TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

IAY40LT

Mihkel Kasepuu 123993IAPB

Juhtmevaba nutipistiku integratsioon olemasoleva nutikodu lahendusega

Bakalaureuse töö

Mairo Leier
MSc
Nooremteadur

Tallinn 2015

Autorideklaratsioon

Autorideklaratsioon on iga lõputöö kohustuslik osa, mis järgneb tiitellehele. Autorideklaratsioon esitatakse järgmise tekstina:

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

Autor: Mihkel Kasepuu

05.05.2015

Annotatsioon

Käesoleva töö eesmärgiks on õppida tundma riistvara, tarkvara ning andmebaaside koostöötamise lahendusi ning nende teadmiste põhjal ehitada töötav juhtmevaba nutipistik.

Nutilülituse tekitamiseks on mitmeid võimalusi. Kergeks lahenduseks on iga pistiku juurde vedada mingi kaabel, mis justkui tõeväärtusena ütleks kas lahendus peab töötama või mitte. Tihtilugu on kaablivedu ebamugav ning otsitakse alternatiive. Käesolev töö pakub juhtmevaba lülituse jaoks ühe näite.

Antud töö käigus valmis juhtmevaba nutipistik, kus pistiku töötamist implementeerib releemoodul. Lisaks disainiti kaks lülitusmustrit kuidas antud pistikut kontrollida ning andmebaasi disain hoidmaks pistiku ning tema lülituste kohta andmeid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 40 leheküljel, 8 peatükki, 21 joonist, 2 tabelit.

Abstract

Wireless smartcord integration with existing smarthome solution

The purpose of this work is to learn and research patterns for creating software, hardware and database solutions together as a one project. This knowledge is used to build a wireless smart cord.

There are many ways to create a smart switching solution. The easiest way is to create smart switching is by cable that can control the cord with Boolean values. Having cables and adding them is often more difficult than preferred solution. This thesis will offer one example for wireless switching.

The result of this thesis is a wireless smart cord as a relay. The smart cord can be switched automatically and manually using a database solution that was created to maintain cord's info and switching data.

The thesis is in Estonian and contains 40 pages of text, 8 chapters, 21 figures, 2 tables.

Lühendite ja mõistete sõnastik

ATI	TTÜ Arvutitehnika instituut
UART	Universaalne asünkroonne vastuvõtja/transmitter, riistvara osa, mis aitab infot saata bit bit-i haaval
SPI	Sünkroonse bit-i edastuse liides, mis kasutab nelja loogika kanalit info liigutamiseks
CASE	Computer-aided Software Engineering, Arendustööriist, mis võimaldab luua erinevaid tarkvara ja riistvara lahendusi
MVC	Model-view-controller, tarkvara disainimuster
Clean-code	Puhas kood, ehk hästi loetav ning vormistatud tarkvara kood [1]
Tagarakend	Backend, Maakeelne vaste ingliskeelsele IT-s kasutatavale teriminile "backend".

Sisukord

Jooniste nimekiri

Tabelite nimekiri

1. Sissejuhatus

Tänapäeval elades tuleb aina rohkem harjuda erinevate kiirelt arenevate tehniliste lahendustega meie ümber ning ka neid kasutama hakata. Kõik need lahendused on loodud, et ümbritsevat elu mugavamaks muuta mingi varasema tegevuse arvelt. Paari viimase aasta jooksul on järjest rohkem uusi lahendusi nimetatud *nuti*-lahendusteks. Ka käesoleva töö sisu on nutilahenduse võimalik kirjeldus, mis võiks lähitulevikus iga indiviidi igapäeva elu osa olla. Nimelt on nutiseadmed jõudnud kodudesse ja majadesse ning võimaldavad sealseid elemente kontrollida. Käesolev töö on nutimaja üks osa, mis disainitud kontrollimaks ühe kodu pistikuid.

1.1. Taust ja probleem

Käesoleva töö peamiseks aluseks oli Tallinna Tehnikaülikooli Arvutitehnika Instituudi arenduses olev nutikodu lahenduse põhi, mis võimaldas kontrollida erinevaid *Raspberry Pi* [2] peale ehitatud võimalusi üle veebiliidese nagu näiteks veebikaamera. Probleemiks oli koheselt olemasoleva põhja võimalused uusi funktsionaalsusi juurde lisada, mis tekitaks suurel mahul segadust lähtekoodis ning raskendaks sõltuvuste ja failinimede haldust.

Kuna nutikodu peamiseks funktsionaalsuseks on võimalus erinevaid seadmeid eemalt sisse-välja lülitada on käesolev töö heaks näiteks kuidas parema arhitektuuri korral nutikodule lisasid liita. Selleks tuleb disainida nutipistik, mille komponendid oleks omavahel hästi ühilduvad ja omaks võimalust ühilduda Arvutitehnika Instituudi nutikodu põhjaga.

Nutipistiku puhul on ka vajadus, et see oleks konfigureeritav nii aja, kui ka teiste nutipistikute suhtes.

Käesoleva tööga tegeleti 2015 aasta kevadel.

1.2. Ülesande püstitus

1. Ehitada nutipistik, mis on juhtmevabalt juhitav.
2. Liidestada nutipistik olemasoleva põhjaga ja disainida sellele veebipäringud.
3. Veebiliidese ehitamisel ümber disainida *python*-il põhinev veebirakenduse arhitektuur paketipõhisele kasutades *MVC* ja *clean-code* põhimõtteid.
4. Tekitada näide automaatsest reeglipõhisest lülitusest

1.3. Metoodika

Käesolev töö valmis rakendades *agiilse* [3] arenduse põhimõtteid, kus püüti aru saada TTÜ Arvutitehnika instituudi vajadustest. Antud vajadusi mõistes hakati ehitama nutipistikut, kasutades algselt TTÜ Arvutitehnika instituudi näiteid ja nende põhjasid ning hiljem uut koodi kirjutades. Lisaks harilikule koodikirjutamisele kasutati erinevaid *CASE* vahendeid nii andmebaasi disainimisel, kui ka mitmete mudelite ehitamisel.

1.4. Ülevaade tööst

Käesolev töö koosneb neljast suuremast sisupeatükist, kus esimeses peatükis „Nutikodu tähendus“ kirjeldatakse laial taustal nutikodu vajalikkust ja olemust. Järgnevad peatükid kirjeldavad lähemalt nutipistiku valmimist, kus esimene neist, peatükk „Ülesehitus“, kirjeldab üldist rakenduse disanimustrit. Järgnev peatükk „Riistvara arhitektuur“ kirjeldab riistvara ehitust ja kasutust ning peatükk „Tarkvara arhitektuur“ läheneb kooditasemel ehitatud lahendustele.

2. Nutikodu tähendus

Käesolev töö on lisa nutikodu süsteemile. Nutikoduks (või nutimajaks) nimetatakse maja, mis annab intelligentset tagasisidet ja informatsiooni majas toimuvate erinevate tegevuste kohta [4]. Nutikaks teeb nutikodu temas töötavate seadmete kasutusmugavus ja kasutusvõimalused. Turul on kodu nutikaks muutmiseks mitmeid võimalusi, kuid nende maksumus on kallis ja sagel jäävad nende laiendusvõimalused tootja poolt piiratuks. Näiteks ettevõtte *Fibaro* [5] tooted.

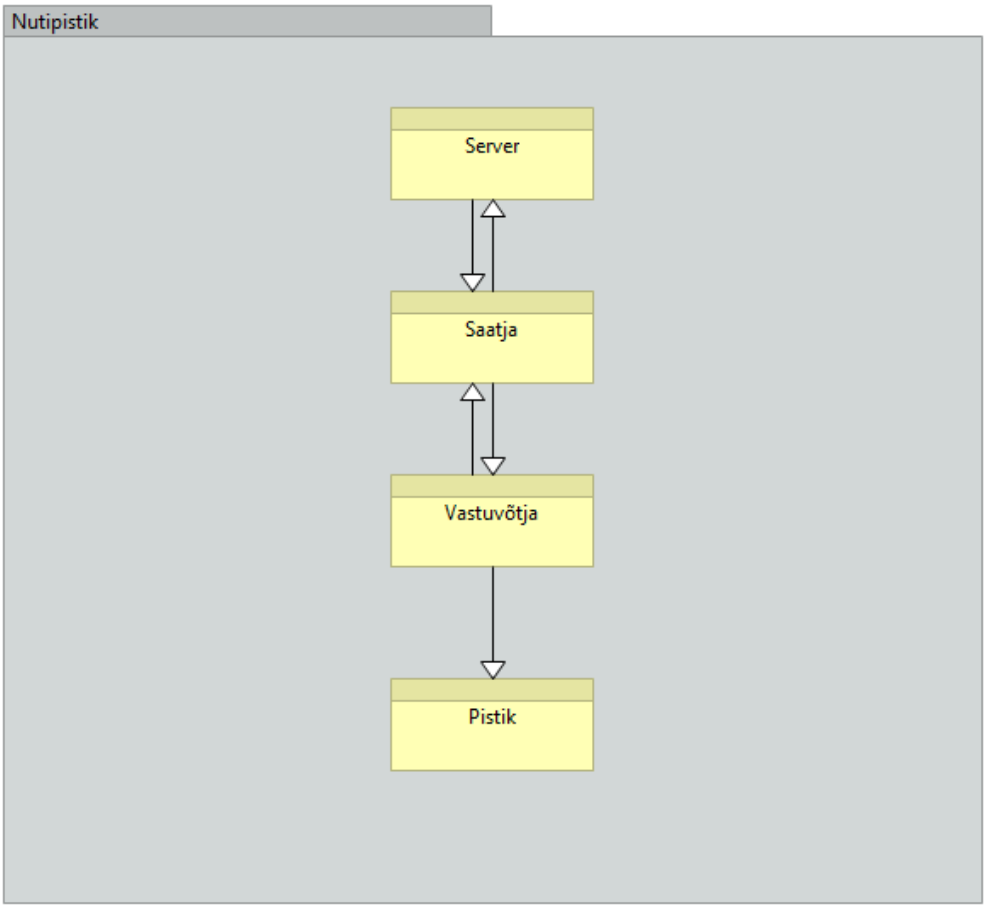
Käesolev töö püüab läheneda nutikodu ehitamisel kui avatud lähtekoodiga projektile. Sellest järeldades võib öelda, et nutikodu tähendab käesoleva töö jaoks lisaks toote enda lahendustele ka võimalust uusi soodsamaid, enese programmeeritud ja ehitatud arendusi kergelt olemasoleva süsteemiga liidestada. Seega jääb antud nutikodu piiratus võrdeliseks ainult kasutaja enese soovide ja hetke tehnoloogia võimalustega, kui mitte arvestada finantsvõimekust.

