Tartu Ülikool

Loodus- ja täppisteaduste valdkond

Tehnoloogiainstituut

Meelis Pihlap

Mitme roboti koostöö funktsionaalsuste väljatöötamine tarkvararaamistikule TeMoto

Bakalaureusetöö (12 EAP)

Arvutitehnika eriala

Juhendajad:

Robootika dotsent Karl Kruusamäe

Nooremteadur Robert Valner

Tartu 2019

Resümee

Robotid võimaldavad töötada eluohtlikes keskkondades või muul moel ligipääsmatutel aladel nagu näiteks kaevandustes, tulekahjude kustutamisel ja radioaktiivsetes keskkondades. Hõlbustamaks inimene-robot ja robot-robot koostöösüsteemide tarkvaraarendust on loodud robotite operatsioonisüsteemil (ROS) põhinev tarkvararaamistik TeMoto. TeMotol on vaja keskkonna esitamise funktsionaalsusi, et oleks võimalik arendada robotisüsteeme, mis on keskkonnast teadlikud. Töö eesmärgiks oli kavandada ja luua TeMoto arhitektuuris raamistik keskkonnamudelitega töötamiseks, luua funktsionaalsus keskkonnamudelite sünkroniseerimiseks TeMoto instantside vahel, pakkuda keskkonnamudel ja testida süsteemi reaalses stsenaariumis. Töö tulemusena valmis terviklik süsteem ja infrastruktuur, millega saab edukalt jagada semantilist ja topoloogilist informatsiooni mitme roboti vahel ja seda demonstreeriti heterogeense mitme roboti süsteemiga läbi viidud otsingumissiooni näitel. Implementeeritud TeMoto keskkonnamudeli raamistik paneb aluse keskkonnamudelite kasutusele TeMoto arhitektuuris, mis võimaldab TeMoto abil arendada keskkonnaga mõtestatult tegutsevaid robotsüsteeme.

CERCS: T125 Automatiseerimine, robootika, control engineering, T120 Süsteemitehnoloogia, arvutitehnoloogia

Märksõnad: TeMoto, ROS, robootika, mitme roboti koostöö, heterogeensed robotite süsteemid, teadmiste esitus, keskkonnamudel, topoloogiline kaardistamine, semantiline kaardistamine

Abstract

Multi-robot collaboration functionalities for robot software development framework TeMoto

Robots enable us to operate in hazardous or otherwise unvisitable environments such as mines, fires and radioactive environments. TeMoto, which is built upon the Robotics Operating System (ROS), makes it easier to develop scalable, manageable and reliable software for robotics systems. TeMoto needs functionalities to represent the environment in

order to support development of robot systems that are aware of their surroundings. The aim of this work is to plan and design a framework for working with environment models, enable robots to synchronize the environment data, create an environment model to match the framework and test the system in a real world scenario. This was achieved by implementing data structures representing objects and rooms/spaces in the, designing an abstract interface to work with the data structures and implementing the interface to create a corresponding environment model. The work resulted in a functional system and infrastructure, which allows sharing semantic and topological data between robots, which was demonstrated in a trash collecting mission featuring a heterogeneous multi-robot system. The implemented framework lays a foundation for use of environment models in TeMoto, which allows designing robot systems that interact with the world in a meaningful way.

CERCS: T125 Automation, robotics, control engineering, T120 Systems engineering, computer technology

Märksõnad: TeMoto, ROS, robotics, multi-robot cooperation, heterogeneous multi-robot systems, knowledge representation, environment model, topological mapping, semantic mapping

Sisukord

1 Sissejuhatus

Robotid võimaldavad töötada eluohtlikes keskkondades või muul moel ligipääsmatutel aladel nagu näiteks kaevandustes [1], päästemissioonidel [2] [3], tulekahjude kustutamisel [4] ja radioaktiivsetes keskkondades [5]. Tundmatus ja etteaimamatus keskkonnas mitme roboti kasutamine suurendab veakindlust ja võimaldab töötada efektiivsemalt kui oleks võimalik ühest robotist koosneva süsteemiga [6].

Üks valdkonna suurimaid väljakutseid on mitme roboti koostöö koordineerimine. Efektiivse mitme roboti koostöö korraldamisel on oluline tagada ühtne arusaam keskkonnast ja suhtluskanalid, mille kaudu saavad robotid vahetada informatsiooni. Vastavad suhtluskanalid ja ka lahendused mitmetele muudele robootika tüüpprobleemidele pakub vabavaraline tarkvararaamistik ROS, mis pakub tarkvara infrastruktuuri robotite vahelise suhtluse haldamiseks ja robotiplatvormist sõltumatu tarkvara loomiseks. Kuigi ROS aitab luua ühtlustatud tarkvara robotite vahel, peab mitme roboti süsteemides arhitektuuri paika panema arendaja.

TeMoto on ROSil põhinev tarkvararaamistik, mis hõlbustab inimene-robot ja robot-robot koostöösüsteemide tarkvaraarendust. TeMoto pakub erinevaid tööriistu mitme roboti koostöörakenduste arendamiseks, nagu näiteks infrastruktuuri informatsiooni automaatselt sünkroniseerimiseks erinevate robotite vahel. Praeguses arendusfaasis puuduvad TeMotos keskkonna esitamise funktsionaalsused.

Käesoleva bakalaureusetöö eesmärgiks on arendada tarkvararaamistiku TeMoto keskkonna esitamiseks ja mitme roboti vahel sünkroniseerimiseks vajalikke funktsionaalsusi. Töös antakse ülevaade olemasolevatest keskkonna esitamise meetoditest (peatükk 2.2), tutvustatakse tarkvararaamistikku TeMoto (peatükk 2.4) ja kirjeldatakse valminud lahendust (peatükk 4). Lisaks demonstreeritakse valminud funktsionaalsusi reaalses stsenaariumis mitme robotiga läbi viidud otsingumissiooni näitel (peatükk 5).

2 Kirjanduse ülevaade

2.1 Mitme roboti koostöö

Robotid võimaldavad töötada eluohtlikes keskkondades või muul moel ligipääsmatutel aladel. Roboteid kasutatakse näiteks kaevandamisel [1], päästemissioonidel [2] [3], tulekahjude kustutamisel [4] ja radioaktiivsetes keskkondades töötamisel [5], kus enamasti on töösse üheaegselt rakendatud üks robot.

Mitme erineva roboti kasutamine lisab paindlikkust – meid ei piira üheainsa roboti füüsilised omadused ja suutlikkus. Näiteks tulekahju kustutamiseks võib olla otstarbekas kasutada mitut väikest ja kiiret anduritega varustatud robotit, mis annaks edasi andmeid tulekustutiga varustatud suuremale ning parema kõrgete temperatuuride taluvusega robotile [7].

Mitme roboti koostöösüsteeme on rakendatud näiteks maavärina tagajärjel hävinud varisemisohus kiriku kahjustuste hindamisel, kasutades ühest UAV-st ja ühest UGV-st koosnevat süsteemi [8]. Erinevate robotite kasutamine võimaldas saada terviklikuma ülevaate hoone kandekonstruktsioonide seisust.

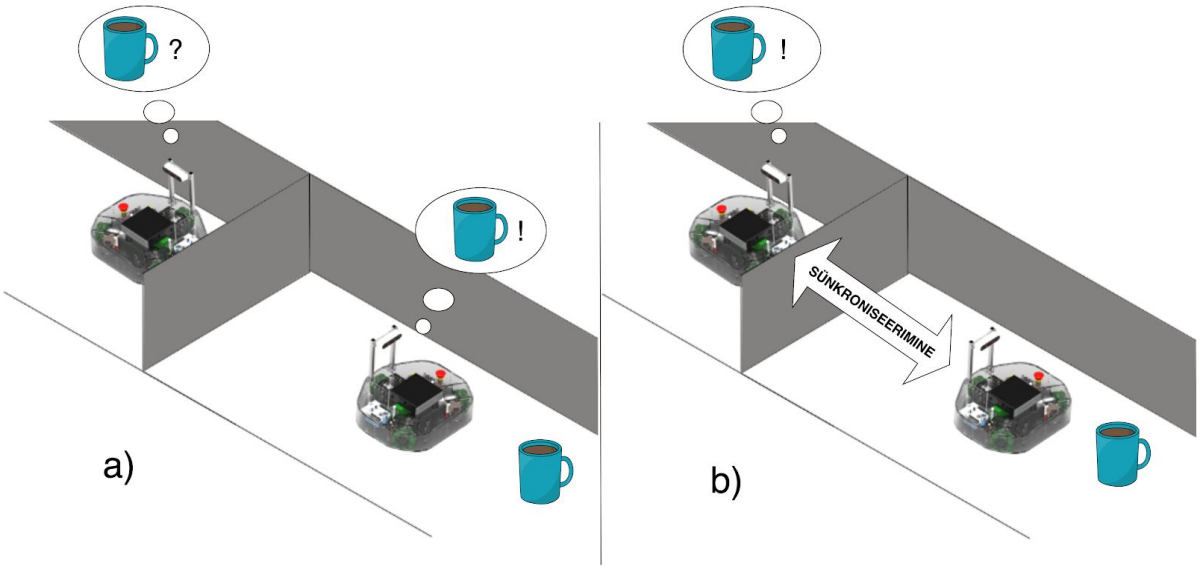
Levinud mitme roboti süsteemide rakendus on tuvastamis- ja jälitamismissioonid [9]. Üks levinud alamjuht on kunstigalerii probleem, mis üritab minimeerida valvurite arvu mingi teatud geomeetriaga ruumi valvamiseks, kusjuures valvurite asukoht ei muutu [9]. Pimenta jt [10] tutvustavad lahendust probleemi edasiarendusele, kus valvurid on suutelised liikuma. Kui valvurid näevad sihtmärki, siis määratakse sihtmärki jälitama lähim valvur ja ülejäänute trajektoorid arvutatakse ümber. Seema Kamath jt [11] tutvustavad lahendust mitme robotiga mitme sihtmärgi jälgimiseks. Robotid jagatakse paaridesse ja sihtmärgi asukoht tuvastatakse kahe roboti andurite kombineerimisel. Igale sihtmärgile vastab üks robotite paar, kes määratakse dünaamiliselt vastavalt robotite ja sihtmärgi asukohtadele. See lahendus oli implementeeritud tühjades tubades, võtmata arvesse tavalises keskkonnas leiduvaid esemeid, mis teevad trajektoori arvutamise keerulisemaks.

