleks kasutades selleks kas mingit programmeerimise keskkonda (näiteks PyCharm) või tekstiredaktorit (näiteks Notepad++).

```
automatsiooni_reegel :-
    light_sensor_status("sensor.outside_light_density", low),
    actuator_action("light.kitchen_light_1", on).
```

Joonis 19. Kasutaja poolt kirjutatud automatsiooni reegel

Joonisel 19 on kujutatud automatsiooni reegel mis on kirjutatud rakenduse kasutaja poolt faili nimega “automation_rules.pl”. Automatsiooni reegli esimene rida kontrollib, et kas sensor tüüpi olem “sensor.outside_light_density” tase on *low* ehk madal. Kui fakti kontroll tagastab tõese väärtuse, siis automatsiooni reegel kasutab abipredikaati mille abil salvestatakse aktuaatori uus olek tulemuste faili. Lõputöö jooksul valminud abipredikaatidest räägitakse lähemalt peatükis 3.2.4.

Kui rakendus on lõpetanud seadmetest faktide kirjutamise, siis rakendus loeb kasutaja poolt loodud automaatika päised endale mällu. Rakendus käib läbi faili kuhu rakenduse kasutaja automatsiooni reeglid lisas. Faili läbi lugedes jätab rakendus meelde kõik automatsiooni reeglite päised. Joonisel 16 oleva automatsiooni reegli päiseks on “automatsiooni_reegel”. Funktsionaalse nõude FN-6 esimese poole rahuldamiseks kasutab rakendus Prolog failide jooksutamiseks kasutab rakendus PySwip[6] teegis olevat Prolog klassi. See kasutab arvutis SWI-Prolog[7] rakendust, et teha automatsioonide pihta päringuid.

Prologi klass loeb sisse kolm Prolog faili. Esimesest failist loeb klass sisse kõik faktid mis rakendus eelnevalt tegi. Teisest failist loetakse kasutaja poolt loodud automaatika predikaadid.

[6] https://github.com/yuce/pyswip
[7] https://www.swi-prolog.org/

Kolmandast failist loetakse sisse automatiseerimiseks kasutatavad abipredikaadid. Pärast failide sisse lugemist teeb rakendus eelnevalt mällu salvestatud reegli päisete pihta päringu mille tulemusel käivitatakse kasutaja poolt loodud automatsiooni reeglid.

3.2.3 Automaatika tulemuste saatmine HA seadmetele/teenustele

Kõik aktuaatorite muudatused, mille Prologi automaatika reeglid salvestasid, on kirjutatud automatsioonide tulemuse faili. Enne PySwip kasutamist automatsiooni reeglite peal, loob rakendus automatsiooni tulemuste salvestamiseks fail mille nimeks on kuupäev ja kellaaeg (näiteks on faili nimeks "2022_05_04_03_04_29.txt"). Prologist saadud automatsioonide tulemused kirjutatakse faili järgmiselt: "seadme_id uus_olek". Näiteks joonisel 16 kujutatud reegli tulemus kirjutatakse faili „*light.kitchen_light_1 on*".

Rakendus nõude FN-6 teise poole rahuldamiseks loeb rakendus kõik olekute muutused failist ning saadab seadmete uued olekud üle HA REST rakendusliidese kasutades Requests[8] teeki. Rakendusega on võimalik hetkel saata uusi olekuid üle REST API *light*, *switch* ja *climate* tüüpi seadmetele.

Rakenduse nõude FN-8 täitmiseks kasutatakse APScheduleri[9] BackgroundScheduler klassi, millega rakendus käivitab Prolog automatiseerimise süsteem intervallidega. Rakenduses on kasutatud klassi funktsiooni "add_job", mille parameetriteks on antud rakenduse funktsioon mis jooksutab Prologi automatiseerimise süsteemi, automatiseerimise süsteemi funktsiooni parameetrid ning intervall kui tihti rakenduse automatiseerimise süsteemi välja kutsutakse(vt. Joonis 20).

```
scheduler = BackgroundScheduler()
scheduler.add_job(invoke_prolog, 'interval', args=[data, rest], seconds=5)
scheduler.start()
```

Joonis 20. BackgroundScheduler-i kasutamine rakenduses

Kasutades joonisel 20 olevat klassi on meil Prologi tööd võimalik itereerida. Itereerimine on tähtis sellepärast, et rakendus ei uuenda jooksvalt defineeritud olemite fakte. Selle tulemusel uuendatakse kõik olemite faktid ning automatsioonid kasutavad koguaeg õigeid andmeid. Lisaks kasutades itereerimist on võimalik meil Prologis saavutada eesmärgipõhine automatiseerimine. Näiteks meil on valgust mõõtev sensori olem, mille eesmärgiks on valguse

[8] https://docs.python-requests.org/en/latest/
[9] https://apscheduler.readthedocs.io/en/3.x/

tase alandamine. Automatsiooni reegel lülitab ühe tule välja, kuid kuna Prologi faktid pärast tule välja lülitamist aegunud, siis lõpetab reegel töö. Järgmine kord kui Prologi jooksutatakse on faktid ära uuendatud ja automatsioon jätkab oma ringiga. Eesmärgipõhise automatsiooni ring jätkub nii kaua kuni reeglis seatud eesmärk on täidetud.