Kuigi nutikodu suudab teha elukohas elamise mugavaks, on sellel ka oluline väärtus säästmisel. Iga koduomanik peab oma elukoha elektri ja veenäite kontrollima ning avaldama, mille osas nutikodu aidata oskab. Hea nutikodu suudab veelgi rohkem abistada, hoides neid numbreid madalal. Näiteks elektri kokkuhoiu valdkonnas võiks nutikodu katkestada kõikide ooteseisundil olevate seadmete toite kui omanik on kodust lahkudes sisse lülitanud turvasüsteemi. Sellised lihtsad lülitused võimaldavad kasutajal võrreldes tavalise koduga aasta lõikes märkimisväärselt säästa.

Nutikodud ei ole väga levinud, kuigi selle üksikud komponendid ei ole keerukad. Nende suurimaks miinuseks on raskendatud asjaolud toote integratsioonil vanema hoonega, kus sageli pole igas toas kaabliga internetiühendust, ega kerget võimalust selle tekitamiseks. Seega on käesolev töö lähtunud juhtmevabadest lahendusest pistikute vahel. Selle eeliseks on omada ühendust kindlate pikkuste vahel erinevate seadmetega ilma füüsilise ühenduseta, kuid tõstab kasutajates üles turvalisuse küsimuse.

3. Süsteemi ülesehitus

Käesoleva töö nutipistiku tööjada koosneb 4-st komponendist: **serverist**, **saatjast**, **vastuvõtjast** ja **pistikust**, nagu näidatud joonisel 1.



Joonis 1 Lihtsustatud ülesehituse skeem

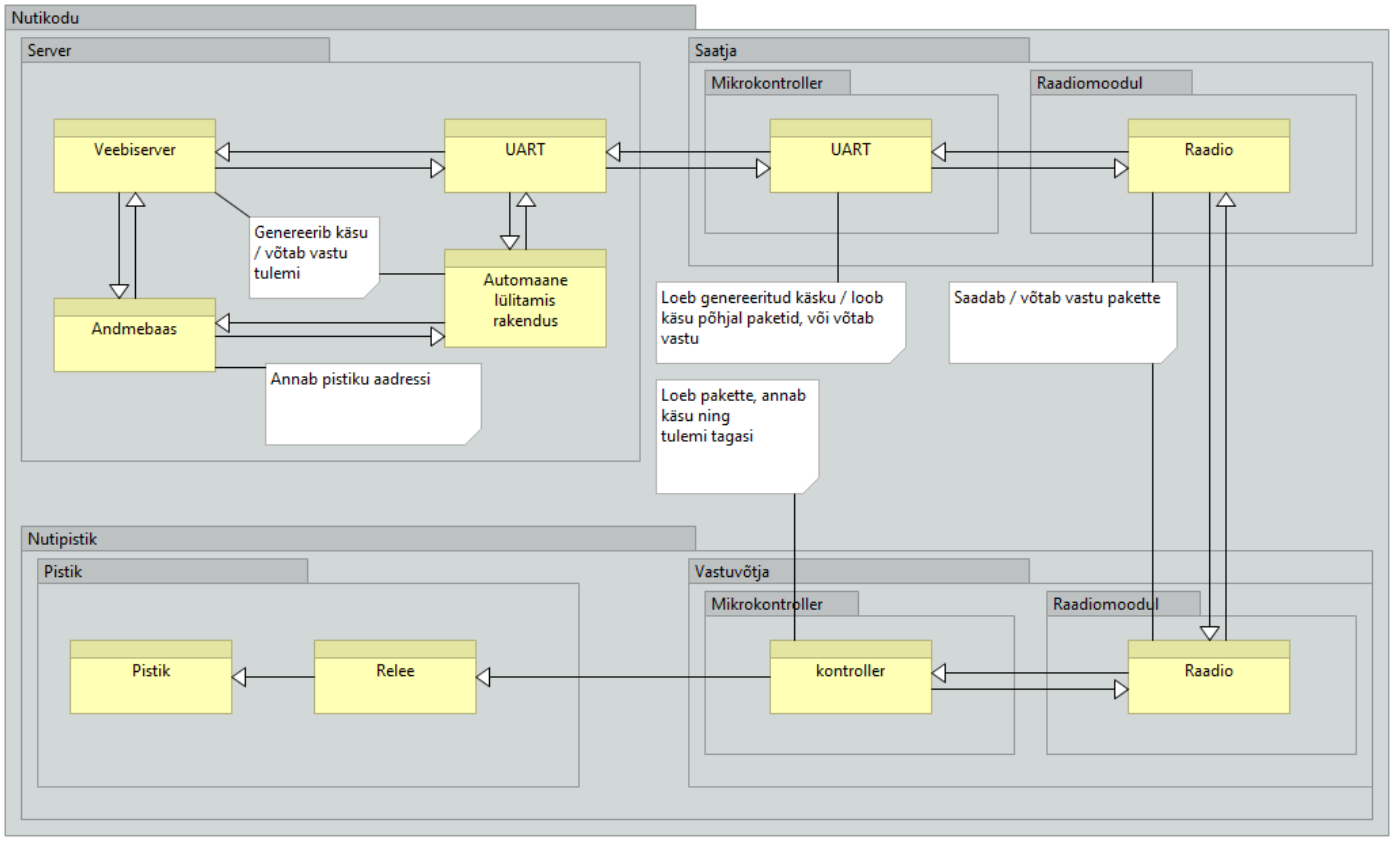
Serveri osa hakkab haldama kõike, mis on kasutaja poolt defineeritud ning talle veebiliidese poolt kuvatav. Ta hoiab pistikute infot andmebaasis, kus on defineeritud kas pistik, mingil ajahetkel ja missugustel tingimustel peaks töötama või mitte, lisaks tema aadress ja määratud nimi. Pistiku sees ja väljas olemise tingimusi saab muuta kasutades veebilehitsejat, mis pöördub serveril asetseva veebirakenduse poole. Lisaks kontrollib aja põhine lülitamis rakendus iga kindla ajavahemiku tagant. Server suhtleb saatjaga kasutades UART liidest [6].

Saatjat kasutab server, et anda edasi konkreetne kasutaja poolt seatud lülitus nutipistikule, mis on kas manuaalselt või automaatselt valitud.

Saatja osa haldab *UART*-ilt saabunud ja sinna minevaid sõnumeid, kui ka vastuvõtjale saadetavaid ning sealt tulevaid pakette. Ta suudab saabunud sõnumitest välja lugeda vastuvõtja aadressi ja käsu, kas sisse või välja. Sellest tulenevalt seatakse paketid ja suunatakse vastuvõtja poole.

Vastuvõtja on eraldiseisva nutipistiku esimene komponent. Vastuvõtjas loetakse saabunud pakette, kus defineeritakse, mis on konkreetse sõnumi eesmärk, kas pistik lülitatakse sisse või välja. Pärast eesmärgi selgitamist edastab vastuvõtja vastava käsu releele.

Pistik on sisuliselt relee. Vastavalt vastuvõtjas olevale pingele lülitatakse pistik sisse ja välja.



Joonis 2 Üldine ülesehituse skeem

4. Riistvara arhitektuur

Riistvara on nutikodu südameks. Kõik komponendid peavad olema võimelised omavahel suhtlema. Käesoleva töö valmimiseks valitud komponendid on kõik väikese voolutarbega ja hea hinna ning kvaliteedi suhtega.

4.1. Komponendid

- **Server** – Raspberry Pi Model B
- **Saatja** – Texas instruments Launchpad MSP430G2 [7] + Microchip MRF89XA [8]
- **Vastuvõtja** – Texas instruments Launchpad MSP430G2 + Microchip MRF89XA
- **Pistik** – 3.3V poolt juhitav relee

Server Raspberry Pi osutus käesoleva töö jaoks valituks oma populaarsuse ja võimekuse tõttu. Populaarsus aitab seda liidestada erinevate komponentide vahel, kuna üksikute lisade piires on seda suurema tõenäosusega juba katsetatud.

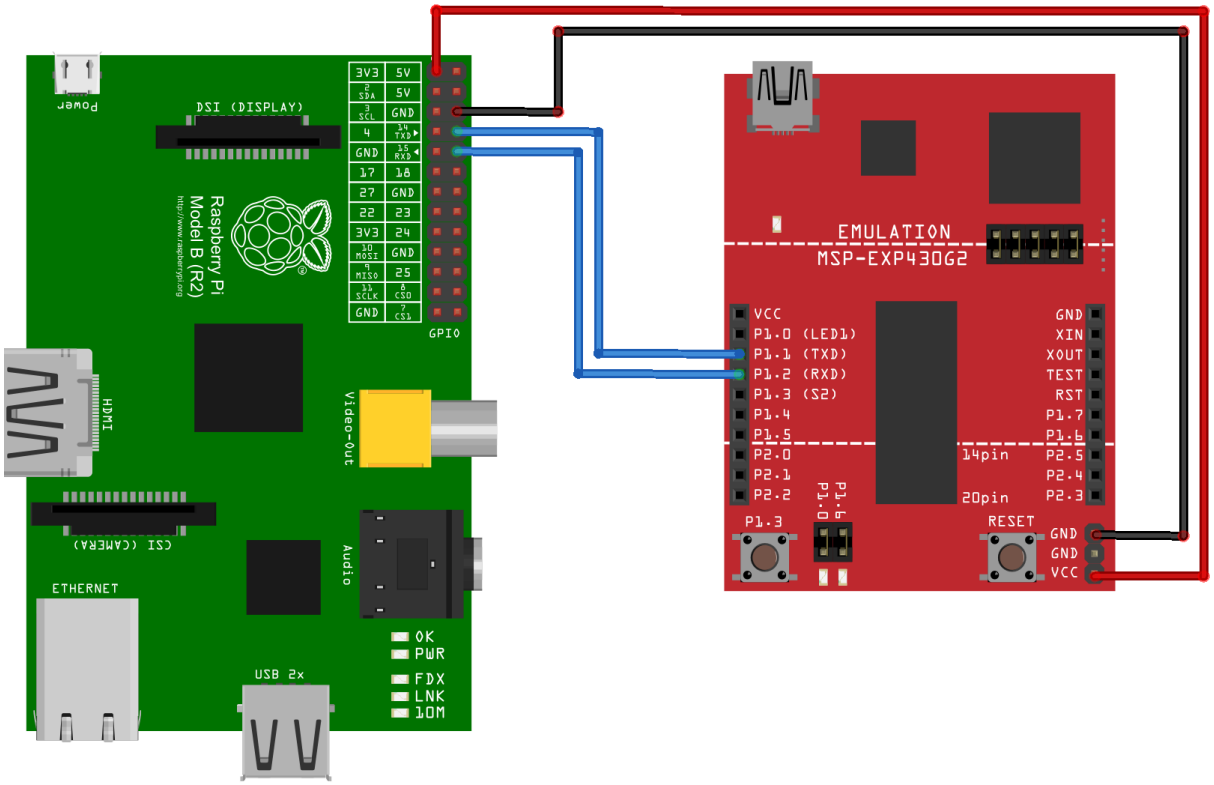
Saatja ja **vastuvõtja** TI Launchpad MSP430G2 valiti tema hinna ja kvaliteedi suhte ning tööpinge pärast. Nimelt on MSP430 odavam kui konkurent Arduino Uno ning töötab 3.3V piires. MRF89XA osutus valituks oma võimekuse pärast, mis ilmnes Arvutitehnika Instituudi kogemusest. Ka MRF89XA on väga odav ja töötab madalatel sagedustel (862MHz), mis tagab laiema leviala võrreldes sarnaste 2,5 GHz moodulitega.