2.2 Teadmiste esitus robootikas

Teadmiste esitus (ingl *knowledge representation*) on uurimisvaldkond, mille eesmärgiks on võimaldada robotitel kasutada semantilist ehk tähenduslikku informatsiooni keskkonna kohta [12], [13]. Semantilise informatsiooni kasutamine roboti töös võimaldab roboti juhtimiseks kasutada inimesele loomulikke juhiseid [14]. Näiteks, kui paluda robotil tuua endale tass kohvi, siis oleks robotil vajalik teada ka mis on tass, mis on kohv, ja et mõlemat on võimalik leida köögist.

Randall Davis jt [13] kirjeldavad teadmiste esitust viie põhilise rolli kaudu, mida teadmiste esitus täidab. Esiteks võimaldab teadmiste esitus teha järeldusi maailma kohta ilma maailmaga kokku puutumata. Teiseks paneb teadmiste esitus paika, mis terminite ja kontseptsioonide abil robot maailma tunnetab. Kolmandaks võimaldab teadmiste esitus teha oma teadmiste põhjal järeldusi. Neljandaks võimaldab teadmiste esitus teha efektiivsemaid arvutusi. Viiendaks on teadmiste esitus inimesele intuitiivselt mõistetav.

Keskkonna informatsiooni saab kasutada ka mitme roboti koostöö korraldamiseks. Joonisel 1 on kujutatud olukorda, kus informatsiooni jagamine kahes erinevas ruumis paiknevate robotite vahel aitab leida kohvitassi. Vasakpoolne robot tahab teada kohvitassi asukohta (joonis 1-a), kuid kuna tass asub vaateväljast väljas, siis ei ole seda võimalik tuvastada. Küll aga näeb kohvitassi teine robot (joonis 1-a), kes tuvastab tassi asukoha ja teavitab sellest ka esimest robotit (joonis 1-b).



Joonis 1. *Erinevates ruumides asuvad robotid jagavad informatsiooni keskkonna kohta. Pildil a) soovib vasakpoolne robot teada kohvitassi asukohta, kuid tal ei ole seda iseseisvalt võimalik tuvastada. Parempoolne robot näeb kohvitassi ja pildil b) toimub sünkroniseerimine, mille tulemusena jõuab informatsioon esimese robotini.*

Selline informatsiooni jagamine eeldab, et robotitel oleks ühine arusaam ümbritsevast keskkonnast. Üks moodus ühise keskkonna arusaama tagamiseks on keskkond kaardistada.

Ümbritseva keskkonna kaardistamise meetodid on ajalooliselt jagunenud kaheks – topoloogiline kaardistamine ja ruudustiku-põhine kaardistamine [15]. Topoloogilise kaardistamise korral on tegemist graafipõhise keskkonna esitusega, kus teada on vaid graafi tippudena esitatud punktide asukohad teineteise suhtes. Ruudustiku-põhisel kaardistamisel moodustatakse n-mõõtmeline ruudustik, mille lähtepunkti suhtes asetatakse ülejäänud punktid [15]. Viimastel aastatel on tõusnud esile lisaks kahele eelmainitud meetodile semantiline kaardistamine [16]. Semantiline ehk tähenduslik kaardistamine töötab koostöös standardsete kaardistamismeetoditega, lisades geomeetrilisele informatsioonile inimestele tähenduslikkku informatsiooni [17], mis võimaldab teha robotite juhtimist intuitiivsemaks [18] ja suurendada kaardistamise efektiivsust [19].

KnowRob-Map [20] on keskkonna mudelite ehitamise süsteem, mis ühendab ruumilist ja semantilist informatsiooni esemete kohta. Süsteemi tuumaks on andmebaas, mis sisaldab endas erinevate esemete semantilisi ja füüsilisi kirjeldusi. Süsteem võimaldab tuvastada

keskkonnas asuvaid esemeid, kategoriseerida nad vastavalt andmebaasis leiduvale informatsioonile ja teha informatsiooni põhjal järeldusi ümbruskonna kohta.

Vasudevan jt artiklis [21] tutvustatakse keskkonna esitamise lahendust, kus esitatakse keskkonda objektipõhise hierarhilise kaardina. Lahenduses on ühendatud topoloogiline ja objektipõhine kaardistamine nii, et ruumide asukohad on esitatud topoloogilise kaardina ja ruumide sisu on esitatud objektide graafina. See võimaldas robotitel edukalt ruume esemete põhjal eristada ja ära tunda. Töös valminud tööriistad ei ole avalikult kättesaadavad.

2.3 Arendusplatvorm ROS

ROS ehk robotite operatsioonisüsteem on avatud lähtekoodiga tarkvarateekide, konventsioonide ja tööriistade kogumik, mis võimaldab arendada robotitele hõlpsasti liidestuvat, riistvara platvormist sõltumatut ja taaskasutatavat tarkvara [22].

ROSi põhjal arendatud tarkvara põhineb erinevate iseseisvate ROSi sõlmede (ingl *node*) koostööl, kus iga sõlm on iseseisev arvutiprogramm [22]. Sõlmed suhtlevad omavahel saates ROS-sõnumeid (ingl *message*), mida edastatakse TCP/IP suhtlusprotokolli kasutades [23]. ROSile on omane andmekeskne lähenemine. See tähendab, et ROSi rakenduse kontekstis ei ole oluline kuidas individuaalsed sõlmed jõuavad tulemusteni, vaid ainult mis tüüpi andmeid sõlmed soovivad sisendiks ja milliseid annavad väljundiks. See võimaldab hõlpsasti sõlmi välja vahetada, sest veenduda tuleb ainult sisendandmete ja väljundandmete sobivuses.

Keerulisemad ROSi süsteemid koosnevad mitmest sõlmest. Sõlmede vahelised suhtluskanalid seadistab arendaja, kasutades vastavaid sõlmede parameetreid [22]. Kuna arvukate sõlmede puhul osutuks parameetrite käsitsi määramine tülikaks, toetab ROS nn käivitusfaile (ingl *launch file*). Käivitusfailid võimaldavad käivitada mitu erinevat sõlme ja panna samal ajal paika süsteemi toimimiseks vajalikud parameetrid.

2.4 Tarkvararaamistik TeMoto (手元)

TeMoto on ROSil põhinev tarkvararaamistik, mis hõlbustab inimene-robot ja robot-robot koostöösüsteemide, eeskätt kaugjuhitavate robotite, tarkvaraarendust. Robotite tarkvara arenduses on olulisel kohal koodi skaleeritavus, töökindlus ja hallatavus [24]. TeMoto tarkvararaamistik annab tarkvaraarendajale vajalikud tööriistad, mis hõlbustavad kaugjuhitavate robotite tarkvara loomist pidades samaaegselt silmas eelmainitud aspekte [24]. Kuna TeMoto on ROSil põhinev süsteem, pole selle rakendamine piiritletud kindla riistvaraga. See tähendab, et üks TeMoto instants (TeMoto alamprogrammide kogum) võib hallata mitut robotit, kuid rakenduse skaleeruvuse huvides on enamasti igal robotil oma TeMoto instants.

Kontseptuaalselt koosneb TeMoto alamprogrammidest ehk halduritest (ingl *manager*), nende poolt hallatavatest ressurssidest (ingl *resource*) ning haldureid juhtivatest toimingutest (ingl *action*) [24].

Ressurss on TeMoto arhitektuuris miski, mis saadakse päringu tulemusena [24]. Näiteks võib olla ressursiks videosignaal, täitur, roboti asukoha informatsioon jne. Ressursid võivad olla kombineeritud alamressurssidest [24].

Haldurite eesmärgiks on ressursside kättesaadavuse tagamine ja taaskasutamise võimaldamine. Näiteks kui ressursipäring nõuab aktiivset videosignaali, siis halduri tööks on käivitada kaamera ja edastada vajalik informatsioon päringu algatajale. See töötab vastupidiselt standardsele ROSi töövoole, kus arendaja peaks ise valima ja üles seadma sobiva kaamera ja seda haldava ROSi sõlme, mis annaksid välja soovitud videosignaali. Seega haldurid abstraheerivad ressursse nende hankimise viisidest.

Toimingud on modulaarsed käitusaegselt väljakutsutavad programmid, mis kätkevad endas arendaja poolt loodud tarkvara ja võimaldavad arendajal ressursse integreerida ressursipäringute kaudu [24]. Toimingute rakendusliides toetab loomuliku kõne liidestamist ehk toiminguid saab käivitada läbi loomuliku kõne. TeMoto tarkvararaamistik sisaldab ka loomuliku kõne toiminguteks teisendamise süsteemi, mis on asendatav TeMoto väliste lahendustega [24].

TeMoto võimaldab arendada dünaamilisi ROSi-põhiseid süsteeme, kus ROSi sõlmesid saab käivitada käitusaegselt. ROSi sõlmede tööd ja omavahelist liidestamist haldab täielikult arendaja ise. Selliste lahenduste puhul pannakse programmi tööks vajalikud parameetrid enne käivitamist paika ja programmi töö ajal neid ei muudeta, millest tulenevalt on ROSil põhinev tarkvara käitusaegselt staatiline. Reeglina määrab arendaja parameetrid ROSi käivitusfailides, mis disainitakse kindlat eesmärki silmas pidades ja ei ole tulemusena kirjutatud skaleeruvust ja modulaarsust silmas pidades.

Staatiline tarkvara piirab süsteemi paindlikkust ja kohanemisvõimet, mis on oluline süsteemi töökindluse ja ülesande täitmise edukuse seisukohast [28]. Näiteks, et saavutada staatilise süsteemiga videotagasiside liiasus, peab kõiki kaameraid hoidma üheaegselt aktiivses seisundis.