3.2.4 Rakenduse automatiseerimise süsteemi abipredikaadid

Käesolev peatükk kirjutab lõputöö jooksul valminud Prologi predikaadid, mida rakenduse automatiseerimise süsteemis kasutama peab (vt joonis 12(C)). Peatüki esimeses osas kirjutatakse valminud predikaatidest millega on võimalik olemite olekuid Prologis muuta. Neid kasutades on võimalik ehitada HA platvormile sarnaseid süsteeme. Peatüki teises osas kirjutatakse predikaatide kohta tuuakse välja predikaadid millega on võimalik seadmeid ära gruppeerida Prologi siseselt ehk skoopidesse panna. Kolmandas osas näidatakse predikaate mida kasutades on võimalik luua eesmärgipõhiseid automatsiooni reegleid. Kolmandas osas seletatud predikaadid on mõeldud valgus domeenis kuuluvate olemite jaoks.

Rakenduse nõude FN-4 rahuldamiseks loodi Prologis abipredikaat nimega *actuator_action*. *Actuator_action* predikaati on automatiseerimise süsteemis kahte vormi. Esimene predikaadi vorm on mõeldud *light* ja *switch* domeeni kuuluvate olemite jaoks ning teine on mõeldud *climate* domeeni olemite jaoks. Esimene predikaat loeb sisse *light* ja *switch* olemite korral kaks väärtust. Nendeks on olemi nimi ja uus olemi oleku väärtus. Teine predikaat loeb sisse *climate* kolm väärtust, sest *climate* domeeni kuuluvatel olemitel võib olla erinevaid mooduleid mida on võimalik kontrollida. Nendeks väärtusteks on olemi nimi, olemi oleku väärtus ning moodul mille olekut muudetakse. Abipredikaat kasutab SWI-Prolog predikaati *string_concat*[10], mis liidab predikaadile antud väärtused üheks sõneks. Kui sõne on valmis tehtud funktsiooni poolt, siis saadetakse sõne edasi abipredikaati, mis kirjutab sõne automatsiooni tulemuste faili.

[10] https://www.swi-prolog.org/pldoc/man?predicate=string_concat/3

```
actuator_action(Entity_id, Action):-
    string_concat(Entity_id, " ", TempResp),
    string_concat(TempResp, Action, TempResp_2),
    string_concat(TempResp_2, "\n", Response),
    write_output(Response).

actuator_action(Entity_id, Action, Mode):-
    string_concat(Entity_id, " ", TempResp),
    string_concat(TempResp, Action, TempResp_2),
    string_concat(TempResp_2, " ", TempResp_3),
    string_concat(TempResp_3, Mode, TempResp_4),
    string_concat(TempResp_4, "\n", Response),
    write_output(Response).
```

Joonis 21. Abipredikaat "actuator_action"

Joonisel 21 on kujutatud eelmises lõigus kirjeldatud abipredikaat. Joonisel olev esimene variant on mõeldud *light* ja *switch* domeeni kuuluvate olemite jaoks kuna nende domeenis olevatel olemitel pole erinevaid mooduleid mida ühes olemis mõjutada. Joonisel teine variant on mõeldud *climate* domeeni kuuluvate olemite jaoks kuna nende domeenis olevatel olemitel on erinevaid mooduleid mille olekut on võimalik muuta. Näiteks on kliimasüsteemil on võimalik seadistada ventilaatori kiirust, kas süsteem hetkel soojendab tuba või jahutab jne.

```
actuator_action("light.kitchen_light_1", on).
actuator_action("climate.air_conditioner", heat, mode).
```

Joonis 22. "actuator_actioni" kasutamine light ja climate olemite peal

Joonis 22 on kujutatud kaks võimalust kuidas abipredikaate kasutada. Esimesel juhul on predikaati kasutatud *light* domeeni kuuluva olemi oleku muutmiseks. Teisel juhul on kasutatud *climate* olemi oleku muutmiseks.

```
light.kitchen_light_1 on
climate.air_conditioner heat mode
```

Joonis 23. Predikaadi poolt valminud sõne tulemuste failis

Joonis 23 on kujutatud *actuator_action* predikaatide poolt koostatud sõned automatsiooni tulemuste failis. Neid ridu loeb rakendus ükshaaval sisse ning saadab läbi REST rakendusliidese HA.

Rakenduse nõude FN-5 ja tema alamnõuete rahuldamiseks on loodud kolme tüüpi predikaate. Nõude FN-5-1 rahuldamiseks on loodud kahte tüüpi abipredikaate nimega *group_scope*.

```
group_scope(Group, Result):-
    findall(Entity_id, group(Entity_id, Group), Result).

group_scope(Group, Scope, Result):-
    findall(Entity_id, group(Group_id, Group), GroupScope),
    intersection(Scope, GroupScope, Result).

domain_scope(Domain, Result):-
    findall(Entity_id, domain(Entity_id, Domain), Result).

domain_scope(Domain, Scope, Result):-
    findall(Entity_id, domain(Entity_id, Domain), DomainScope),
    intersection(Scope, DomainScope, Result).

find_actuator_by_goal(Entity_id, Goal, Scope) :-
    actuator(Entity_id, Entity_status),
    actuator_type(Entity_id, Entity_type),
    action(Entity_type, Goal, Entity_status),
    member(Entity_id, Scope).
```