4.2. Ühendused

i. Serveri ja Saatja vaheline ühendus

Raspberry Pi ja MSP430 vahel, kus viimane töötab kui saatja, kasutati info vahetamiseks *UART* liidest, et tõlkida andmeid, mis bit bit-i järel vahetati. Et üle *UART* liidese infot vahetada kahe seadme vahel, tuleb ühe seadme saatev viik ühendada teise seadme vastuvõtva viiguga ning vastupidi. See tähendab, et ühel pool ühendust on üks kanal ainult info vastuvõtmiseks ning teine kanal ainult saatmiseks, ehk lihtsustatult on Raspberry Pi ja MSP430 vahel kaks juhet info vahetamiseks (Joonisel 3 sinised kaablid). Lisaks info vahetamiseks on mikrokontrolleri ja serveri vahel ühendatud toite ja maanduse juhe (Joonisel 3 vastavalt punane ja must juhe).

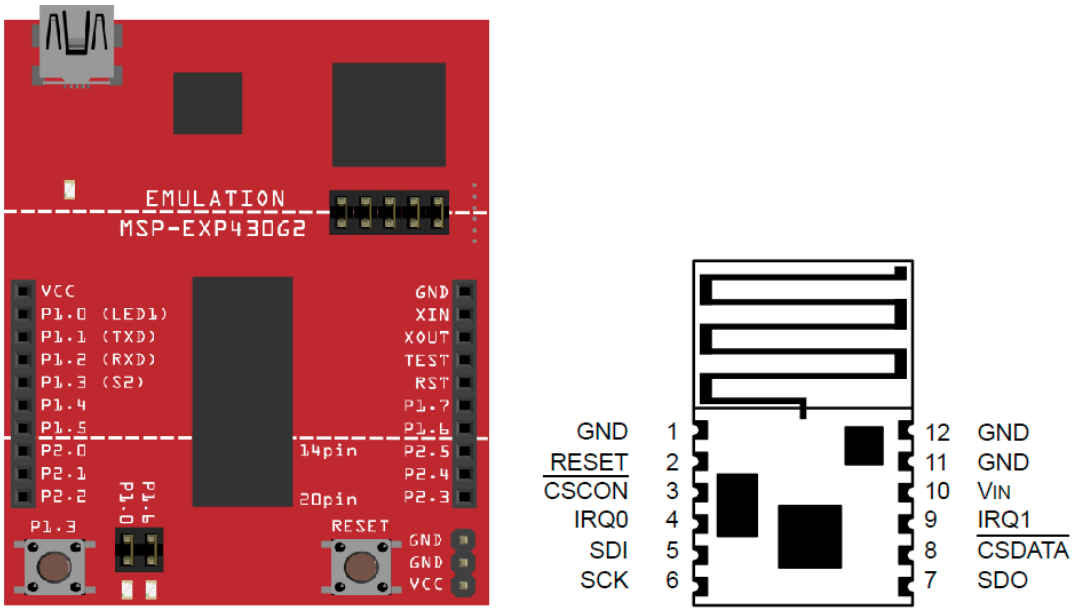
Selleks, et Raspberry Pi ja MSP430 saaksid infot vahetada on mõlemal tarvis ühenduda vooluvõrku. Raspberry vool saavutatakse läbi Micro USB toite sisendiga. MSP430 toidet saab tänu madalale vooluvajadustele võtta serverilt kasutades Raspberry Pi [9] 3,3V pinge ja maanduse allikat.



Joonis 3 Raspberry Pi ja MSP430 UART ühendus

ii. MSP430 ja MRF89XA vaheline ühendus

Mikrokontroller MSP430 ja raadiomoodul MRF89XA vahel saavutatakse ühendus 6 juhtme abil. Kui Raspberry Pi ja MSP430 vahel oli asünkroonne ühendus kasutades UART-i, siis MSP430 ja MRF89XA vahel kasutatakse SPI [10] liidese poolt defineeritud sünkroonset andme edastust. MRF89XA toide ja maandus on võetud MSP430 Launchpadi pealt.



Joonis 4 MSP430 ja MRF89XA

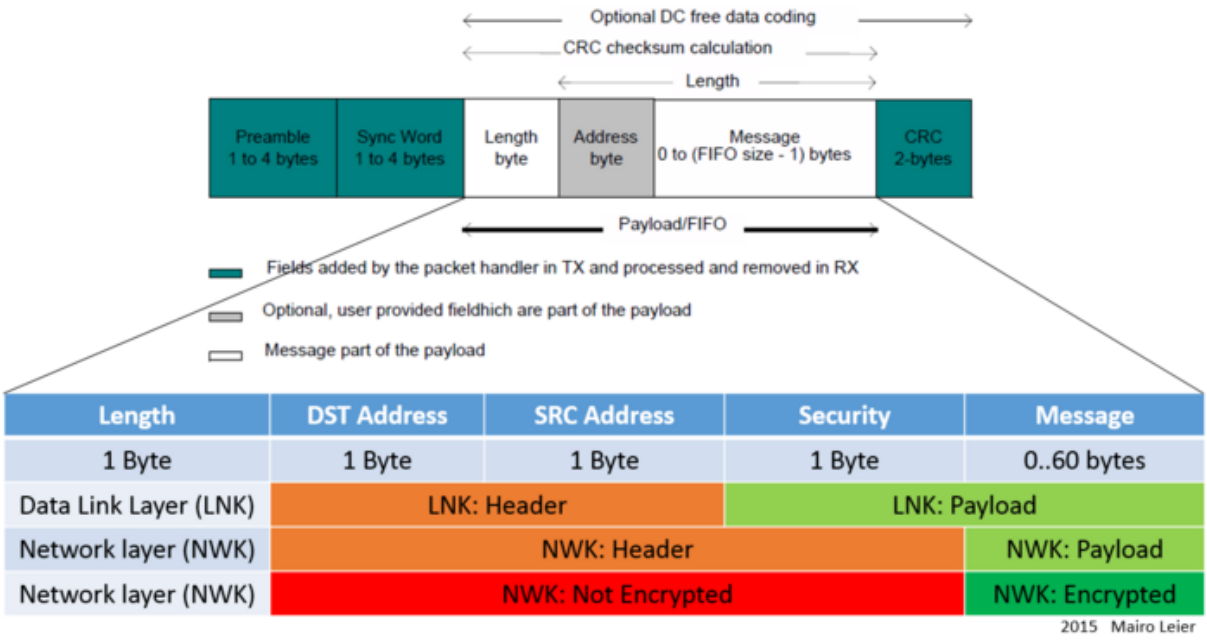
Tabel 1 MSP430 ja MRF89XA vaheline ühendus [11]

Kirjeldus	**MRF89XA**	**MSP430**
Maandus	1 GND	GND
Reset nupp	2 RESET	P2.5
SPI Conf Chip Select	3 CSCON	P1.4
Interrupt 0	4 IRQ0	P2.0
SPI MOSI	5 SDI	P1.7
SPI Clock	6 SCK	P1.5
SPI MISO	7 SDO	P1.6
SPI Data Chip Select	8 CSDATA	P1.3
Interrupt 1	9 IRQ1	P2.1
Toide	10 VIN	VCC

iii. Kahe MSP430 vaheline ühendus

Kaks MSP430 kasutavad küll omavahel suhtlemiseks MRF89XA-d, kuid paketid suhtlemiseks defineeritakse mikrokontrolleri tasandil.

Pakettide struktuur. Käesolev töö saadab pakette üle võrgukihi (*Network layer* [12] – NWK).



Joonis 5 Pakettide jaotus [11]

Võrgukiht sai valituks üle andmekihi (*Data link layer* – LNK [13]), sest see annab võimaluse kontrollida pakettide vastuvõtmist (*ACK*), kui ka info turvalisust tõsta. Käesolev töö kontrollib eemal seisvat nutipistikut saadetud paketi sõnumi viimase baiti alusel. Kui viimane bait on 1, lülitatakse pistik sisse. Kui viimane bait on 9 lülitatakse pistik välja. Kui viimane bait on 8 annab pistik tagasi oma hetkeseisu, kas sees või väljas. Kui paketid on välja saadetud jäädakse ootama ACK-i. Kui ACK-i ei ilmu, siis korratakse tegevust ning kui siis ei saada ACK-i vastu, tõstatakse viga, mida hiljem veebiliides kasutajale kuvab.

Käesoleva töö nutipistikuga info vahetamiseks on võimalik kasutada ka krüpteeritud infovahetust. Selleks krüpteeritakse pakettide sõnumiosa 128-bitise täiustatud krüpteerimisstandardi võtmega. Sama krüpteerimisviisi kasutatakse täna näiteks ruuterite paroolide krüpteerimiseks (WPA2). Selleks lisatakse üks bait saadetud paketti, et vastuvõtja dekrüpteeriks sõnumi, enne kui seda kasutama saab hakata.