3 Töö eesmärk ja nõuded

TeMoto praeguses arendusfaasis on puudu semantilise ja topoloogilise informatsiooni kogumise, kasutamise ja mitme roboti vahel sünkroniseerimise funktsionaalsused. Keskkonna esitamiseks on vaja tutvustada TeMoto süsteemi vastav keskkonnamudel, kus semantilist ja topoloogilist informatsiooni talletada, ning keskkonnamudeli kasutamist hõlbustavad funktsionaalsused. Keskkonnamudel võimaldab TeMoto raamistikku kasutades lahendada mitme roboti koostöös esinevaid probleeme, mis vajavad robotit ümbritseva keskkonna kohta semantilist või topoloogilist informatsiooni. Ühtlasi aitab keskkonnamudel leida seoseid roboti riistvara- ja tarkvararessursside vahel (vt. Lisa 1).

Käesoleva bakalaureusetöö eesmärgiks on:

- kavandada ja luua TeMoto arhitektuuris raamistik keskkonnamudelitega töötamiseks,
- luua funktsionaalsus keskkonnamudelite sünkroniseerimiseks TeMoto instantside vahel,
- pakkuda omalt poolt nõudeid rahuldav keskkonnamudel mis ühtiks TeMoto keskkonnamudeli raamistikuga,
- testida süsteemi reaalses stsenaariumis.

Tööle esitati järgmised nõuded:

1. Keskkonnamudeli raamistik peab kirjeldama keskkonnamudelit puuna, mille tipud ehk kirjed kirjeldavad keskkonnas paiknevaid ruume, esemeid ja nendevahelisi suhteid.
2. Keskkonnamudeli raamistik peab võimaldama erinevat tüüpi TeMoto kirjete (vt. pt. 4) haldamist, sealhulgas kirjete lisamist, lugemist ning eemaldamist.
3. TeMoto instantsid peavad omavahel keskkonnamudeleid sünkroniseerima, misläbi avalduksid ühes instantsis tehtud keskkonnamudeli muutused ka teistes TeMoto instantsides.
4. TeMoto ei tohi sõltuda konkreetsest keskonnamudeli haldusprogrammist.

4 TeMoto keskkonnamudeli raamistik

Järgnevalt kirjeldatakse keskkonna esitust TeMotos, ehk kuidas esitab TeMoto informatsiooni keskkonnas leiduvate esmete ja ruumide kohta (pt 4.1). Seejärel tutvustatakse TeMoto keskkonnamudelite raamistikku, mille abil suhtleb TeMoto keskkonnamudelitega (pt 4.2), millele järgneb implementeeritud keskkonnamudeli hoidla kirjeldus (pt 4.3).

4.1 Keskkonna esitus TeMotos

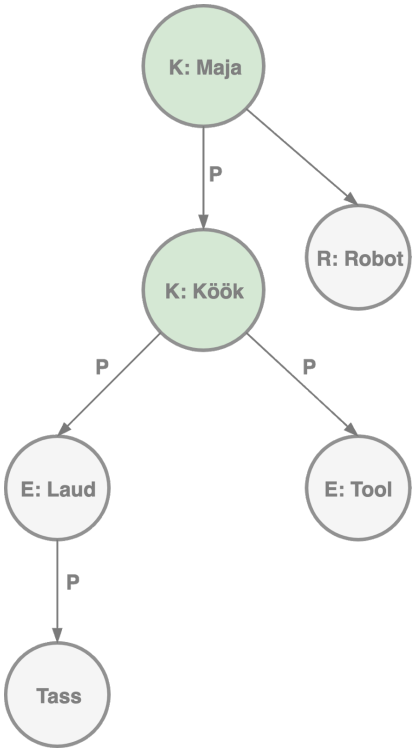
TeMoto arhitektuuris kirjeldatakse keskkonda puuna, mille tippudeks on keskkonnas esinevad füüsilised esemed ja ruumid, mida esitatakse erinevate TeMoto kirjetena. Kirjed hoiavad endas füüsilisi omadusi (suurus, kuju), topoloogilist informatsiooni (asukoht vanema suhtes) kui ka semantiliselt olulist informatsiooni (kirje tüüp). Keskkonna esitamisega tegeletakse TeMoto kontekstihalduri nimelises alamsüsteemis. Töö raames kirjeldati ära kolme erinevat tüüpi kirjed – kaardikirje, komponendikirje ja robotikirje – ja uuendati olemasolevat esemekirjet. Kõikidele kirjetele on omaseks unikaalne kirjeldav nimi ja asukoht.

Kaardikirje kirjeldab ruumi, hoides endas ruumi nime, asukohta ja kaarti ennast. **Komponendikirjed** kirjeldavad objekte, mis on osa roboti/süsteemi riistvarast (nt veebikaamera). Komponendikirjed võimaldavad leida seoseid roboti riistvara draiverite ja vastava riistvara topoloogia vahel, mis on oluline näiteks olukorras, kus on vaja käivitada teatud ruumipiirkonda katvat kaamerat.

Robotikirjed on minimalistlik esitus robotist, mis sisaldab endas informatsiooni roboteid iseloomustavate koordinaadistike kohta. See võimaldab TeMoto kontekstihalduril käivitada käitusaegselt erinevaid koordinaadistikke kasutavaid algoritme, nagu näiteks lokaliseerimisalgoritmid või esemetuvastusalgoritmid.

Esemekirje oli TeMotos kirjeldatud enne käesolevat tööd ja võimaldab kirjeldada maailmas eksisteerivat füüsilist eset. Igat eset iseloomustab unikaalne nimi, eseme kuju, eseme asukoht ja metainformatsioon, mis võimaldas vastavat tüüpi esemeid maailmas tuvastada. Esemekirjete abil oli võimalik talletada informatsiooni üksikute esemete kohta ruumis ilma omavaheliste seosteta.

Kirjed on omavahel seotud vanem-laps seostega. Igal kirjel on kuni üks vanem ja piiramatu arv lapsi. Vanem-laps seosed esindavad maailmas esinevate objektide/alade omavahelisi suhteid. Näiteks võib mõelda keskkonnast, kus köögis laua peal asub kohvitass kui puustruktuurist, kus kohvitassi vanemaks on laud, mille vanemaks on omakorda köök (vt Joonis 2).



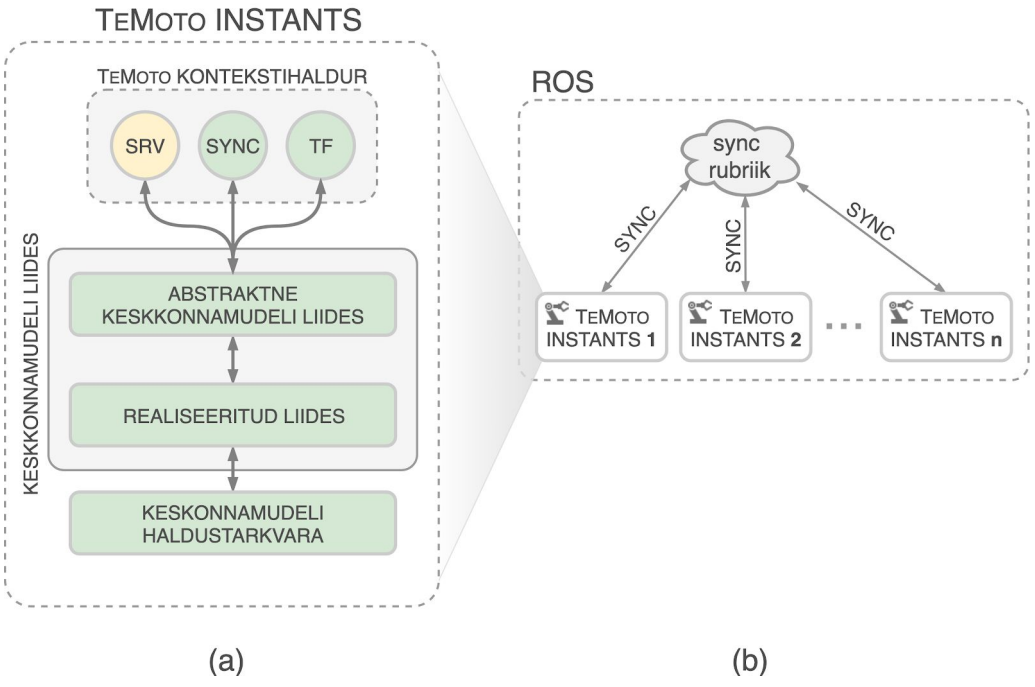
Joonis 2. *TeMoto keskkonna esituse näidis. "K" tähega märgistatud kirjed on kaardi-tüüpi kirjed, "E" tähega on tähistatud esemed, "R" tähistab robotikirjet. "P" tähega on tähistatud vanem-laps suhet.*

Selline kirjete hierarhia võimaldab lahendada erinevaid esemete omavahelisi suhteid puudutavaid praktilisi küsimusi. Vastavalt üles seatud süsteemi puhul on näiteks võimalik teha kindlaks, kas köögilaud on esemetest tühi (päring köögilaua laste kohta) või kas telekapult jäeti viimati diivanipadja alla (päring telekapuldi vanema kohta).

4.2 TeMoto keskkonnamudeli liidese ülesehitus

TeMoto keskkonnamudeli raamistiku ülesehitus on kujutatud joonisel 3. Kuna TeMotos vastutab keskkonnamudelite eest kontekstihalduri alamsüsteem, siis sellest lähtuvalt on see välja toodud ka Joonisel 3. Joonis 3-a näitab tarkvara ülesehitust ühe TeMoto instantsi siseselt. Selleks, et TeMoto kontekstihaldur ja konkreetne keskkonnamudeli haldustarkvara ei sõltuks üksteisest, loodi abstraktne liides, mida pärib konkreetse keskkonnamudeli haldustarkvaraga suhtlev realiseeritud liides. TeMoto kontekstihaldur pakub funktsionaalsusi esemete ruumis jälgimiseks ja kannab hoolt ka esemete vaheliste koordinaatteisenduste publitseerimise eest. Arendaja saab neid funktsionaalsusi kasutada defineeritud ROSi-teenuste abil, mille mugavamaks kasutamiseks on defineeritud TeMoto kontekstihalduri liideses ka vastavad funktsioonid.