Joonis 24. "group_scope" abipredikaadid

Joonisel 24 on kujutatud eelmises lauses kirjeldatud grupi abipredikaat. Joonisel kujutatud esimene *group_scope* abipredikaat kasutab väärtusteks grupi olemi nime ning muutujat nimega "Result". Antud abipredikaadi ülesandeks on üles leida talle etteantud grupi olemi järgi gruppi kuuluvad olemid. Predikaat kasutab selleks SWI-Prolog predikaati *findall*[11] (tõlkes "leia kõik"), millele esimeseks väärtuseks on muutuja mida me soovime salvestada. Teiseks väärtuseks on päring mis unifitseerimise abil leiab olemi nime ning mis seotakse esimese muutujaga. Kolmandaks väärtuseks järjend kuhu leitud olemid lisatakse ning mis seotakse predikaadi päises oleva "Result" muutujaga. Pärast abipredikaadi kasutamist on võimalik automatsioonis kasutada "Result" muutujasse kirjutatud järjendit ehk olemite skoopi. Teine abipredikaat on mõeldud kasutamiseks kui soovitakse leida eelnevalt defineeritud olemite skoobi ja grupis olevate seadmete vahel ühised seadmed. Prolog võtab kaks eelnevalt valmis tehtud olemite skoopi ja otsib kahe skoobi vahel olevaid ühiseid olemeid. Selle leidmisel lisatakse see uude järjendisse. Sellist tegevust nimetatakse kahe hulga ühisosa leidmiseks. Selle teostamiseks kasutab abipredikaat SWI-Prolog predikaati nimega *intersection*[12] (tõlkes

[11] https://www.swi-prolog.org/pldoc/man?predicate=findall/3
[12] https://www.swi-prolog.org/pldoc/man?predicate=intersection/3

“ühisosa”). Predikaadi esimeseks väärtuseks on abipredikaadi päises defineeritud olemite skooop. Teiseks väärtuseks on eelnevalt grupi olemi nime järgi leitud gruppi kuuluvate olemite skoop. Kolmandaks väärtuseks on kahe skoopi vaheline ühisosa, mis pärast predikaadi kasutamist seotakse päises oleva muutujaga “Result”. Näiteks rakendusel on järgmised grupi faktid (vt Joonis 25).

```
group("light.table_lamp", "group.living_room").
group("light.sofa_lamp", "group.living_room").
group("light.eating_table_lamp", "group.kitchen").
group("light.stove_lamp", "group.kitchen").
```

Joonis 25. Prologis defineeritud grupi faktid

Kasutades joonisel 25 defineeritud grupi fakte, rakendus jooksutab elutoa olemite leidmiseks grupi abipredikaati.

```
?- group_scope("group.living_room", Result).
Result = ["light.table_lamp", "light.sofa_lamp"].
```

Joonis 26. group_scope abipredikaadi tulemus

Joonisel 26 on kujutatud *group_scope* predikaadi jooksutamist mille väärtuseks on kasutatud elutoa olemi nime. Jälgides joonist 26 on näha, et abipredikaat tagastas kõik olemid mis kuuluvad gruppi nimega “group.living_room”.

Rakenduse nõude FN-5-2 täitmiseks on loodud kahte tüüpi predikaadid, mis töötavad täpselt samal põhimõttel nagu eelnevalt näidatud *group_scope* abipredikaadid. Antud abipredikaadid on defineeritud nime all *domain_scope*, mille ülesandeks on leida seadmeid domeeni faktide põhiselt (vt Joonis 24).

Rakenduse nõude FN-5-3 täitmiseks on loodud abipredikaat nimega *find_actuator_by_goal*. Antud predikaadi ülesandeks on leida aktuaatoreid kasutades selleks talle ette antud eesmärki ja olemite skoopi. Joonisel 24 kujutatud abipredikaat kontrollib kas etteantud aktuaatori eesmärk on õige ning kas ta on olemite skoobis. Selleks leiab ta kõige pealt aktuaatori nime kasutades aktuaatori hetke oleku ning aktuaatori tüübi. Kasutades hetke olekut ning aktuaatori tüüpi on võimalik meil pärida *action* faktist millist tüüpi eesmärgiga on tegu. Kui päises defineeritud eesmärk kattub *action* faktis defineeritud eesmärgiga, siis kontrollitakse, kas aktuaator on olemite skoobis olemas. Kui aktuaator on skoobis olemas, siis aktuaatori nimi

seotakse päises oleva *Entity_id* muutujaga ning predikaat lõpetab töö tagastades tõese tõeväärtuse. Kui sellist aktuaatorit ei leita, siis predikaat tagastab väära tõeväärtuse.

```
find_actuator_by_goal(Actuator_id, "decrease_light", Scope),
```

Joonis 27. find_actuator_by_goal automatsiooni reeglis

Joonisel 27 on kujutatud *find_actuator_by_goal* predikaati, mida on kasutatud automatsiooni reeglis ühe seadme leidmiseks. Selleks ta kasutab ette seatud eesmärki ehk “decrease_light” ning eelnevalt defineeritud olemite skoopi. Antud predikaat töötab ainult valgust mõjutavate seadmetega.