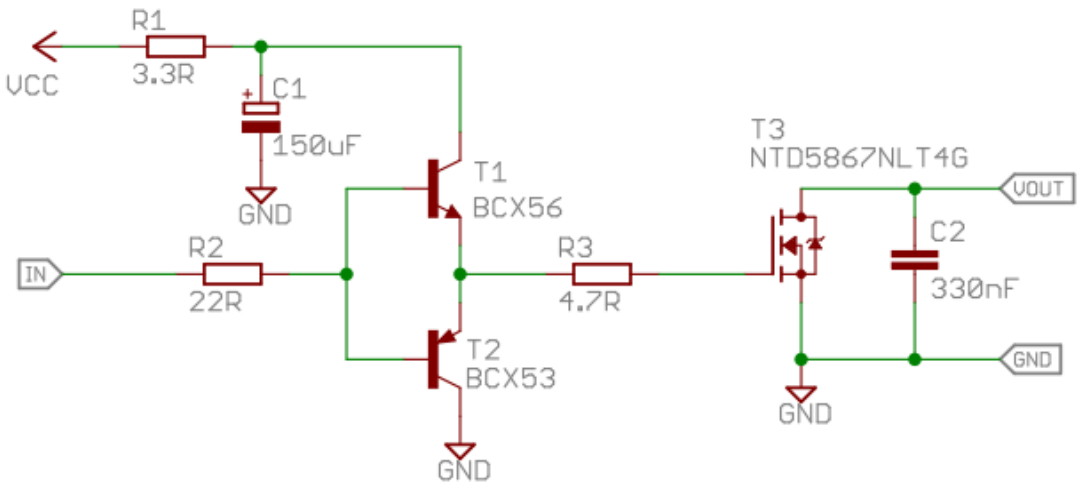
Tabel 2 Käsud nutipistikule

Käsk	**Bait**
Nutipistik sisse	0x01
Nutipistik välja	0x09
Anna nutipistiku seis	0x08

iv. MOSFET väljatransistor ja relee

Käesolevas töös otsustati relee ise ehitamise asemel võtta kasutusele olemasolev releemoodul *MOSFET NTD5867 Stamp* [14], mis koosneb kahest n-tüüpi isoleeritud paisuga väljatransistorist (*MOSFET*), kahest kondensaatorist, kolmest takistist ja releest endist. Lisaks otsustati, et käesoleva töö nutipistiku implementatsioon viiakse läbi 12V piires, et suurendada autori ohutust.

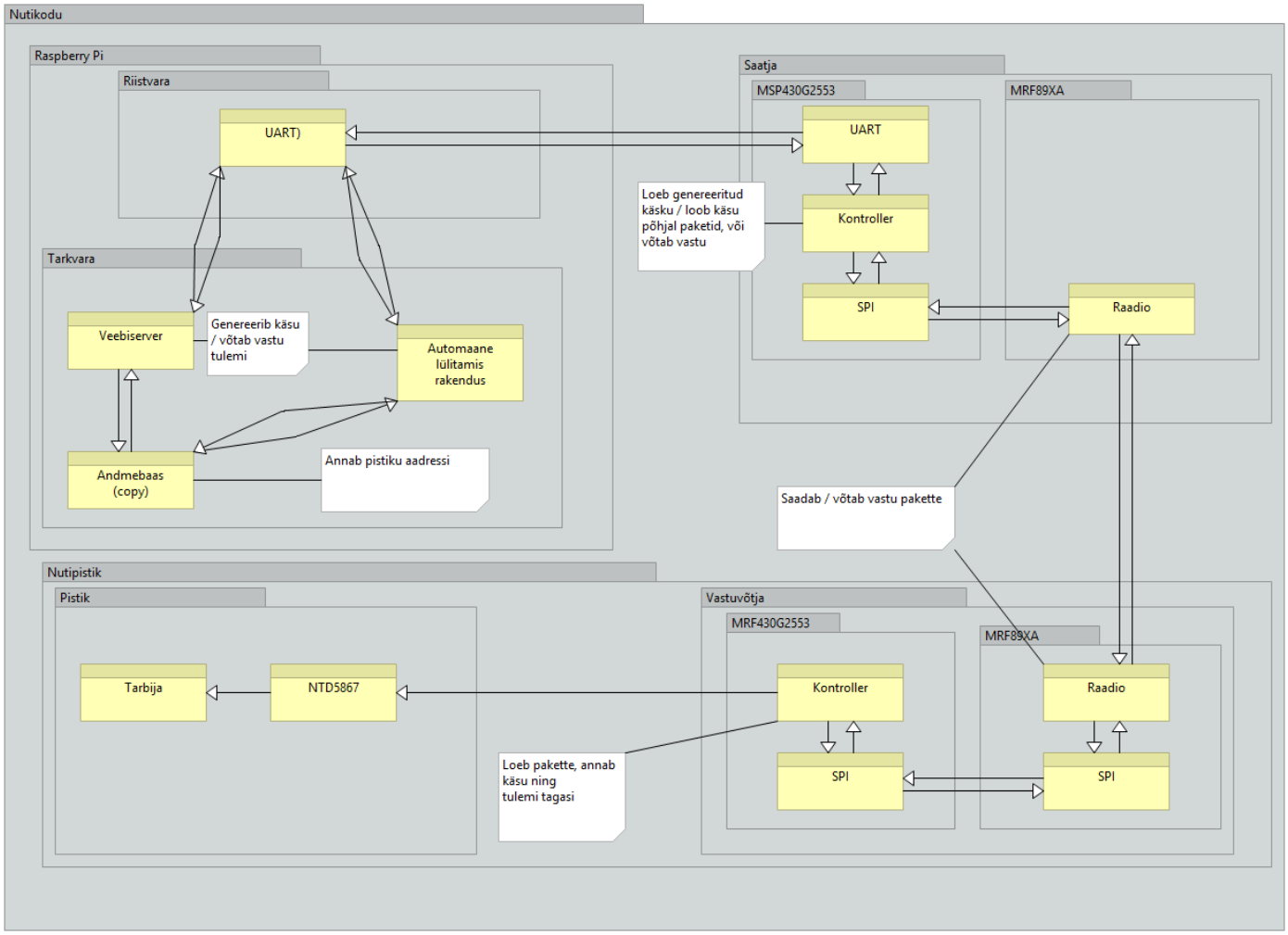
Releemoodul võtab toite ja maanduse MSP430 pealt, lisaks ühendatakse üks kaabel lülituseks. Lülitus hakkab toimima ja 3.3V piires sisuliselt diskreetselt: kui kõrgem pinge eksisteerib, on pistik sees ja vastupidi.



Joonis 6 MOSFET NTD5867 Stamp skeem [14]

Käesolevas töös on kõik ühendused nutipistiku töö osas vajalikud. See tähendab, et kui üks ühendus puudub, siis nutipistiku töötamine ei ole tagatud. Kõige keerulisem ühendustest on pakettide saatmine kahe mikrokontrolleri vahel, kus tuleb anda kasutajale

vastus ka siis kui ühendust üldse ei saavutatud. Iga ühenduse vahel rikkeid avastades tuleb neid ka kasutajale kuvada, kuid see on juba tarkvaraline realisatsioon. Erinevate riistvara komponentide suhtlus on kirjeldatud joonisel 7, kust on hästi näha, et ühe komponendiga suhtluse ebaõnnestumise korral, enam nutilülitust tagada ei saa



Joonis 7 Riistvara ülesehitus

5. Tarkvara arhitektuur

5.1. Andmebaasi disain

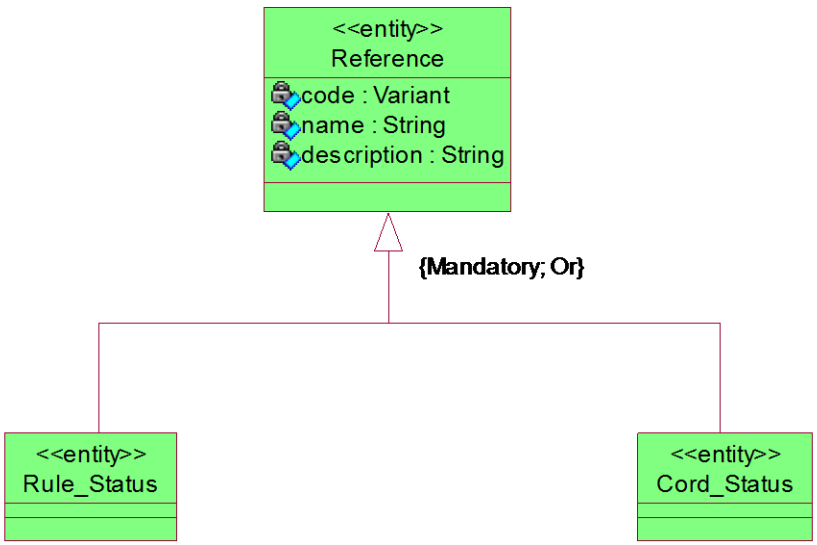
Käesoleva töö rakendus baseerub *MySQL* [15] andmebaasil. Seega on kõik järgnevad kirjeldused ja skeemid *MySQL*-il põhinevad.

v. Olemisuhte diagrammid

Käesolev töö omab nutipistiku lülitamisel kahte võimalust.

1. Manuaalselt lülitamine, kus kasutaja ise sekkub nutipistiku lülitamisse.
2. Automaatne lülitamine, kus nutipistik lülitatakse, kas sisse või välja kui antud hetke aeg väljub andmebaasis defineeritud vahemikust, või kui lülitatakse välja konkreetse pistiku ülempistik, mis on andmebaasis defineeritud seos.