Joonisel 3-b on näidatud, kuidas toimub keskkonnamudelite sünkroniseerimine TeMoto instantside vahel, mis on realiseeritud ROSi sõnumite abil. Põhjalikum ülevaade keskkonnamudelite sünkroniseerimise tööpõhimõttest pakutakse peatükis 4.2.1.



Joonis 3. *(a) TeMoto keskkonnamudeli raamistiku ülesehitus ühes instantsis. (b) keskkonnamudelite sünkroniseerimine TeMoto instantside vahel.*

4.2.1 TeMoto keskkonnamudeli abstraktne liides

Kuna vastavalt nõudele 4 peab TeMoto olema konkreetsest keskkonnamudeli haldusprogrammist sõltumatu, siis sellest lähtuvalt töötati välja lahendus, kus keskkonnamudelile päringute tegemine käib läbi liidesklassi.

Liides esitati abstraktse klassina, mida pärib keskkonnamudeli haldustarkvara liides (vt joonis 3-a). Liidese defineerimiseks oli kõigepealt oluline määratleda, milliseid baasfunktsionaalsusi TeMoto keskkonnamudelitelt ootab. Abstraktne klass kirjeldab ära funktsioonide signatuurid, mida kasutab TeMoto oma töös. TeMoto kutsub välja funktsioone otse abstraktse klassi viida kaudu, mis tähendab, et keskkonnamudeliks võib olla ükskõik milline mudel, mis pärib keskkonnamudeli abstraktset liidest ja implementeerib vajalikud funktsioonid. See võimaldab rakendada TeMotos erinevaid keskkonnamudeleid ja hoida TeMoto konkreetsest keskkonnamudeli haldurist sõltumatuna.

Keskkonnamudeli abstraktse liidese meetodid on kirjeldatud tabelis 1 ning lähtekood on toodud välja lisas 2.

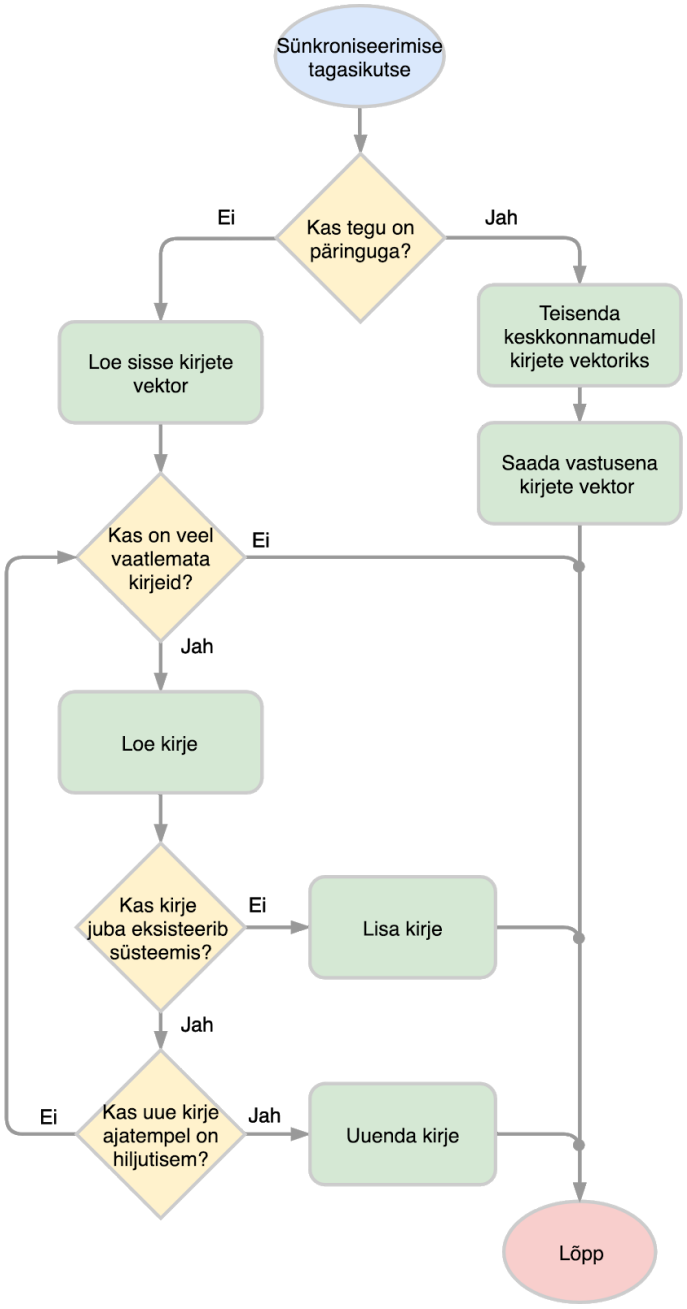
Tabel 1. *Funktsioonid, mille peab implementeerima TeMotoga kasutatav keskkonnamudeli haldustarkvara liides.*

Funktsiooni signatuur	Funktsiooni kirjeldus
`ObjectContainer getObject(string name);`	Tagastab andmebaasist nime põhjal esemekirje. (Nõue 3)
`MapContainer getMap(string name);`	Tagastab andmebaasist nime põhjal kaardikirje. (Nõue 3)
`ComponentContainer getComponent (string name);`	Tagastab andmebaasist nime põhjal komponendikirje. (Nõue 3)
`RobotContainer getRobot(string name);`	Tagastab andmebaasist nime põhjal robotikirje. (Nõue 3)
`string getTypeByName(string name);`	Tagastab kirje nime põhjal vastava kirje tüübi.
`MapContainer getNearestParentMap( string name);`	Tagastab ruumi, kus vastava nimega kirje asub.

`void updateEm(vector<ItemContainer> items_in);`	Lisab andmebaasi uue(d) kirje(d). (Nõue 1)
`void removeItem(string name);`	Eemaldab andmebaasist kirje. (Nõue 1)
`vector<ItemContainer> EmToVector();`	Tagastab keskkonnamudeli sisu jadaks teisendatuna. (Nõue 2)
`void updatePose(string name, PoseStamped newPose);`	Uuendab kirje asukoha informatsiooni nime põhjal.

4.2.2 Sünkroniseerimine

Joonisel 4 on kirjeldatud keskkonnamudelite sünkroniseerimise algoritm (lähtekood toodud välja lisas 3). Sünkroniseerimise tulemusena tagatakse kõikidel süsteemis olevatel TeMoto instantsidel alati ühtne ja ajakohane keskkonnamudel. Sünkroniseerimine toimub kahes olukorras: kui kirjeid lisatakse/uuendatakse/eemaldatakse ja kui süsteemi lisatakse uus TeMoto instants. Kui süsteemi lisandub uus TeMoto instants, siis saadab uus instants välja päringud kõikidele teistele instantsidele, et saada neilt olemasolev informatsioon keskkonna kohta. Edaspidi toimub sünkroniseerimine siis, kui keskkonnamudelis toimub mingi muutus – näiteks muutub roboti asukoht või tuvastatakse mõni uus ese.

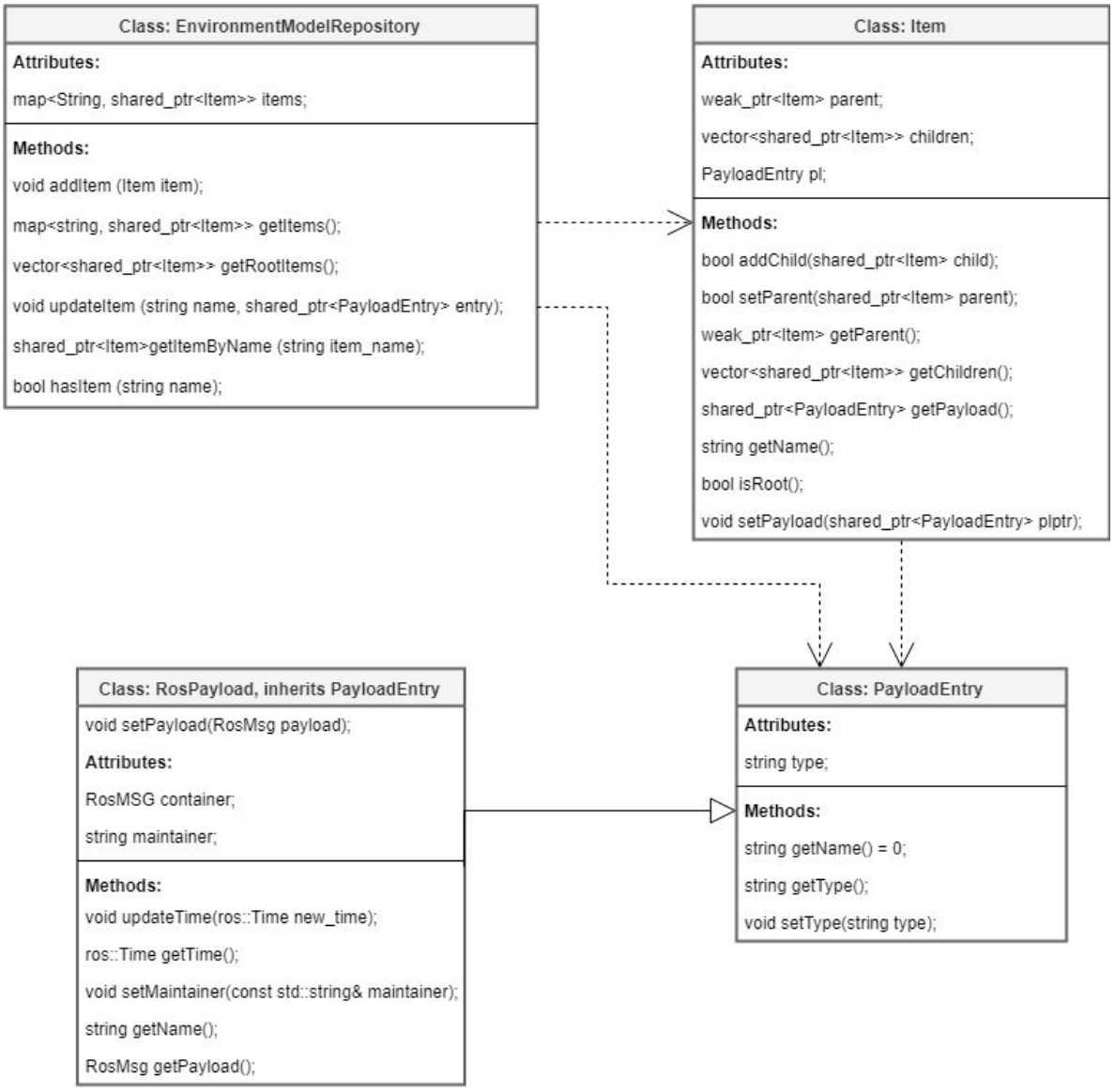


Joonis 4. *Sünkroniseerimine toimub kirjete nime ja ajatempli põhjal.*

Sünkroniseerimise käigus teisendatakse kõik keskkonnamudeli kirjed jadaks ja moodustatakse neist vektor, mis saadetakse ROSi sõnumi kujul kõikidele teistele töös olevatele TeMoto instantsidele (teistele robotitele). Kui TeMoto instantsini jõuab teise roboti keskkonnamudel, siis käiakse iga kirje ükshaaval läbi ja uuendatakse lokaalses keskkonnamudelis need kirjed, mille ajatempel on vanem kui sissetuleva kirje ajatempel.