FN-6 rahuldamiseks on loodud faktid mille nimeks on *action*. Faktid *action* on loodud selle jaoks, et siduda oma vahel olemite olekuid ja kasutaja poolt seatud eesmärke. Näiteks seadmed, mille eesmärgiks on valguse suurendamine.

```
action(light, "increase_light", on).
action(light, "decrease_light", off).
action(switch, "increase_light", on).
action(switch, "decrease_light", off).
action(curtain, "increase_light", up).
action(curtain, "decrease_light", down).
```

Joonis 28. action faktid

Joonisel 28 on kujutatud *action* fakte mis on mõeldud valgus domeeni kuuluvate aktuaatoritele. Fakti esimene väärtus defineerib ära millist tüüpi olemiga on tegemist. Näiteks *light*, *switch* ja *curtain*. Teine väärtus defineerib mida aktuaatori olek teeb ehk kas ta suurendab valgust või vähendab valgust. Kolmas väärtus faktis defineerib milline on olemi olek, et näiteks valgust suurendada.

Rakenduse nõude FN-6 rahuldamiseks on loodud abipredikaat nimega *actuator_state*. Abipredikaadil on defineeritud kaks vormi (vt Joonis 29). Abipredikaatide päises on ära defineeritud olemi nimi ja olemi eesmärk.

```
actuator_state(Entity_id, State):-
    actuator_type(Entity_id, Entity_type),
    action(Entity_type, State, Action),
    actuator(Entity_id, Action),!.
actuator_state(Entity_id, State):-
    actuator_type(Entity_id, Entity_type),
    action(Entity_type, State, Action),
    \+ actuator(Entity_id, Action),
    actuator_action(Entity_id, Action),!.
```

Joonis 29. actuator_state abipredikaadid

Joonisel 26 kujutatud predikaadid on loodud lõputöö jooksul sellepärast, et automatiseerimise reegel ei muudaks olemi olekut selliseks nagu ta juba on. Lisaks on rõhku pandud, et olemeid oleks võimalik lülitada eesmärgipõhiselt. Näiteks selle asemel, et kasutaja ütleb lülita see olem sisse, kirjeldab kasutaja aktuaatori eesmärgi. Esimene predikaat alustab otsinguga mis otsib aktuaatori tüüpi, kasutades selleks olemi nime. Leitud aktuaatori tüüpi ja predikaadi päises defineeritud olekut kasutades leiab predikaat eelnevalt defineeritud *action* (tõlkes tegevus) faktide seast aktuaatori õige oleku. Kui aktuaatori soovitud olek on ära defineeritud predikaadi poolt, siis tehakse kontroll kas aktuaatori hetke olek on sama mis soovitud olek. Kui tegemist on samade olekutega, siis lõpetab predikaat töö. Kui ei ole, siis hakkab Prolog läbi töötama järgmist abipredikaati. Teine *actuator_state* abipredikaat toimib alguses samamoodi, otsides aktuaatori tüübi ja soovitud aktuaatori oleku. Teise predikaadi kolmas rida kontrollib, kas aktuaatori hetke olek ei ole soovitud aktuaatori olek. Kui ei ole, siis rakendus kasutab eelnevalt defineeritud abipredikaati *actuator_action* millega salvestatakse olemi oleku muutus.

```
actuator_state("light.living_room_light", "increase_light").
```

Joonis 30. actuator_state predikaat automatsiooni reeglis

Joonisel 30 on kujutatud automatsiooni reeglis kasutatud abipredikaati *actuator_state*. Abipredikaadi väärtusteks on joonisel tule olemi nimi ning eesmärk valgust suurendada. Predikaati *actuator_state* on võimalik kasutada ainult valgus mõjutatavate seadmetega.

Lõputöö raames viimane valminud abipredikaat, mis rahuldab nõuet FN-6, on *light_sensor*. Antud predikaati kasutades tuleb päises ära defineerida olem mis on sensor tüüpi, milline on soovitud olek ning millist skoopi olemeid antud predikaat mõjutama hakkab.

```
light_sensor(Entity_id, low, Scope) :-
    \+ light_sensor_status(Entity_id, low),
    % Otsime ühe aktuaatori mille eesmärk oli valguse suurendamine ning mis on skoobis olemas
    find_actuator_by_goal(Actuator_id, "increase_light", Scope),
    % Lülitame välja leitud töötava aktuaatori, et valgus taset vähendada
    actuator_state(Actuator_id, "decrease_light"),!.
```

Joonis 31. light_sensor predikaat

Joonisel 31 on eelnevalt kirjeldatud *light_sensor* predikaat, mille eesmärgiks on seatud, et sensori valgustase peab olema madal. Jooniselt on näha, et predikaat kontrollib esialgu antud sensori olemi olekut ning uurib kas see kattub seatud eesmärgiga. Juhul kui ta ei kattu, siis jätkab predikaat tööd ning otsib eelnevalt defineeritud olemite skoobist aktuaatori mille eesmärgiks hetkel on valguse suurendamine ehk *increase_light*. Kui aktuaator leitakse, siis predikaat defineerib, et olemi eesmärgiks peab olema valguse vähendamine. Predikaadi lõpus olev hüüumärk tähendab seda, et kui aktuaatori eesmärk on ära defineeritud, siis lõpetatakse predikaadi töö. Seda tehakse sellepärast, et eesmärgipõhisuse saavutamiseks on meil igakord vaja kontrollida, kas valgus tase on saavutanud eesmärgi. Kuna Prolog jooksvalt olemite fakte ei uuenda, siis kontrollitakse seatud eesmärki uuesti järgmise tsükliga.