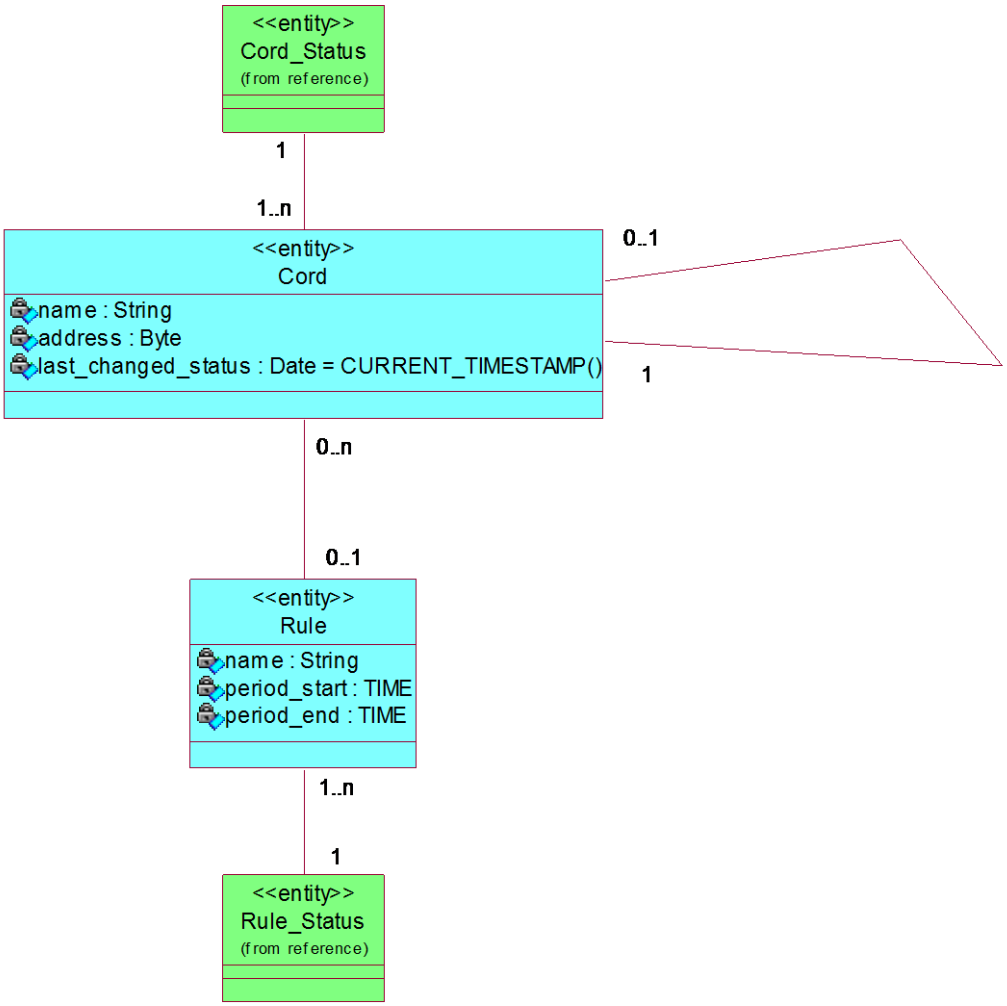
Andmebaasi vajadus tekkis pigem vajadusest nr. 2, sest nii on rakendusel kergem ajavahemikke ja seoseid võrrelda. Nii defineeriti andmebaasis lisaks olemasolevatele olemitele uued koos omavaheliste seostega. Uute olemite seas disainiti ka lihtne klassifikaatorite register, mida varasemal mudelil ei eksisteerinud, kuna oli vaja defineerida pistikute staatusi kui ka automaatse lülitamise reeglite staatusi, et nende üle paremat kontrolli hoida.



Joonis 8 Klassifikaatorite registri olemisuhte diagramm

Järgnevalt defineeriti põhiolemid nutipistiku lülitamiseks ehk defineeriti kaks olemit - nutipistiku enda andmestiku olem *Cord* ja nutipistiku reeglite andmestiku olem *Rule*. Nutipistiku osas arvestati, et see peab omama nutipistiku nime, mida kasutajale veebiliidese kaudu kuvatakse, konkreetse pistiku aadressi ning viimast manuaalse lülituse kellaaega. Lisaks otsustati, et reegli ajavahemiku vahel väärtustatav lülitus võetakse pistiku olemi küljest, ehk seos pistiku staatusega luuakse koos pistikuga. Lisaks võib eksisteerida pistikul seos mingi reegliga, aga see ei ole kohustuslik. Viimaseks tekitati rekursiivne seos pistiku olemi endaga, et lülitusel ka alampistikuid lülitada.

Reeglites defineeriti sarnaselt pistiku andmestikuga nimi, mida kuvatakse kasutajale, ajavahemiku algus ja lõpp. Seejärel defineeriti seosed reeglite staatuste klassifikaatori ning pistiku olemiga.

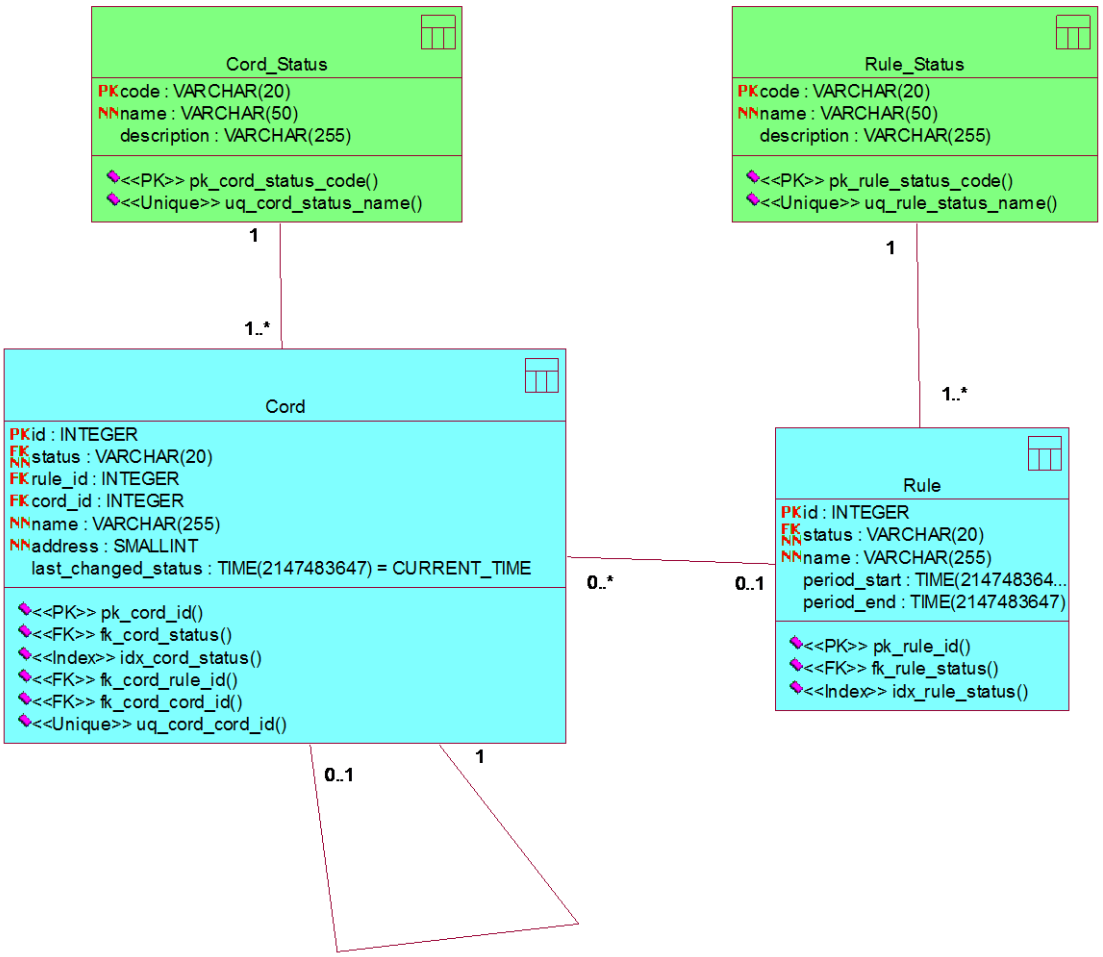


Joonis 9 Nutipistiku all-süsteemi olemisuhte diagramm

vi. Andmebaasi diagramm

Andmebaasi diagrammi koostamisel valiti ära andmeveergude tüübid ja seoste tüübid. Klassifikaatorite seoseid hakatakse esitama koodi alusel, mis on maksimum 20 tähemärki pikk. Lisaks omavad klassifikaatorid maksimum 50 tähemärgist nimetust ja maksimaalselt 255 tähemärgilist kirjeldust.

Reeglite osas otsustati, et aja veerud saavad olema tüüpi *TIME*. Pistiku osas, et aadress on *SMALLINT* tüüpi ja viimane manuaalselt muudetud aeg ka *TIME* tüüpi. Pistiku tabeli ja reeglite tabeli nimed saavad tüübiks *VARCHAR* pikkusega 255 ning omavahelised seosed genereeritakse *id* välja järgi. Kõik indeksid ja kitsendused hakkavad kindla prefiksiga, mis on tuletatud inglise keelsetest vastetest ja millele järgneb antud tabeli nimi ning veeru nimi eraldatuna alakriipsude abil.



Joonis 10 Nutipistiku all-süsteemi andmebaasi diagramm

vii. Klassifikaatorite väärtused

Käesolevas töös kirjeldati pistiku ja reeglite olemite jaoks vastavalt kolm ning kaks staatust. Selleks tuli sisestada klassifikaatoritesse kindlad väärtused. Pistikute osas sisestati klassifikaatorid koodi väärtustega *CORD_ON*, *CORD_OFF* ning *CORD_DISABLED*, mis tähendavad vastavalt pistik sees, väljas ja mitteaktiivne. Reeglite olemi klassifikaatorisse sisestati väärtused *RULE_ACTIVE* ja *RULE_DISABLED*, mis tähendavad vastavalt, et reegel on aktiivne ning reegel ei ole aktiivne.

```
INSERT INTO Cord_Status (code, name, description)
     VALUES
           ('CORD_ON',
           'Cord On',
           'Smart Cord will be switched on'),
           ('CORD_OFF',
           'Cord Off',
           'Smart Cord will be switched off'),
           ('CORD_DISABLED',
           'Cord Disabled',
           'Smart Cord is not enabled for switching');

INSERT INTO Rule_Status (code, name, description)
     VALUES
           ('RULE_ACTIVE',
           'Rule Active',
           'Rule will be accounted in automation'),
           ('RULE_DISABLED',
           'Rule Disabled',
           'Rule will not be accounted in automation');
```

Joonis 11 Klassifikaatorite sisestus

5.2. Veebirakenduse disain

Käesoleva töö veebirakenduse tagarakend [16] on realiseeritud *Python*-is [17]. Tagarakendi põhi kasutab *python bottle* [18] mikro-raamistikku vastu võtmaks *HTTP* [19] päringuid veebiliidese poolt. Lisaks *HTTP* päringutele lubab antud raamistik veebiliidest disainida ja ehitada ka *python*-i koodi abil sarnaselt *Java JSP*-le [20]. Samuti lubab raamistik ehitada rakendust ja üles panna nii *WSGI* [21] liidese abil mingil veebiserveril nagu *Apache2* [22] *või nginx* [23], kui ka raamistiku enda ehitatud veebiserveri abil, mis põhineb samuti *WSGI*-l. Et *WSGI*-ga liidestus lihtsamaks teha ning nii nutipistiku

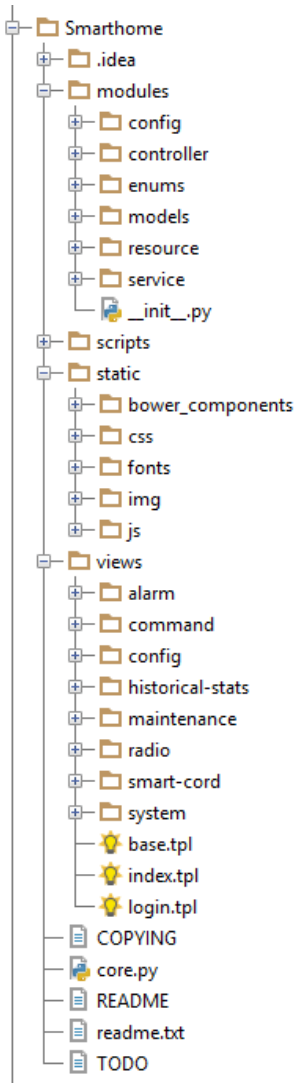
mooduli lisamise kui ka mistahes uue lisa lisamise tõhusust parandada oli käesoleva töö tagarakendi põhja vaja umber disainida.