4.3 TeMoto keskkonnamudeli hoidla (EMR)

TeMoto kirjete puustruktuuri haldamiseks implementeeriti TeMoto keskkonnamudeli hoidla ehk Environment Model Repository (EMR) koos vastava realiseeritud liidesega (Joonis 3-a). EMRi lähtekood on toodud välja lisas 4.

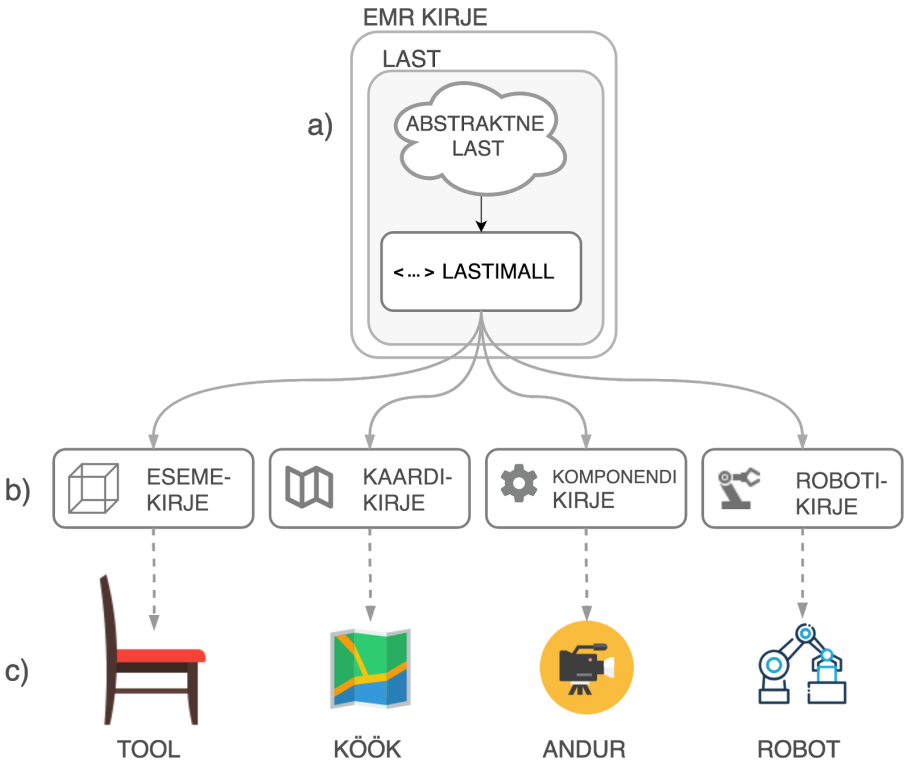


Joonis 5. *Keskkonnamudeli hoidla koosneb kirjetest, millega on seotud last.*

EMR on puu kujul realiseeritud andmebaas, mille tippusid nimetatakse kirjeteks (ingl *item*) ja kirjega seotud andmeid lastiks (ingl *payload*). Joonisel 5 on kirjeldatud EMRi kirjete (Item) ja

lastide (PayloadEntry) klassid. EMR on implementeeritud C++ klassina EnvironmentModelRepository. Klassi põhifunktsioonideks on kirjete lisamine, uuendamine, eemaldamine ja lugemine. Kirjetele ligi pääsemiseks on kaks moodust. Esiteks hoiab iga kirje endas viita oma vanemale ja viitasid oma lastele, mis võimaldab efektiivselt pöörduda antud kirje otsese vanema (O(1)) või lapse (O(n)) poole. Teiseks hoitakse EMRis viitasid kõikidele kirjetele kasutades C++ std::map konteinerit, mille võtmeks on unikaalne kirjega seotud nimi. See võimaldab efektiivset (O (log n) [25]) pöördumist suvalise kirje poole. EMR toetab ka mitme puu hoiustamist. See on kasulik olukordades, kus mingite kirjete vahelised seosed ei ole veel täpselt määratletud. Näiteks olukorras, kus süsteemis on kaks eri ruumides robotit, kuid ei ole veel teada, kus ruumid teineteise suhtes asuvad. EMR toetab mitmelõimelisust; kõik kriitilised operatsioonid on kaitstud vastastikuse välistamise kaudu (ingl *mutual exclusion*).

EMRi lastid on implementeeritud abstraktse PayloadEntry klassina (Joonisel 6-a abstraktne last), mis võimaldab hoiustada lastina ükskõik millist PayloadEntry klassi pärivat objekti.



Joonis 6. *TeMoto kirjete abstraheerimine. Isend (c) esitatakse TeMoto kirjena (b), mida hoitakse klassimalli põhjal genereeritud lastis, mis realiseerib abstraktse lasti (a) funktsionaalsused.*

Lastide sisu on rakendus-spetsiifiline, ainsad nõuded, mida EMRi tasemel lastidelt nõutakse, on nime ja tüübi olemasolu. Nime järgi toimub kirje poole pöördumine ja tüübi järgi teatakse, millise PayloadEntry klassi alamklassiga on tegu. TeMoto kirjete hoidmiseks implementeeriti klassimall ROSPayload, mis suudab hoida endas ükskõik millist TeMoto kirjet ja pakub nende haldamiseks kasulikke funktsionaalsusi nagu asendi uuendamine ja muutmisaja jälgimine.

5 TeMoto keskkonnamudeli raamistiku kasutusnäide

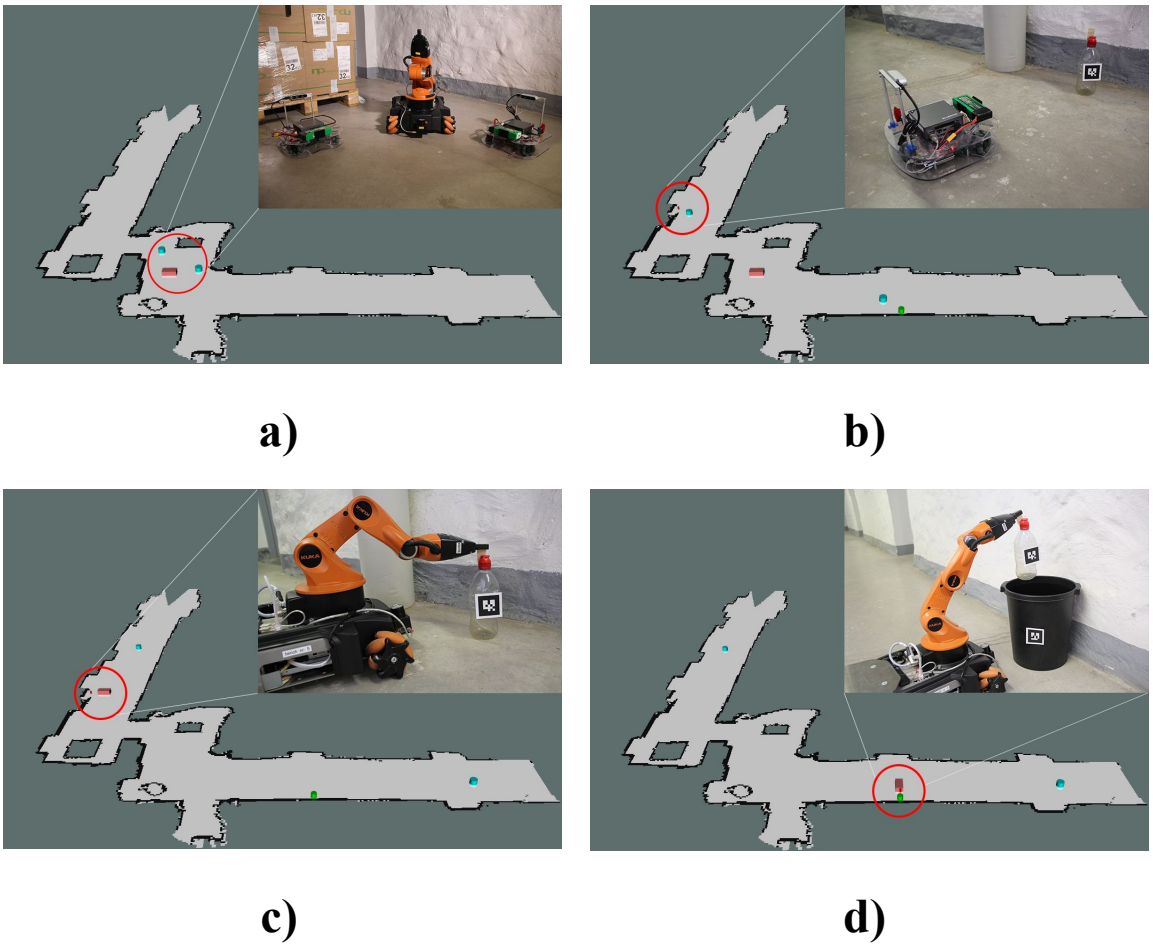
Käesolevas töös valminud TeMoto keskkonnamudeli raamistiku demonstreerimiseks sai püstitatud mitme roboti koostööd rakendav näidisülesanne, mis hõlmab endas semantilise ja topoloogilise informatsiooni kogumist ja rakendamist. Missiooni stsenaariumiks on efektiivne prügikoristus milles osales kolm robotit – kaks luurerobotit ja üks manipulaatoriga tööline (Joonis 7-a). Käsklusi jagas sülearvutiga varustatud operaator. Luurerobotite ülesandeks oli navigeerida eelnevalt kaardistatud keskkonnas ja tuvastada LR-märgistega märgitud esemeid, mis on töölise jaoks olulised. Antud stsenaariumis olid olulisteks objektideks näidisprügi ja prügikast (Joonis 7-b).

a) **b)**

Joonis 7. *Demonstratsioonis osales kaks luurerobotit ja üks manipulaatoriga tööline (a). Ülesandeks oli leida prügi ja prügikast (b) ning viia prügi prügikasti.*

Töölise ülesandeks on:

1. Oodata, kuni luuremeeskond leiab olulised objektid (Joonis 8-b),
2. Liikuda saadud informatsiooni põhjal näidisprügini ning võtta see haardesse (Joonis 8-c).
3. Liikuda prügikasti juurde.
4. Visata näidisprügi prügikasti (Joonis 8-d).