4. Ehitatud rakenduse testimine ning analüüs

Selles peatükis analüüsitakse töö käigus valminud rakendust, võrreldakse rakenduse automatiseerimise süsteemi HA platvormi süsteemiga.

4.1 HA ja ehitatud rakenduse automaatikad

Peatükis 3.1.2.3 sai kirjutatud mis on HA platvormi automatiseerimise nõrkused. HA platvormil ei ole võimalik ehitada eesmärgipõhiseid automaatikaid ning väiksemad automaatikad tunduvad kohmakad.

```
rule3 :-
    light_sensor_status("sensor.outside_light_density", low),
    sensor("switch.livingroom_window", off),
    actuator_action("light.living_room_light", "increase_light").

automation rule3:
  - alias: "Turn on lights when its dark outside and the window is closed"
    trigger:
      - platform: numeric_state
        entity_id: sensor.outside_light_density
        below: 130
    condition:
      - condition: state
        entity_id: switch.livingroom_window
        state: "off"
    action:
      - service: light.turn_on
        target:
          entity_id: light.living_room_light
```

Joonis 32. Deklaratiivne reegel ja imperatiivne reegel

Joonisel 32 on välja toodud automaatikad mille tööpõhimõte on täpselt samasugune. Mainida tuleb siinkohal seda, et deklaratiivse reegli valgus sensori fakti väärtuseks on *low*, mis tähendab, et valgus sensori väärtus on väiksem võrdne väärtusega 130. Deklaratiivse reegli esimene rida on HA reegli terve trigger plokk. Sama kehtib ka järgmise kahe deklaratiivse rea ja HA reegli plokiga. Sellise lihtsa automaatika puhul saame väita, et deklaratiivne reegel on (kui HA reeglis alias välja jätta) neli korda pikem. See eest HA reeglis on kõik ära defineeritud, selgelt on aru saada mis automatsiooni käivitab, millised on tema lisa tingimused ning mis on automaatika tulemus.

```
rule :-
    % Kontrollime, kas telekas töötab
    actuator("switch.tv_switch", on),
    % Teeme kindlaks mis seadmed meil grupis on
    group_scope("group.living_room", Scope),
    % Otsime välja õige domeeniga seadmed grupist
    domain_scope("light", Scope, Final_Scope),
    % Kuna reegliks on valguse vähendamine ruumis, siis vähendame valgust
    light_sensor("sensor.motion_sensor_light_level", low, Final_Scope).
```

Joonis 33. Eesmärgipõhine reegel

Joonisel 33 on reegel, mis vaatab kas telekas töötab ning kui jah, siis Prolog otsib välja kõik seadmed mis on elutoa grupis ning mille domeeniks on valgus. Pärast seda on meil seatud valgussensorile eesmärk, milleks on toas valgustuse vähendamine. Antud reegli ülesandeks on otsida välja skoobist üks aktuaator millega oleks võimalik sensori eesmärki täita. Näiteks joonisel 33 on kirjutatud, et valgussensor olek peab olema madal. Antud reegel kontrollib esimesena sensori enda olekut, et kas ikka on mõtet seda automatsiooni jooksutada. Kui kontroll läbitakse, siis otsib reegel ühe seadme skoobist, mis on aktuaator tüüpi. Sellise seadme leidmisel saadab reegel uue oleku HA-le. Rakendus jooksutab antud reeglit intervallidega ehk iga intervall lülitatakse üks seade välja. See võimaldab meil seadmeid lülitada nii kaua kuni me oleme sensorile saavutanud sobiva staadiumi. Sellist tüüpi reegleid pole võimalik HA automatiseerimise süsteemis implementeerida. HA imperatiivse süsteemi eeliseks on see, et selle õppimine ja kirjutamine on lihtsam, sest kõik asjad on plokkideks jaotatud. Sul on vähemalt kaks kindlat asja mida sa pead automatiseerimiseks ära defineerima (trigger ja action plokk).

4.2 Järeldus ja üldine arutelu

Lõputöö jooksul valminud rakendus arendati sellepärast, et ei leidu selliseid targa kodu platvorme, mis võimaldavad eesmärgipõhist automatiseerimist. Töös kasutatud platvorm ning lisaks teised kodu automatiseerimise platvormid mida töö jooksul uuriti ei pakkunud sellist lahendust.

Töös kasutati loogilist programmeerimist sellepärast, et eesmärgipõhise süsteemi arendamine peab toimuma deklaratiivselt. Deklaratiivsete süsteemide arendamise üheks võimaluseks on kasutada loogilist programmeerimist. Rakenduse implementeerimiseks otsustati Prologi kasuks.