Arhitektuuri muudatus. Käesoleva töö põhja tagarendi osa *HTTP* päringute vastuvõtmise loogika oli esialgselt kirjutatud kõik ühte faili *core.py*, mis oli üsna mittehallatav ja pikk. Antud fail sõltus (kui välja arvata kõik veebiliidese kuva failid) failidest nimega *storage.py,* mis mingis osas haldas andmebaasi päringud, *config.py*, kus defineeriti rakenduse seadistus, *helpers.py,* kus realiseeriti kogu rakenduse äriloogika ning *alarm.py*, kus eksisteeris alarmi mooduli disain. See, aga teeb keeruliseks uute moodulise lisamise, installimise antud põhjale, kus tuleks iga uue mooduli juhul *core.py*-le uued read juurde liita. See teeks struktuuri veelgi segasemaks ning vajadus uue järgi oli möödapääsmatu.

Probleemi lahendamiseks otsustati põhja struktuur ümberdisainida ehitades üles erinevad kihid kindlate ülesannetega. Nii defineeriti *HTTP* päringute kiht (*controller)*, teenuste kiht (*service)*, kus toimub nii andmebaasi andmete, kui ka päringute töötlemine ning andmebaasi päringute kiht (*resource).* Igat kihti iseloomustab *python*-i pakett, kuhu on koondatud erinevad moodulid *python* failide näol. Lisaks defineeriti paketid: *config* – konfiguratsiooni pakett, *models* – olemite kirjeldus tagarakendis ning *enums* – konstantide definitsioon.

Seejärel hakati esialgse põhja faile ringi jagama ning tükeldama. Fail *core.py HTTP* päringud ehitati vastavaks mooduliks, failiks ning tõsteti *controller* paketti, *config* suunati oma konfiguratsiooni paketti ning *python-i* failid *helpers.py, alarm.py* ja *storage.py* suunati service kausta. Põhi fail *core.py*-sse jäeti alles ainult rakenduse käivitamise loogika ning konfiguratsioon importimine. Kõikide *HTTP* päringute moodulite importimine jäeti konfiguratsiooni faili *config.py*-le defineerimiseks ja realisatsiooniks.

Samamoodi tükeldati ümber ka veebiliidese vaated, kus iga vaade sai oma päringule vaastava mooduli kausta. Varasemalt olid kõik vaated ühes kaustas nimega *views*, nüüdsest *views/{MOODULI_NIMI}* ja vastavad vaated nagu näidatud joonisel 12.



Joonis 12 Ümber disainitud pakettide struktuur

viii. Uute moodulite lisamine

Arhitektuuri muudatus viidi läbi, et lihtsustada koodi uute moodulite lisamist ning parandada koodi loetavust, lähtudes *clean-code* [24] põhimõtetest. Uue mooduli lisamiseks uue arhitektuuri puhul ei pea enam ühtegi faili muutma või kuskil lisa defineerima. Tuleb ainult oma mooduli *HTTP* päringute koodifailid, milles on defineeritud vajalikud teenusekihi impordid, paigutada *controller* paketti ning *config.py* automaatselt seadistab ka need päringud vastuvõtmiseks. Tuleb ainult jälgida, et uued failid kasutaksid *python bottle* mikro-raamistikku ning serverile tuleb teha restart.

5.3. Veebirakenduse raamistikud

Nagu varem mainitud, kasutab käesoleva töö rakenduse põhi tagarakendis *python bottle* mikro-raamistikku. Antud raamistik on väga lihtne ja töö riistvarale hästi sobiv, sest see ei ole väga nõudlik ressursside osas. Veebiliides kasutab disaini raamistikuna puhast *bootstrap-i*, mis pakub ekraanisuurusele reageerivat välimust, erinevaid lisafunktsioone nagu näiteks *modal* ning näeb hea välja ka mobiilist vaadatuna. *Bootstrap* on valinud töötamiseks *javascript-i* raamistikku nimega *jQuery* [25], mis on ka rakenduse põhjal standardkasutuses.

5.4. Veebirakendusest nutipistiku lülitus

Nagu varem mainitud saab veebirakendusest kasutaja reguleerida lülitusi 3-el viisil: automaatselt, manuaalselt ja ajalise reegli põhjal.

Manuaalne lülitus. Kasutajale kuvatakse tabel nutipistikutest. Tabeli rida koosneb nutipistiku numbrist, nimest, aadressist, võimalikust reegli nimest ja hetke staatusest. Hetke staatuse vajutamisel vahetatakse nutipistiku staatust ning kirjutatakse andmebaasi viimane staatuse muudatuse kella aeg.

Automaatne lülitus. Automaatne lülitus käivitatakse siis, kui mingil hetkel lülitataval pistikul on andmebaasis defineeritud alampistikuid. Käesolev töö lubab kasutajaliideses defineerida kahe tasemelisi alampistikuid, ehk üks ülempistik ja mitu alampistikut, kuid alampistikutel enam oma alampistikuid olla ei lubata.

Ajalisel reeglil põhinev lülitus. Et kasutada automaatset lülitust, defineeriti andmebaasis reeglite tabel. Reegleid saab defineerida veebiliideses, kus defineeritakse reegli algus ja reegli lõpp. Reegli ajavahemikus lülitatakse pistik sellesse staatusesse, mis on määratud pistiku tabelis.

Selleks, et käivitada lülitusi, defineeriti veebirakenduses konstant (*enum*-ina*) Cord_Status*. Vastav konstant omab nelja väärtust – sees, väljas, mitteaktiivne ja staatuse küsimine. Esimesed kolm on defineeritud samal eesmärgil, mis andmebaasiski, viimane on loodud küsimaks pistikult endalt hetkestaatust.

Staatuse muutmisel, kui tegemist ei ole ajalise lülitusega, esitatakse päring:

```
SELECT main.address AS main_addr, sub.address as sub_addr
FROM Cord AS main
  LEFT JOIN Cord AS sub
    ON main.id = sub.cord_id
WHERE main.id = ?;
```

Joonis 13 Veebirakenduse päring lülitusel pistiku andmete saamiseks

Kus ? asendatakse lülitatava pistiku id-ga.

Päringus tekkiva *main_addr* veerus hoitakse ülempistiku aadressi, veerus *sub_addr* alampistiku aadressi. Ülempistiku aadress saadetakse lülitusele ning iga ülempistikuga seotud alampistiku aadress samuti.

Nagu varem mainitud, kasutab *Raspberry Pi* ja MSP430 omavaheliseks vestluseks *UART* protokolli. Selleks kasutatakse veebirakenduse tasandil *python-i* moodulit *serial*.

```
import serial as SerialLib


def generate_cord_request(cord_address, request):
    value = request.value + "|" + str(cord_address) + "-"
    if request is CordStatus.ON:
        return value
    elif request is CordStatus.OFF:
        return value
    elif request is CordStatus.STATUS:
        return value
    else:
        raise "Invalid request"


def set_cord_status(status, addr):
    if status != CordStatus.ON and status !=
CordStatus.OFF:
        raise "Invalid status!"
    else:
        serial = SerialLib.Serial("/dev/ttyAMA0", 9600)
        command = generate_cord_request(addr, status)
        serial.write(command)
```

Joonis 14 Veebirakenduse päring pistikule käsu andmiseks või staatuse küsimiseks Pythonis

Käsu saatmiseks genereeritakse päring, millest ka mikrokontroller aru saab. Seega saadetakse väärtustatav staatus ja pistiku aadress ühise tekstina, kus püstkriips on eraldavaks ning viimaseks tähemärgiks määratakse sidekriips. Mikrokontroller suudab juba saadetud tekstist vajaliku info paketina välja saata.

5.5. Staatuste küsimine

Igas veebisessiooni alguses küsitakse pistikute staatusi, mis ei ole staatuses mitteaktiivne. Pärast esmast küsimist korratakse staatuste uuendamist iga poole minuti tagant saates *AJAX* [26] päringu veebiserverile.

Selleks esitatakse päring:

```
SELECT * FROM Cord WHERE status <> 'CORD_DISABLED';
```

Joonis 15 Aktiivsete pistikute leidmise päring

Seejärel küsitakse üle *UART*-i pistikute staatusi ning info tagastatakse veebiliidesele.

5.6. Ajapõhise lülitusrakenduse disain

Ajapõhine rakendus on ka *python* rakendus, mis kasutab samu teenuseid, mis veebirakenduski, kuid põhjaks on teistsugune päring:

```
SELECT main.address AS main_addr, main.status,
sub.address as sub_addr, Rule.period_start,
Rule.period_end
FROM Cord AS main
  INNER JOIN Rule ON Rule.id = main.rule_id
  LEFT JOIN Cord AS sub ON main.id = sub.cord_id
WHERE main.status <> 'CORD_DISABLED'
      AND Rule.status <> 'RULE_DISABLED'
      AND main.last_changed_status
        NOT BETWEEN Rule.period_start AND
Rule.period_end;
```

Joonis 16 Ajapõhise lülituse päring

Päring tagastab kõik nutipistikute aadressid, millel võib aga ei pea olema alampistik, alampistiku aadress, kus on ülempistikul on reegel ning viimane lülitus ei ole reegli ajavahemiku vahel.