Joonis 8. *Demonstratsiooni käik. (a) robotite algasend, (b) huvipakkuvad objektid on leitud, (c) prügi on üles korjatud (d) prügi on viidud prügikasti*

Demonstratsioonis teevad koostööd neli TeMoto instantsi – Operaator, tööline ja kaks luurerobotit. Robotitele jagatakse ülesandeid käitusaegselt TeMoto NLP (*ingl* Natural Language Processing) liidese kaudu.

Operaator annab robotitele järgnevad käsud:

1. "***Operator,*** *read the hallway map*"
2. "***Everybody,*** *read the configurations*"

3. "***Everybody**, track your location*"
4. "***Scouts**, find the garbage*"
5. "***Worker**, initialize visual navigation*"

Iga käsk käivitab robotitel TeMoto toimingu, mis tegeleb vastava ülesande lahendamisega.

Esimese käsuga loetakse mälust keskkonda kujutav kaart, tehakse robotitele vastaval ROS-rubriigil nähtavaks ja lisatakse kaart ruumina keskkonnamudelisse. Teine käsk käivitab robotite riistvara draiverid ja tutvustab robotid keskkonnamudelisse. Kolmanda käsuga käivitatakse igal robotil TeMoto toru (riist- ja tarkvarakomponentide jada, kus iga komponent suunab oma väljundi järgnevale komponendile), mille tulemusena algab lokaliseerimine (toru ülesehitusega "sügavuskaamera | punktipilve töötleja | roboti lokaliseerija"). Lokaliseerimise käigus uuendatakse ka robotite asukohti keskkonnamudelis. Neljas käsk tutvustab luurerobotitele teada olevate objektide nimistu ja alustab otsinguid. Viies käsk käivitab keskkonna visualiseerimise tööriistad ja valmistab töölise navigeerimiseks ette.

Demonstratsioonis on ühendatud topoloogiline, ruudustikupõhine ja semantiline kaardistamine.

6 Arutelu

Töö tulemusena valmis keskkonnamudeli raamistik, millega saab TeMotos esitada semantilist ja topoloogilist informatsiooni ja sünkroniseerida seda mitme roboti vahel. Selle jaoks defineeriti erinevad TeMoto kirjed, mille abil on võimalik kujutada maailma ja luua käitusaegselt seoseid kirjete vahel. Implementeeriti abstraktne liides, mille kaudu on võimalik kasutada TeMotos erinevaid keskkonnamudelite implementatsioone ilma, et TeMoto sõltuks ühestki konkreetsest lahendusest. Realiseeriti funktsionaalsused TeMoto arhitektuuris, mis võimaldavad TeMoto toimingute kaudu keskkonnamudelit kasutada. Süsteemi testiti reaalses stsenaariumis, kus juhiti TeMoto abil heterogeenset mitme roboti süsteemi.

Antud arenguetapis kajastub semantiline informatsioon erinevate TeMoto kirjete tüüpidena. Iga keskkonnamudelis olev ese/ruum on täpselt ühte tüüpi – ese, ruum, komponent või robot. Tüüpe on võimalik juurde defineerida, kuid praegu ei saa üks ese olla mitut tüüpi. Mitme tüübi tugi ühel esemel oleks kasulik näiteks külmkapi esitamisel – ühest küljest konkreetsete mõõtmetega objekt, aga teisest küljest ruum, mille sees on omakorda esemed. Tulevikus võiks süsteem toetada keerulisemaid tüüpe ja seoseid tüüpide vahel, mis võimaldaks keerukamaid otsustamisprotseduure ja päringuid.

Praeguses seisus on TeMoto keskkonnamudeli liides ja implementatsioon kirjutatud samasse ROSi pakki kui TeMoto kontekstihaldur. Tulevikus võiks need olla teineteisest eraldatud ja dünaamiliselt lingitud, et oleks võimalik keskkonnamudeli haldurit käitusaegselt välja vahetada.

ROSi sõnumite kasutamine kirjete implementeerimiseks võimaldab lahendada mugavalt keskkonnamudelite sünkroniseerimist erinevate TeMoto instantside vahel, kuid seab ka teatud piirangud. Nimelt ei toeta ROSi sõnumid C++ polümorfiat ega klassimalle, mis võimaldaksid kirjeldada kirjetega tehtavaid operatsioone üldisemal kujul. See avaldab tugevat mõju süsteemi skaleeruvusele. Tulevikus, kui soovida kasutada rohkem kirjete tüüpe, tuleks leida lahendus, mis võimaldaks kirjetega üldisemal kujul manipuleerida.

7 Kokkuvõte

Mitme roboti süsteemid võimaldavad lahendada ülesandeid efektiivsemalt ja paindlikumalt kui ühe roboti süsteemid. Mitme roboti süsteemides on olulisel kohal veakindlus ja liiasus, mille tagab ROSil põhinev tarkvararaamistik TeMoto. Oluline tööriist mitme roboti koostöö koordineerimisel on informatsioon ümbritseva keskkonna kohta. Käesoleva töö raames töötati välja funktsionaalsused keskkonnamudelite kasutamiseks ja robotite vahel jagamiseks tarkvararaamistikus TeMoto. Töö tulemusena valmis TeMoto keskkonnamudeli raamistik, mis koosneb TeMoto keskkonna esitusest ja TeMoto keskkonnamudeli liidesest. Ühtlasi loodi keskkonnamudeli liidesele vastav keskkonnamudeli haldustarkvara. Valminud süsteemi demonstreeriti otsingumissiooni kontekstis, kus tegid koostööd erinevate suutlikkustega robotid.

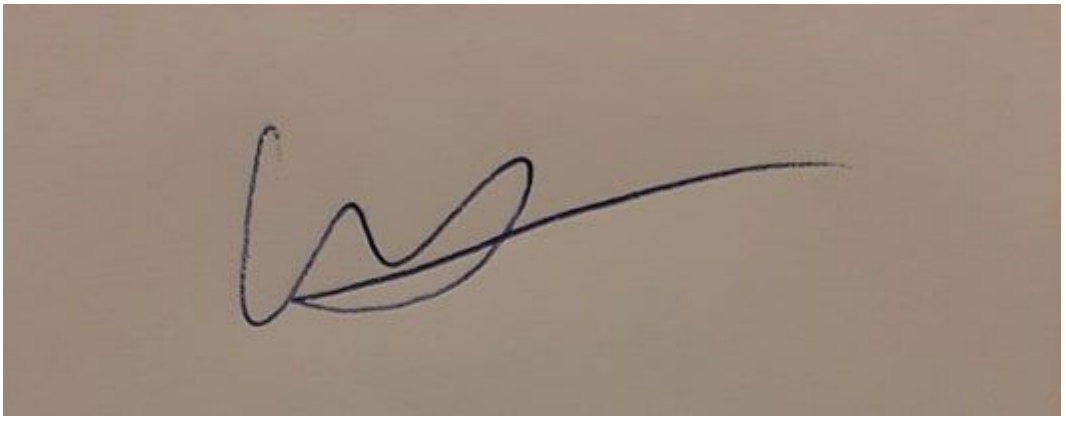
8 Tänuavaldused

Eestkätt soovin tänada oma juhendajaid Karl Kruusamäe ja Robert Valner, kelle tugi ja hea nõu innustas, inspireeris ja julgustas terve õppeaasta vältel.

Teiseks tänan robotondi tiimi – Renno Raudmäe, Veiko Vunder ja Madis Kaspar Nigol, kes võimaldasid kasutada demonstratsiooni jaoks oma suurepärast robotiplatvormi ja olid alati hea nõuga valmis.

Viimaseks tänaksin Tartu Ülikooli Tehnoloogiainstituuti, mis lahendab ära kõik mured alates tasuta kohvist ja lõpetades ruuterite hankimisega.

Allkiri:



9 Viited

[1] “Robotics in Mining | SpringerLink.” [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-32552-1_59.

[2] “Cognition-enabled Framework for Mixed Human-Robot Rescue Teams - IEEE Conference Publication.” Available: https://ieeexplore.ieee.org/abstract/document/8594311.

[3] “Disaster Robotics | SpringerLink.” [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-32552-1_60. [Accessed: 22-Apr-2019].

[4] “The Use of Robotics in Firefighting.” [Online]. Available: https://safetymanagement.eku.edu/blog/the-use-of-robotics-in-firefighting/. [Accessed: 22-Apr-2019].

[5] “Towards robotic decommissioning of legacy nuclear plant: Results of human-factors experiments with tele-robotic manipulation, and a discussion of challenges and approaches for decommissioning - IEEE Conference Publication.” [Online]. Available: https://ieeexplore.ieee.org/document/7784294. [Accessed: 22-Apr-2019].