Rakendus täidab lõputöös seatud eesmärgi ehk HA platvormil on võimalik seadmeid automatiseerida kasutades Prologi. Rakenduses on võimalik automatiseerida seadmeid kasutades selleks eesmärgipõhisust kui ka HA *trigger-condition-action* põhinevat stiili. HA stiilis rakenduses on võimalik seadmeid automatiseerida kasutades selleks lihtsalt abipredikaati millega on võimalik soovitud seadme olek salvestada. Nendeks seadmeteks on *light*, *switch* ja *climate* domeeni kuuluvad seadmed. Lisaks on rakendus võimeline eesmärgipõhiseid automatsioone läbi viima. Selle jaoks on loodud abipredikaadid, mida peab kasutama, et sellist tüüpi automatsiooni teha. Eesmärgipõhist automatsiooni on võimalik teha valgust mõjutavate

seadmetega kasutades selleks abipredikaate. Näiteks abipredikaati mis küsib kasutajalt sensori nime, soovitud valgustaset ning seadmete ja teenuste skoopi. Selliseid automatsioone kasutades on võimalik meil luua intelligente automaatika, mis ei käitu robustselt nagu hetkel HA süsteemi automatsioonid.

Rakenduse kasutamine ei ole mõistlik kui soovitakse ehitada lihtsaid automatsioone. Selliste automatsioonide ehitamiseks piisab täiesti HA platvormil pakutavast automatiseerimise süsteemist. Rakendus hakkab enda võimekust näitama, siis kui soovitakse luua eesmärgipõhiseid automatsioone. Näiteks kui telekas töötab, siis toa valgustase peab olema madal, siis lülita ükshaaval välja valgust mõjutavaid seadmeid kuniks eesmärk on saavutatud. Selliseid automatsioone HA platvormi pole võimalik teha, sest HA automatsiooni süsteem imperatiivsele süsteemile kohaselt nõuab peab automatsiooni reeglis kõik rangelt defineeritud olema. Näiteks reeglis peab olema rangelt kirjas kõik aktuaatorid mida lülitatakse vastavalt automatsioonile.

Rakenduse integreerimine platvormile ei ole keeruline. Enne rakenduse käivitamist tasub üle vaadata HA platvormil millised on valgus ja temperatuuri sensorite mõõtühikud. Kui need erinevad, siis teha rakenduses vastavad muudatused. Kui soovitakse hakata kirjutama automatsioone on soovituslik enne rakendus käima panna nii, et ei kasutata automatsiooni reegleid. Selle tulemusel on võimalik kätte saada kõik seadme ja teenuse faktid ning nende abil luua kohe korrektseid automatsiooni reegleid.

Rakenduse arendamisel sai selgeks, et väga keeruline on luua automatiseerimise süsteemi kui pole välja mõeldud korraliku ontoloogiat mille järgi süsteemi ehitada. Korralik seadmete ja teenuste ning selle automatiseerimise ontoloogia aitab kaasa Prologi kasutamisel. Prologi idee seisneb selles, et faktide vahele ehitatakse relatsioone. Kui mudeli relatsioonid on head, siis Prologis seda implementeerida pole keeruline. Selle lõputöö jooksul oli see heaks õppetunniks, sest ilma mudelita sellist asja on väga keeruline valmis ehitada.

Peale nende domeenide ja olemite mis lõputöö jooksul sai valmis implementeeritud, leidub HAs veel objekte mida on võimalik juurde implementeerida. Näiteks pakub HA palju erinevaid domeene mida on võimalik kasutada kas HA stiilis või eesmärgipõhiseks automatiseerimiseks. Rakenduse ehitamisel on jälgitud seda, et uute asjade integreermine oleks võimalikult lihtne. Näiteks uue domeeni implementeerimiseks võiks kõrvale võtta valgus domeeni lahenduse,mis annab idee kuidas seda on implementeeritud nii Pythonis kui ka Prologis. Rakenduse kasutaja kes hakkab tuge implementeerida võiks selgeks teha kuidas rakendus Prologi fakte teeb, kuidas

Prologis kirjutatud automatiseerimise predikaadid toimivad ning kuidas rakendus automatsioonidelt saadud tulemusi HA-sse edasi saadab. Nende mõistmine aitab tööprotsessi kindlasti kiirendada uute seadmete lisamisel ning vastava toe implementeerimisel.

5. Kokkuvõte

Lõputöö eesmärkideks oli valmis ehitada automatiseerimise süsteem millega oleks võimalik seadmeid HA platvormi automatiseerida Prologi kasutades, uurida loogilise programmeerimise võimekust targa kodu seadmete automatiseerimisel ja millistes olukordades on Prolog põhine automaatika eelistatud HA põhisele automaatikale. Rakenduse loomiseks uuriti ja analüüsiti avatud lähtekoodiga platvorme. Seda tehti sellepärast, et leida kas selline süsteem leidub, kus automatiseerimiseks on kasutatud loogilist programmeerimist. Lisaks uuriti ka süsteemiga sarnaseid lahendusi ning võrreldi lõputöös valminud rakendust leitud lahendustega.