Käesolev töö lülitab kõik alampistikud automaatsel lülitusel samasugusesse staatusesse nagu on ülempistik.

Automaatne lülitus käivitatakse kui *linux-i crontab* [27] töö, mida iga kasutaja peab manuaalselt seadistama.

5.7. Saatja disain

Nii saatja, kui ka vastuvõtja arendamisel on põhjaks võetud Arvutitehnika teaduskonna MSP430 + MRF89XA näiterakendus [11]. Nii saatja, kui ka vastuvõtja on rakendusena üks projekt, kus kasutatakse vastava mikrokontrollerile mõeldud loogikat seadistatuna konfiguratsiooni failis.

UART sõnum mikrokontrolleris. Sõnumit vastu võttes lähtub MSP430 samasugusest sõnumi ehitusest nagu on tagarakendis defineeritud. Antud sõnumis on käsk nutipistikule üks tähemärk pikk. Kui täht on *a* lülitatakse nutipistik sisse, *b* välja ning *s*-i puhul küsitakse nutipistiku staatust.

Sõnumit vastu võttes käivitatakse iga tähemärgi kohta mikrokontrolleris *__interrupt* funktsioon, kus tähemärk kirjutatakse massiivi. Antud massiivi rakenduse poolt pidevalt uuritakse, lõpmatus tsüklis, et kas lõppmärk on saabunud. Kui on, loetakse sõnum ning hakatakse vastavat käsku täitma.

```
uint8 UART_Receive_Data(char *uart_data_buf) {
   uint8 i, k;
   int8 j;
   if (uart_buf_size > 0) {
         for (i = 0; i < MAX_UART_STR_LENGTH; i++) {
         // Search for the end of packet
               if (uart_rcv_buf[i] == UART_END_CHAR) {
                     // If end string found then make buffer empty
                     for (k = 0; k < MAX_UART_STR_LENGTH; k++) {
                           uart_data_buf[k]='\0';
                     }

                     // Copy received data byte into new array
                     for (j = i-1; j >= 0; j--) {
                           // "j=i-1" deletes
                           //last char; "j>=0" includes first
                           // "dataBuf" get the value of string
                           uart_data_buf[j] = uart_rcv_buf[j];
                     }

                     // Reset buffer
                     for (uart_buf_size = 0; uart_buf_size <
                           MAX_UART_STR_LENGTH; uart_buf_size++)
                           uart_rcv_buf[uart_buf_size]='\0';

                     uart_buf_size = 0;
                     return 1;
               }
         }
         return 0;
   }
   return 0;    // Exit with 0 when no byte received
}

#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void) {
   uart_rcv_buf[uart_buf_size++] = UCA0RXBUF;
}
```

Joonis 17 Sõnumi vastuvõtmine kontrolleris

ix. Aadressi ja käsu teisendamine sõnumist ja pakettide saatmine

Käsu ja aadressi sõnumist kättesaamisel lahutatakse vastuvõetud sõnum püstkriipsust. Esimene pool sõnumist on käsk, teine pool pistiku aadress. Mõlemad salvestatakse maha massiivi, kust neid hiljem loetakse (Joonis 18).

```
#if (TRANSMITTER)
void parse_message(char request[]) {
	int i = 0;
	char *p;
	p = strtok(request, "|");
	while (p != NULL) {
		request_elems[i++] = p;
		p = strtok(NULL, "|");
	}
}
#endif
```

Joonis 18 Sõnumi teisendamine kontrolleris

Joonisel 19 on *TxPacket* sõnum, mis saadetakse üle raadiovõrgu teele. Paketi viimane liige on vastav bait, mida vastuvõtja loeb. Antud funktsioon tagastab tüübi *unsigned char*, millega saab kontrollida funktsiooni käivitumist. Parameetri *error* abil uuritakse koodis kas on kätte saadud *ACK*. Kui jah, saadetakse serverile tagasi vastus *200*, mis tähendab, et päring õnnestus. Teisel juhul on saadetakse päring uuesti ja kui ka siis ei tule vastust, antakse serverile *404*, ehk pistikut ei leitud.

```
Radio_Send_Data(TxPacket, payload_length,
addr,PAYLOAD_ENC_OFF, PCKT_ACK_ON, &error);
```

Joonis 19 Pakettide saatmise funktsiooni väljakutse

6. Vastuvõtja disain

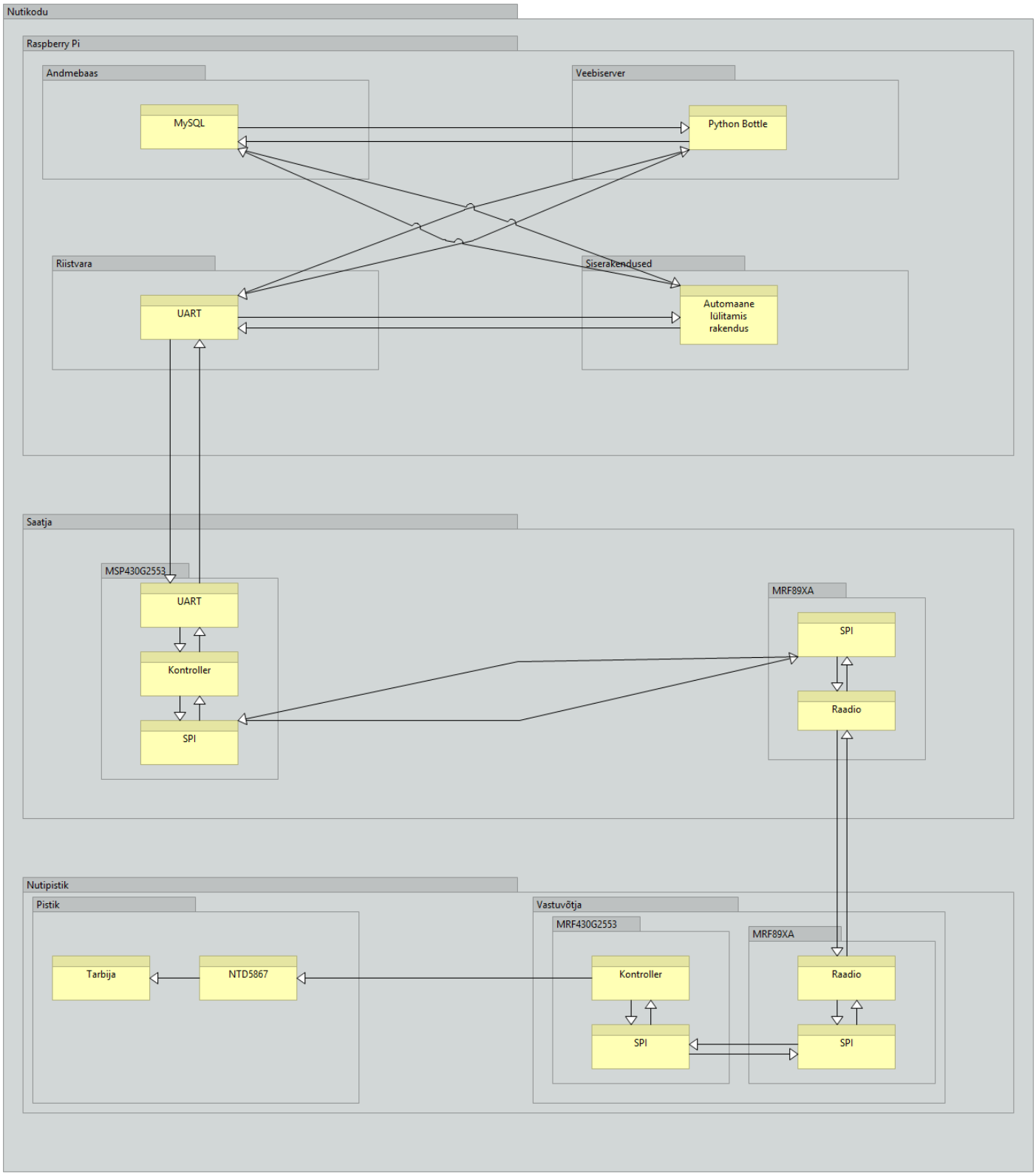
Vastuvõtja disain on üsna lihtne. Kontrollitakse paketi viimast väärtust ja vastavalt käsule kas lülitatakse või mitte. Käskude uurimisel võetakse täitmisele ainult need käsud, mille vastuvõetud paketi viimane väärtus on

```
uint8 command = RxPacket[len - 1];
if(command == cord_on) {
	CORD_ON();
} else if (command == cord_off){
	CORD_OFF();
} else if (command == cord_status) {
	TxPacket[payload_length++] = CORD_STATUS();
	Radio_Send_Data(TxPacket, payload_length,
		ADDR_LOCAL,PAYLOAD_ENC_OFF, PCKT_ACK_ON,
			&error);
	Radio_Set_Mode(RADIO_RX);
}
```

Joonis 20 Käsu kontroll vastuvõtjas

Ainus keerukus tekib staatuse küsimisel, kus vastuvõtja lülitatakse hetkeks saatmis režiimi saates ära pistiku staatuse ning pärast seda lülitab end tagasi. Suurim probleem on eemal seisva pistikuga see kui sellega üldse ühendust ei saa, tuleb probleemi tuvastamiseks siiski vastuvõtjat lähemalt uurida.

Tarkvara on tihedas seoses riistvaraga. Suhtlus nende vahel on pidev ning riistvarata ei saaks olla nutilülitust ja vastupidi. Nagu varem mainitud, on antud disainis oluline, et kõik komponendid oleksid hästi ühilduvad, sest sõltuvused nende vahel on tugevad. Joonis 21 kirjeldab riistvara ja tarkvara komponentide omavahelist suhtlust. Kirjeldatakse nii riistvara komponentide kui ka tarkvara komponentide suhtlust omavahel.



Joonis 21 Nutipistiku ülesehitus

7. Kokkuvõte

Käesoleva töö eesmärgiks oli tutvuda riistvaraliste, tarkvaraliste ning andmebaasi lahenduste sümbioosiga ning selle näitel disainida nutipistik, millele on ehitatud lihtne veebiliides. Lisaks oli eesmärk parandada TTÜ Arvutitehnika instituudi nutikodu rakenduse põhja tarkvaralist lahendust, et võimaldada paremat hallatavust.