[6] P. Gonzalez-de-Santos *et al.*, “Fleets of robots for environmentally-safe pest control in agriculture,” *Precis. Agric.*, vol. 18, no. 4, pp. 574–614, Aug. 2017.

[7] “A cooperative UAV/UGV platform for wildfire detection and fighting - IEEE Conference Publication.” [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4675411. [Accessed: 14-May-2019].

[8] G. M. Kruijff *et al.*, “Rescue robots at earthquake-hit Mirandola, Italy: A field report,” in *2012 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2012, pp. 1–8.

[9] C. Robin and S. Lacroix, “Multi-robot target detection and tracking: taxonomy and survey,” *Auton. Robots*, vol. 40, no. 4, pp. 729–760, Apr. 2016.

[10] L. C. A. Pimenta *et al.*, “Simultaneous Coverage and Tracking (SCAT) of Moving Targets with Robot Networks,” in *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*, G. S. Chirikjian, H. Choset, M. Morales, and T. Murphey, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 85–99.

[11] S. Kamath, E. Meisner, and V. Isler, “Triangulation Based Multi Target Tracking with Mobile Sensor Networks,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3283–3288.

[12] A. G. Cohn, F. Giunchiglia, and B. Selman, *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR2000), Breckenridge, Colorado, April 12-15 2000*. Morgan Kaufmann Publishers, 2000.

[13] R. Davis, H. Shrobe, and P. Szolovits, “What Is a Knowledge Representation?,” *AI Mag.*, vol. 14, no. 1, pp. 17–17, Mar. 1993.

[14] J. Fasola and M. J. Mataric, “Using semantic fields to model dynamic spatial relations in a robot architecture for natural language instruction of service robots,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 143–150.

[15] S. Thrun, “Learning metric-topological maps for indoor mobile robot navigation,” *Artif. Intell.*, vol. 99, no. 1, pp. 21–71, Feb. 1998.

[16] “Semantic mapping for mobile robotics tasks: A survey,” *Robot. Auton. Syst.*, vol. 66, pp. 86–103, Apr. 2015.
[17] A. Pronobis and P. Jensfelt, “Large-scale semantic mapping and reasoning with heterogeneous modalities,” in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 3515–3522.
[18] “Conceptual spatial representations for indoor mobile robots - ScienceDirect.” [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889008000304. [Accessed: 07-May-2019].
[19] “Speeding-up multi-robot exploration by considering semantic place information - IEEE Conference Publication.” [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1641950. [Accessed: 14-May-2019].
[20] M. Tenorth, L. Kunze, D. Jain, and M. Beetz, “KNOWROB-MAP - knowledge-linked semantic object maps,” in *2010 10th IEEE-RAS International Conference on Humanoid Robots*, Nashville, TN, USA, 2010, pp. 430–435.
[21] S. Vasudevan, S. Gächter, V. Nguyen, and R. Siegwart, “Cognitive maps for mobile robots—an object based approach,” *Robot. Auton. Syst.*, vol. 55, no. 5, pp. 359–371, May 2007.
[22] “Programming Robots with ROS: A Practical Introduction to the Robot Operating ... - Morgan Quigley, Brian Gerkey, William D. Smart - Google Books.”
[23] “ROS: an open-source Robot Operating System.” [Online]. Available: https://scholar.googleusercontent.com/scholar?q=cache:Qjc31cXD_gEJ:scholar.google.com/+ros+operating+system&hl=en&as_sdt=0,5. [Accessed: 16-May-2019].
[24] “ETIS - TeMoto 2.0: Source Agnostic Command-to-Task Architecture Enabling Increased Autonomy in Remote Systems.” [Online]. Available: https://www.etis.ee/Portal/Publications/Display/563941a8-200b-4ada-b509-1ae6a84c1e89. [Accessed: 18-May-2019].
[25] “std::map::operator[] - cppreference.com.” [Online]. Available: https://en.cppreference.com/w/cpp/container/map/operator_at. [Accessed: 23-Apr-2019].

10 Lisad

Lisa 1 Keskkonnamudeli kasutamine tarkvara ja riistvara ressursside ühendamiseks

Järgnev kood kasutab keskkonnamudelit, et moodustada ühendusi riistvara (komponendid) ja keskkonnamudeli kirjete vahel.

```
if (list_components_client_.call(list_components_srvmsg))
    {
      /*
       * Start finding liks between EMR items and Component Manager components.
       * Link is created if EMR item name is equal to component name
       */
      component_info_msgs_ = list_components_srvmsg.response.component_infos;
      for(const auto& ci : component_info_msgs_)
      {
        if (eri->hasItem(ci.component_name) && !c_em_reg->hasLink(ci.component_name))
        {
          TEMOTO_INFO_STREAM("Linking component: " << ci.component_name);
          c_em_reg->addLink(ci, ci.component_name);
        }
        else if (!eri->hasItem(ci.component_name) && c_em_reg->hasLink(ci.component_name))
        {
          TEMOTO_INFO_STREAM("Un-linking component: " << ci.component_name);
          c_em_reg->removeLink(ci.component_name);
        }
      }
    }
```

Lisa 2 TeMoto keskkonnamudeli abstraktse liidese lähtekood

```
#ifndef TEMOTO_ENV_MODEL_INTERFACE_H
#define TEMOTO_ENV_MODEL_INTERFACE_H

#include "temoto_context_manager/context_manager_containers.h"
#include <ros/ros.h>

namespace temoto_context_manager
{

/**
 * @brief Abstract class to highlight which functionalities
 *
 */
class EnvModelInterface
{
public:
/**
 * @brief Get the type of EM item by name
 *
 * @param name
 * @return std::string
 */
  virtual std::string getTypeByName(const std::string& name) = 0;

  // C++ does not support templated virtual classes :(
  /**
   * @brief Get an object type item
   *
   * @param name
   * @return ObjectContainer
   */
  virtual ObjectContainer getObject(const std::string& name) = 0;
  /**
   * @brief Get a map type item
   *
   * @param name
   * @return MapContainer
   */
  virtual MapContainer getMap(const std::string& name) = 0;
  /**
   * @brief Get a Component type item
   *
   * @param name
   * @return ComponentContainer
   */
  virtual ComponentContainer getComponent(const std::string& name) = 0;
  /**
   * @brief Get a Robot type item
   *
   * @param name
   * @return RobotContainer
   */
  virtual RobotContainer getRobot(const std::string& name) = 0;
  /**
   * @brief Get first parent item of type MAP
   *
   * @param name
```

```
   * @return MapContainer
   */
  virtual MapContainer getNearestParentMap(const std::string& name) = 0;
  /**
   * @brief Get first parent item of type OBJECT
   *
   * @param name
   * @return ObjectContainer
   */
  virtual ObjectContainer getNearestParentObject(const std::string& name) = 0;
  /**
   * @brief Get first parent item of type COMPONENT
   *
   * @param name
   * @return ComponentContainer
   */
  virtual ComponentContainer getNearestParentComponent(const std::string& name) = 0;
  /**
   * @brief Get first parent item of type ROBOT
   *
   * @param name
   * @return RobotContainer
   */
  virtual RobotContainer getNearestParentRobot(const std::string& name) = 0;

  /**
   * @brief Check if the EM has an item with this name
   *
   * @param name
   * @return true
   * @return false
   */
  virtual bool hasItem(const std::string& name) = 0;
  /**
   * @brief Remove an item
   *
   * @param name
   */
  virtual void removeItem(const std::string& name) = 0;
  /**
   * @brief Update the EMR structure with new information
   *
   * @param items_to_add
   * @param from_other_manager
   * @return std::vector<temoto_context_manager::ItemContainer> that could not be added
   */
  virtual std::vector<ItemContainer> updateEm(const std::vector<ItemContainer> & items_to_add, bool
update_time=false) = 0;
  virtual std::vector<ItemContainer> updateEm(const ItemContainer & item_to_add, bool
update_time=false) = 0;

  /**
   * @brief Save the EM state as a temoto_context_manager::ItemContainer vector
   *
   * @return std::vector<temoto_context_manager::ItemContainer>
   */
  virtual std::vector<ItemContainer> EmToVector() = 0;

  /**
   * @brief Update pose of EM item
   *
```

```
   * @tparam Container
   * @param name
   * @param newPose
   */
  virtual void updatePose(const std::string& name, const geometry_msgs::PoseStamped& newPose) = 0;
};

} // namespace emr_ros_interface

#endif
```