Lõputöö jooksul valmis rakendus millega oli võimalik automatiseerida HA platvormil olevaid seadme ja teenuse olemeid. Rakendusele on implementeeritud *light*, *switch*, *sensor*, *group*, *person*, *climate* domeenis olevate olemite tugi. Kõik toetatud olemid salvestatakse sõnastikku ning seda uuendatakse kui HA platvormil toimub olemitega mingi muudatus. Sõnastiku abil loob rakendus automatiseerimiseks vajalikud Prolog faktid. Lõputöö jooksul implementeeriti Prologis automatiseerimiseks vajalikud abipredikaadid mida kasutades on võimalik salvestada aktuaator tüüpi olemite olekuid ning luua eesmärgipõhiseid automatsioone. Saadud automatsiooni tulemused edastab rakendus HA platvormile kus tehakse vastavad muudatused.

Töö jooksul selgus, et mõlematel automatiseerimise süsteemidel on omad plussid ja miinused. Näiteks HA automatiseerimise süsteemil puudub võimekus luua eesmärgipõhiseid süsteeme. Sellist võimekust pakub lõputöö jooksul valminud rakendus. HA automatiseerimise süsteemil põhinevaid reegleid on võimalik valminud rakenduses ka kirjutada. Selliseid reegleid on mõistlik kirjutada HA platvormil, sest automatsiooni reeglite struktuur on loogiline ning kergesti õpitav. Valminud rakendus samas näitas, et võrreldes HA automatsiooni reeglitega, on reeglite pikkus oluliselt väiksem.

Rakenduse ehitamisel on jälgitud, et uute asjade integreermine oleks võimalikult lihtne ja loogiline. Kasutaja kes rakendusele implementeerib näiteks domeeni tuge võiks võtta näitena kõrvale implementatsiooni mis rakenduses on olemas. Uue toe implementeerimisel on soovituslik eelnevalt selgeks teha kuidas rakendus loob Prolog fakte, kuidas lõputöö jooksul loodud automatiseerimise predikaadid toimivad ning kuidas rakendus tulemusi HA platvormile edastab. Nendest arusaamine aitab rakenduse arendamise protsessi kiirendada.

Viidatud kirjandus

[1] Home Assistant koduleht: https://www.home-assistant.io/ (9.12.2021)

[2] Home Assistant Core Github: https://github.com/home-assistant/core (9.12.2021)

[3] Home Assistant integratsioonid: https://www.home-assistant.io/integrations/ (9.12.2021)

[4] Home Assistant seadmete ja teenuste integreermine: https://developers.home-assistant.io/docs/architecture/devices-and-services (08.05.2022)

[5] Home Assistant olemid: https://developers.home-assistant.io/docs/core/entity (08.05.2022)

[6] Home Assistant sõnastik: https://www.home-assistant.io/docs/glossary/ (08.05.2022)

[7] Home Assistant Group integratsioon: https://www.home-assistant.io/integrations/group/ (08.05.2022)

[8] Home Assistant Configuration.yaml: https://www.home-assistant.io/docs/configuration/ (08.05.2022)

[9] Home Assistant YAML: https://www.home-assistant.io/docs/configuration/yaml/ (08.05.2022)

[10] Home Assistant REST rakendusliides: https://developers.home-assistant.io/docs/api/rest (08.05.2022)

[11] Home Assistant WebSocket rakendusliides: https://developers.home-assistant.io/docs/api/websocket (08.05.2022)

[12] Home Assistant automaatika: https://www.home-assistant.io/docs/automation/ (10.12.2021)

[13] Home Assistant automatiseerimise põhitõed: https://www.home-assistant.io/docs/automation/basics/ (10.12.2021)

[14] Mahling D. E., King R. C. A Goal-Based Workflow System for Multiagent Task Coordination. *Journal of Organizational Computing and Electronic Commerce*, 1999, nr 9:1, 57 - 82.

[15] Home Assistant automaatiseerimise formaat YAML: https://www.home-assistant.io/docs/automation/yaml/ (11.12.2021)

[16] HA Automation näide: https://www.home-assistant.io/docs/automation/yaml/ (08.05.2022)

[17] Paulus Schousten github: https://github.com/balloob (05.05.2022)

[18] Home-Assistant Core github informatsioon: https://api.github.com/repos/home-assistant/core (07.05.2022)

[19] Genesereth M., Chaudhri V. K. Introduction to Logic programming. California: Morgan and Claypool Publishers. 2020.

[20] Bisicchia G., Forti S., Brogi A. A Declarative Goal-oriented Framework for Smart Environments with LPaaS, 2021. https://arxiv.org/pdf/2106.13083.pdf (03.04.2022)

[21] D'Urso F., Longo C. F., Santoro C. Programming Intelligent IoT Systems with a Python-based Declarative Tool, 2019. http://ceur-ws.org/Vol-2502/paper5.pdf (14.04.2022)

[22] Setz B., Graef S., Desislava I., Tiessen A., Aiello M. A Comparison of Open-Source Home Automation Systems, 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9652536 (15.04.2022)

[23] Node-RED: https://nodered.org/ (15.04.2022)

[24] OpenHAB integrations: https://www.openhab.org/addons/ (15.04.2022)

[25] OpenHAB rules: https://www.openhab.org/docs/configuration/rules-dsl.html (15.04.2022)

[26] ioBroker javascript documentation: https://www.iobroker.net/ - en/documentation (16.04.2022)

[27] Bramer M. Logic Programming with Prolog. London: Springer. 2013.