Töö oluliseks tulemuseks oli võimalus lülitada eemal seisvat kontrolleriga seotud releed, mida käesolevas töös nimetatakse nutipistikuks, sisse ja välja. Lisaks on oluline nutipistiku lülitust võimaldav liides ning *linux-i crontab-i* abil käivitatav automaatne lülitus kindla reeglistiku alusel.

Juhtmevaba nutipistiku olemus on iseenesest lihtne, kuid realisatsioonini jõudmine tekitab sageli keerulisi probleeme, mille lahendamiseks kulub aega. Samas kui lahenduseni korra jõutud, on tulemus hästi töötav. Lisaks tuleb tõdeda, et hästi tehnikat tundes, on võimalik üksikisiku elu mugavamaks teha, automatiseerida ning selle tulemusena ka rahaliselt kokku hoida.

Käesolevat tööd on võimalik edasi arendada päris mitmel viisil. Kui lülituse ehitus on juba disainitud, on võimalik ehitada veelgi keerulisemaid reeglite disaine. Näiteks lisaks ajalisele reeglile võib juurde liita temperatuurist või liikumisandurist tuleneva info põhjal täienduse. Lisaks võib veel veebiliidest parandada liigutades mõned lahendused hoopis veebilehitseja poolt hallatavaks, ehk luues näiteks ühel lehel põhineva rakenduse, mille lehtede vahel liikumine realiseeritakse *javascript*-i poolt. See päästaks serveri osal oluliselt ressursi vabaks.

Käesolevas töös püstitatud eesmärgid saavutati ja nutilülitus on valmis ehitatud. Lisaks disainiti ümber olemasolev põhi tulevasteks arendusteks ning ehitati näitena uue mooduli liidestus, milleks oli nutipistik.

8. Summary

The main purpose of this thesis was to create a wireless smart cord that can be switched in multiple ways. To follow that purpose through one should be familiar how software, hardware and database solutions could work together and that is why writing this thesis required some research to begin. Another purpose was to improve previous software base to become more maintainable and purer.

The result was a switchable wireless smart cord as a relay that managed from distance via web interface or an automatic switcher that was based on certain time rules.

The design of the smart cord is easy, but from zero to production there were some difficulties in creating links with different areas. But when solutions came, they worked well.

The result was successful and switchable smart cord was created. Also the base of the software that was provided, is improved and has now a pure package based architecture. The automated rule based application was created as an example of home automation. This thesis provided a good base for future developments.

Kasutatud kirjandus

[1] „Cvuorinen.net,“ [Võrgumaterjal]. Available: http://cvuorinen.net/2014/04/what-is-clean-code-and-why-should-you-care/. [Kasutatud 1 May 2015].

[2] „Raspberry Pi,“ [Võrgumaterjal]. Available: http://www.tcpipguide.com/free/t_DataLinkLayerLayer2.htm. [Kasutatud 22 May 2015].

[3] „Bsc.ee,“ [Võrgumaterjal]. Available: http://www.bcs.ee/uudis-kaks-erinevat-laehenemist-tarkvaraarendusele. [Kasutatud 22 May 2015].

[4] „Smart home energy,“ [Võrgumaterjal]. Available: http://smarthomeenergy.co.uk/what-smart-home. [Kasutatud 22 May 2015].

[5] „Indome.ee Fibaro,“ [Võrgumaterjal]. Available: http://www.indome.ee/osta/juhtimiskeskused/fibaro-home-center-lite.html. [Kasutatud 22 May 2015].

[6] „WhatIs.com,“ [Võrgumaterjal]. Available: http://whatis.techtarget.com/definition/UART-Universal-A. [Kasutatud 22 May 2015].

[7] „Texas instruments,“ [Võrgumaterjal]. Available: http://www.ti.com/tool/msp-exp430g2. [Kasutatud 22 May 2015].

[8] „Microchip,“ [Võrgumaterjal]. Available: http://www.microchip.com/wwwproducts/Devices.aspx?product=MRF89XA. [Kasutatud 22 May 2015].

[9] „Raspberry GPIO,“ [Võrgumaterjal]. Available: https://www.raspberrypi.org/documentation/usage/gpio/. [Kasutatud 22 April 2015].

[10] „Arduino SPI,“ [Võrgumaterjal]. Available: http://www.arduino.cc/en/Reference/SPI. [Kasutatud 24 May 2015].

[11] M. Leier ja M. Gorev, „ATI Embsys TTÜ,“ [Võrgumaterjal]. Available: http://ati.ttu.ee/embsys/index.php/868MHz_wireless_connection. [Kasutatud 16 April 2015].

[12] „Techtarget,“ [Võrgumaterjal]. Available: http://searchnetworking.techtarget.com/definition/Network-layer. [Kasutatud 22 May 2015].

[13] „TCP ip guide,“ [Võrgumaterjal]. Available: http://www.tcpipguide.com/free/t_DataLinkLayerLayer2.htm. [Kasutatud 22 May 2015].

[14] „Break my board,“ 22 May 2015. [Võrgumaterjal]. Available: http://www.breakmyboard.com/product/mosfet-n-channel-stamp/), . [Kasutatud 22 May 2015].

[15] „MySQL,“ Oracle Corporation, [Võrgumaterjal]. Available: https://www.mysql.com/. [Kasutatud 22 May 2015].

[16] „EKI,“ [Võrgumaterjal]. Available: http://portaal.eki.ee/uuedsonad/Uued-s%C3%B5nad-1/T/tagarakend-139/. [Kasutatud 2015 May 19].

[17] „Python,“ [Võrgumaterjal]. Available: https://www.python.org/. [Kasutatud 22 May 2015].

[18] „Python Bottle,“ [Võrgumaterjal]. Available: http://bottlepy.org/docs/dev/index.html. [Kasutatud 22 May 2015].

[19] „Computer Hope,“ [Võrgumaterjal]. Available: http://www.computerhope.com/jargon/h/http.htm. [Kasutatud 19 May 2015].

[20] „Oracle Java EE,“ [Võrgumaterjal]. Available: http://docs.oracle.com/javaee/5/tutorial/doc/bnagy.html. [Kasutatud 10 May 2015].

[21] „WSGI,“ [Võrgumaterjal]. Available: http://wsgi.readthedocs.org/en/latest/what.html. [Kasutatud 10 May 2015].

[22] „Apache.org,“ [Võrgumaterjal]. Available: http://httpd.apache.org/. [Kasutatud 5 May 2015].

[23] „Nginx,“ [Võrgumaterjal]. Available: http://wiki.nginx.org/Main. [Kasutatud 10 May 2015].

[24] „Pluralsight,“ [Võrgumaterjal]. Available: http://www.pluralsight.com/courses/writing-clean-code-humans. [Kasutatud 2015 May 19].

[25] „jQuery,“ [Võrgumaterjal]. Available: https://jquery.com/. [Kasutatud 22 May 2015].

[26] „W3 schools,“ [Võrgumaterjal]. Available: http://www.w3schools.com/ajax/. [Kasutatud 18 May 2015].

[27] „Crontab,“ [Võrgumaterjal]. Available: http://kvz.io/blog/2007/07/29/schedule-tasks-on-linux-using-crontab/. [Kasutatud 22 May 2015].

[28] J. J. I. B. G. Rumbaugh, The Unified Modeling Language User Guide, Addison-Wesley, 2005.

Lisa 1 – Andmebaasi loomise laused koos klassifikaatorite lisamisega

```
CREATE TABLE Rule (
  id INT NOT NULL AUTO_INCREMENT,
  status VARCHAR ( 20 ) NOT NULL,
  name VARCHAR ( 255 ) NOT NULL,
  period_start TIME ( 2147483647 ),
  period_end TIME ( 2147483647 ),
  CONSTRAINT pk_rule_id PRIMARY KEY (id)
);


CREATE INDEX idx_rule_status ON Rule (status );
CREATE TABLE Cord (
  id INT NOT NULL AUTO_INCREMENT,
  status VARCHAR ( 20 ) NOT NULL,
  rule_id INT,
  cord_id INT,
  name VARCHAR ( 255 ) NOT NULL,
  address SMALLINT NOT NULL,
  last_changed_status TIME ( 2147483647 ) DEFAULT
CURRENT_TIME(),
  CONSTRAINT pk_cord_id PRIMARY KEY (id),
  CONSTRAINT uq_cord_cord_id UNIQUE (cord_id)
);
CREATE INDEX idx_cord_status ON Cord (status );
CREATE TABLE Cord_Status (
  code VARCHAR ( 20 ) NOT NULL,
  name VARCHAR ( 50 ) NOT NULL,
  description VARCHAR ( 255 ),
  CONSTRAINT pk_cord_status_code PRIMARY KEY (code),
  CONSTRAINT uq_cord_status_name UNIQUE (name)
);
CREATE TABLE Rule_Status (
  code VARCHAR ( 20 ) NOT NULL,
  name VARCHAR ( 50 ) NOT NULL,
  description VARCHAR ( 255 ),
  CONSTRAINT uq_rule_status_name UNIQUE (name),
  CONSTRAINT pk_rule_status_code PRIMARY KEY (code)
);
```

```
ALTER TABLE Cord ADD CONSTRAINT fk_cord_status
FOREIGN KEY (status)
  REFERENCES Cord_Status (code)
    ON DELETE NO ACTION ON UPDATE NO ACTION;
ALTER TABLE Cord ADD CONSTRAINT fk_cord_cord_id
FOREIGN KEY (cord_id)
  REFERENCES Cord (id)
    ON DELETE NO ACTION ON UPDATE NO ACTION;
ALTER TABLE Cord ADD CONSTRAINT fk_cord_rule_id
FOREIGN KEY (rule_id)
  REFERENCES Rule (id)
    ON DELETE NO ACTION ON UPDATE NO ACTION;
ALTER TABLE Rule ADD CONSTRAINT fk_rule_status
FOREIGN KEY (status)
  REFERENCES Rule_Status (code)
    ON DELETE NO ACTION ON UPDATE NO ACTION;


INSERT INTO Cord_Status (code, name, description)
VALUES ('CORD_ON',
        'Cord On',
        'Smart Cord will be switched on'),

      ('CORD_OFF',
       'Cord Off',
       'Smart Cord will be switched off'),

      ('CORD_DISABLED',
       'Cord Disabled',
       'Smart Cord is not enabled for switching');

INSERT INTO Rule_Status (code, name, description)
VALUES ('RULE_ACTIVE',
        'Rule Active',
        'Rule will be accounted in automation'),

      ('RULE_DISABLED',
      'Rule Disabled',
      'Rule will not be accounted in automation');
```