Lisad

I. Home Assistant platvormi poolt pakutavad domeenid

Domeen	Kirjeldus
Alarm Control Panel	Olemi tüüp, mis kontrollib kodus nutikat turvasüsteemi. Domeeni lisatud olemi väärtusteks on tema olek. Vajaduse korral on võimalik näha kes olemi väärtust muutis ning tähistada kuidas turvasüsteemi alarmi käivitamine käib. Lähemalt: https://developers.home-assistant.io/docs/core/entity/alarm-control-panel
Binary Sensor	Olem sensoritest, mille väärtuseks saab olla kaks olekut (0 või 1). Lähemalt: https://developers.home-assistant.io/docs/core/entity/binary-sensor
Button	Olem mida võib võtta paralleelina reaalses elus lüliteid jms. See domeen võimaldab platvormil integreeritute seadmete puhul algatada mingit tegevust. Näiteks taaskäivita seadmeid. Antud domeen ei asenda päris lüliteid vaid on mõeldud platvormi siseseks kasutamiseks. Lähemalt: https://developers.home-assistant.io/docs/core/entity/button
Calender	Olem, mille abil on võimalik meelde jätta sündmused seadistades selleks algus ja lõpp kuupäev. Lähemalt: https://developers.home-assistant.io/docs/core/entity/calendar
Camera	Olem, millega on võimalik kuvada pilte ning kaamerate otseülekannet. Lähemalt: https://developers.home-assistant.io/docs/core/entity/camera
Climate	Olem, mille abil on võimalik õhukonditsioneeride ja niisutajate temperatuuri, niisutamis taset jms. kontrollida. Lähemalt: https://developers.home-assistant.io/docs/core/entity/climate
Cover	Olem, mille abil on võimalik kontrollida näiteks akna kaardinad,

	uksed jms. Lähemalt: https://developers.home-assistant.io/docs/core/entity/cover
Device Tracker	Olem, millega on võimalik jälgida seadme asukohta kasutades kas GPS koordinaate või uurides kas seade on lokaalsesse võrku ühendatud. Lähemalt: https://developers.home-assistant.io/docs/core/entity/device-tracker
Fan	Olem, millega on võimalik kontrollida ventilaatoreid. Lähemalt: https://developers.home-assistant.io/docs/core/entity/fan
Humidifer	Olem, millega on võimalik kontrollida õhuniisutajaid. Lähemalt: https://developers.home-assistant.io/docs/core/entity/humidifier
Light	Olem, millega on võimalik kontrollida tulesid. Näiteks heledust, värvi temperatuuri jms. Lähemalt: https://developers.home-assistant.io/docs/core/entity/light
Lock	Olem, millega on võimalik kontrollida lukkusid. Näiteks ukselukk. Lähemalt: https://developers.home-assistant.io/docs/core/entity/lock
Media Player	Olem, millega on võimalik kontrollida helisüsteeme jms. Lähemalt: https://developers.home-assistant.io/docs/core/entity/media-player
Number	Olem, mille abil on võimalik kasutajal seadistada integratsioonile mingi väärtus. Lähemalt: https://developers.home-assistant.io/docs/core/entity/number
Remote	Olem, mis on nagu universaalne pult. Olemisse on võimalik lisada seadmete ja teenuste tegevusi. Lähemalt: https://developers.home-assistant.io/docs/core/entity/remote
Select	Olem, mille abil on võimalik kasutajal valida olemisse seadistatud integratsioonide vahel. Lähemalt: https://developers.home-assistant.io/docs/core/entity/select

Sensor	Olem, mis annab edasi mingeid andmeid. Näiteks temperatuuri või valgus taset. Lähemalt: https://developers.home-assistant.io/docs/core/entity/sensor
Siren	Olem, millega on võimalik kontrollida näiteks uskekellasid või muid sarnaseid seadmeid. Lähemalt: https://developers.home-assistant.io/docs/core/entity/siren
Switch	Olem, millega on võimalik midagi sisse ja välja lülitada. Näiteks nutikad pistikud. Lähemalt: https://developers.home-assistant.io/docs/core/entity/switch
Update	Olem, mille abil on võimalik jälgida kas integratsioonid on uuendatud viimasele versioonile ning neid vajaduse korral paigaldada. Lähemalt: https://developers.home-assistant.io/docs/core/entity/update
Vacuum	Olem, mille abil on võimalik kontrollida robottolmuimejaid. Lähemalt: https://developers.home-assistant.io/docs/core/entity/vacuum
Water Heater	Olem, mille abil on võimalik kontrollida veekütte süsteeme. Lähemalt: https://developers.home-assistant.io/docs/core/entity/water-heater
Weather	Olem, mille abil on võimalik jälgida ilma seisu. Lähemalt: https://developers.home-assistant.io/docs/core/entity/weather

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Markkus Koodi,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose Targa kodu automatsioonid loogilise programmeerimisega Home Assistant platvormil, mille juhendaja on Jakob Mass, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Markkus Koodi

09.05.2022