Lisa 3 Keskkonnamudeli sünkroniseerimise lähtekood

```
/*
 * EMR synchronization callback
 */
void ContextManager::emSyncCb(const temoto_core::ConfigSync& msg, const Items& payload)
{
  if (msg.action == temoto_core::rmp::sync_action::REQUEST_CONFIG)
  {
    advertiseEmr();
    return;
  }

  // Add or update objects
  if (msg.action == temoto_core::rmp::sync_action::ADVERTISE_CONFIG)
  {
    TEMOTO_DEBUG("Received a payload.");
    updateEm(payload, true);
  }
}


Items ContextManager::updateEm(const Items& items_to_add, bool from_other_manager, bool update_time)
{
  // Keep track of failed add/update attempts
  std::vector<ItemContainer> failed_items = em_interface->updateEm(items_to_add, update_time);

  // If this object was added by its own namespace, then advertise this config to other managers
  if (!from_other_manager)
  {
    TEMOTO_INFO("Advertising EMR to other namespaces.");
    advertiseEm();
  }
  return failed_items;
}
std::vector<temoto_context_manager::ItemContainer> EmrRosInterface::updateEm(
                  const std::vector<temoto_context_manager::ItemContainer>& items_to_add,
                  bool update_time)
{
  std::lock_guard<std::mutex> lock(emr_iface_mutex);

  // Keep track of failed add/update attempts
  std::vector<temoto_context_manager::ItemContainer> failed_items;
  for (const auto& item_container : items_to_add)
  {
    if (item_container.type == emr_containers::OBJECT)
    {
      // Deserialize into an temoto_context_manager::ObjectContainer object and add to EMR
      temoto_context_manager::ObjectContainer oc =

temoto_core::deserializeROSmsg<temoto_context_manager::ObjectContainer>(item_container.serialized_co
ntainer);
      if (!addOrUpdateEmrItem(oc, emr_containers::OBJECT, item_container.maintainer, update_time))
failed_items.push_back(item_container);
    }
    else if (item_container.type == emr_containers::MAP)
    {
      // Deserialize into an temoto_context_manager::MapContainer object and add to EMR
```

```
      temoto_context_manager::MapContainer mc =

temoto_core::deserializeROSmsg<temoto_context_manager::MapContainer>(item_container.serialized_conta
iner);
      if (!addOrUpdateEmrItem(mc, emr_containers::MAP, item_container.maintainer, update_time))
failed_items.push_back(item_container);
    }
    else if (item_container.type == emr_containers::COMPONENT)
    {
      // Deserialize into an temoto_context_manager::ComponentContainer object and add to EMR
      temoto_context_manager::ComponentContainer cc =

temoto_core::deserializeROSmsg<temoto_context_manager::ComponentContainer>(item_container.serialized
_container);
      if (!addOrUpdateEmrItem(cc, emr_containers::COMPONENT, item_container.maintainer,
update_time)) failed_items.push_back(item_container);
    }
    else if (item_container.type == emr_containers::ROBOT)
    {
      // Deserialize into an temoto_context_manager::ComponentContainer object and add to EMR
      temoto_context_manager::RobotContainer cc =

temoto_core::deserializeROSmsg<temoto_context_manager::RobotContainer>(item_container.serialized_con
tainer);
      if (!addOrUpdateEmrItem(cc, emr_containers::ROBOT, item_container.maintainer, update_time))
failed_items.push_back(item_container);
    }
    else
    {
      // TEMOTO_ERROR_STREAM("Wrong type " << item_container.type.c_str() << "specified for EMR
item");
      failed_items.push_back(item_container);
    }
  }

  return failed_items;
}


/**
   * @brief Add or update a single item of the EMR
   *
   * @tparam Container
   * @param container
   * @param container_type
   */
  template <class Container>
  bool addOrUpdateEmrItem(const Container & container,
                          const std::string& container_type,
                          const std::string& maintainer,
                          const bool update_time)
  {
  RosPayload<Container> rospl = RosPayload<Container>(container);
  rospl.setType(container_type);
  std::string name = temoto_core::common::toSnakeCase(container.name);
  std::string parent = temoto_core::common::toSnakeCase(container.parent);

  // Check for empty name field
  // Move these to the context manager interface maybe? TBD
  if (name == "")
  {
```

```
    ROS_ERROR_STREAM("Empty string not allowed as EMR item name!");
    return false;
  }
  // Check if the parent exists
  if ((!parent.empty()) && (!env_model_repository_.hasItem(parent)))
  {
    ROS_ERROR_STREAM("No parent with name " << parent << " found in EMR!");
    return false;
  }

  // Check if the object has to be added or updated
  if (!env_model_repository_.hasItem(name))
  {
    // Add the new item
    // TODO: resolve tf_prefixes, if type == component or robot, prepend maintainer
    rospl.setMaintainer(maintainer);
    std::shared_ptr<RosPayload<Container>> plptr = std::make_shared<RosPayload<Container>>(rospl);
    env_model_repository_.addItem(name, parent, plptr);
  }
  else
  {
    if (rospl.getTime() > getRosPayloadPtr<Container>(name)->getTime())
    {
      // Update the item information
      if (update_time) rospl.updateTime();
      std::shared_ptr<RosPayload<Container>> plptr = std::make_shared<RosPayload<Container>>(rospl);
      env_model_repository_.updateItem(name, plptr);
      ROS_INFO_STREAM("Updated item: " << name);
    }
  }
  return true;
}
```

Lisa 4 Keskkonnamudeli haldustarkvara EMR

env_model_repository.h

```
#ifndef TEMOTO_ENV_MODEL_REPOSITORY_H
#define TEMOTO_ENV_MODEL_REPOSITORY_H

#include <mutex>
namespace emr
{

/**
 * @brief Abstract base class for payloads
 *
 */
class PayloadEntry
{
protected:
  std::string type;
public:

  PayloadEntry(std::string type) : type(type) {}

  ~PayloadEntry() {}
```

```
  PayloadEntry() {}

  const virtual std::string& getName() const = 0;
  /**
   * @brief Get the type of the Payload
   *
   * @return std::string
   */
  const std::string& getType() const {return type;}
  void setType(std::string ntype) {type = ntype;}
};


/**
 * @brief A single item in the EMR tree
 *
 * A item contains a payload and pointers to its (singular) parent and children.
 *
 */
class Item : public std::enable_shared_from_this<Item>
{
private:
  std::weak_ptr<Item> parent_;
  std::vector<std::shared_ptr<Item>> children_;
  std::shared_ptr<PayloadEntry> payload_;

public:

  void addChild(std::shared_ptr<Item> child);
  /**
   * @brief Set the Parent pointer
   *
   * NB! Don't use this manually. Use addChild() of parent instead.
   *
   * @param parent
   */
  void setParent(std::shared_ptr<Item> parent);
  const std::weak_ptr<Item>& getParent() const
  {
    return parent_;
  }
  /**
   * @brief Get children of item
   *
   * @return std::vector<std::shared_ptr<Item>>
   */
  const std::vector<std::shared_ptr<Item>>& getChildren() const
  {
    return children_;
  }
  /**
   * @brief Get the pointer to Payload
   *
   * @return std::shared_ptr<PayloadEntry>
   */
  const std::shared_ptr<PayloadEntry>& getPayload() const
  {
    return payload_;
  }
  const std::string& getName() const {return payload_->getName();}
```

```
  /**
   * @brief Check if the item is a root item
   *
   * A item is a root item if it has no parent i.e. the weak pointer
   * to the parent is expired.
   *
   * @return true
   * @return false
   */
  bool isRoot() {return parent_.expired();}
  /**
   * @brief Set the Payload
   *
   * @param plptr shared_ptr to Object inheriting PayloadEntry
   */
  void setPayload(std::shared_ptr<PayloadEntry> plptr) {payload_ = plptr;}

  Item(std::shared_ptr<PayloadEntry> payload) : payload_(payload) {}
};

/**
 * @brief Wrapper class to store pointers to all items of a tree in a map
 *
 */
class EnvironmentModelRepository
{
private:
  std::map<std::string, std::shared_ptr<Item>> items;
  mutable std::mutex emr_mutex;
public:
  const std::map<std::string, std::shared_ptr<Item>>& getItems()
  {
    std::lock_guard<std::mutex> lock(emr_mutex);
    return items;
  }
  void removeItem(const std::string name)
  {
    items.erase(name);
  }
  /**
   * @brief Get the root items of the structure
   *
   * Since the EMR can have several disconnected trees and floating items,
   * we need to be able to find the root items to serialize the tree
   *
   * @return std::vector<std::shared_ptr<Item>>
   */
  std::vector<std::shared_ptr<Item>> getRootItems() const;
  /**
   * @brief Add a item to the EMR
   *
   * If parent name is empty, the item will be unattached and not a part of the tree.
   * You can call the item's setParent() method manually later.
   *
   * If the parent name is not empty, make sure the corresponding parent exists.
   *
   * The first item added to the EMR will be assigned as the root item.
   *
   * @param parent name of parent item
   * @param name name of item to be added
   * @param entry pointer to payload
```

```
   */
  void addItem(const std::string& name, const std::string& parent, std::shared_ptr<PayloadEntry>
payload);
  /**
   * @brief Update EMR item
   *
   * @param name string name of item
   * @param entry pointer to payload
   */
  void updateItem(const std::string& name, std::shared_ptr<PayloadEntry> entry);
  /**
   * @brief Get shared_ptr to item by name
   *
   * @param item_name
   * @return std::shared_ptr<Item>
   */
  const std::shared_ptr<Item> getItemByName(std::string item_name)
  {
    std::lock_guard<std::mutex> lock(emr_mutex);
    return items[item_name];
  }
  /**
   * @brief Check if EMR contains a item with the given name
   *
   * @param name
   * @return true if item exists
   * @return false if item does not exist
   */
  bool hasItem(const std::string& name);

};



} // namespace emr
#endif
```

env_model_repository.cpp

```
#include <iostream>
#include <map>
#include <memory>
#include <sstream>
#include <vector>
#include "temoto_context_manager/env_model_repository.h"

namespace emr
{
void EnvironmentModelRepository::addItem(const std::string& name, const std::string& parent,
std::shared_ptr<PayloadEntry> payload)
{
  std::lock_guard<std::mutex> lock(emr_mutex);
  // Check if we need to attach to a parent
  if (parent == "")
  {
    // Add the item to the tree without a parent
    items[name] = std::make_shared<Item>(Item(payload));
  }
  else
  {
```

```
    // Parent is legit, add the item to the tree
    items[name] = std::make_shared<Item>(Item(payload));
    // Add the new item as a child to the parent, creating the child -> parent link in the process
    items[parent]->addChild(items[name]);
  }
}

void EnvironmentModelRepository::updateItem(const std::string& name, std::shared_ptr<PayloadEntry>
plptr)
{
  std::lock_guard<std::mutex> lock(emr_mutex);
  items[name]->setPayload(plptr);
}

bool EnvironmentModelRepository::hasItem(const std::string& name)
{
  std::lock_guard<std::mutex> lock(emr_mutex);
  return items.count(name);
}
std::vector<std::shared_ptr<Item>> EnvironmentModelRepository::getRootItems() const
{
  std::lock_guard<std::mutex> lock(emr_mutex);
  std::vector<std::shared_ptr<Item>> root_items;
  for (auto const& pair : items)
  {
    if (pair.second->isRoot())
    {
      root_items.push_back(pair.second);
    }
  }
  return root_items;
}
/**
 * @brief Add child item to existing item
 *
 * @param child - pointer to the item to be added
 */
void Item::addChild(std::shared_ptr<Item> child)
{
  children_.push_back(child);
  child->setParent(shared_from_this());
}
/**
 * @brief Set parent of item
 *
 * @param parent
 */
void Item::setParent(std::shared_ptr<Item> parent)
{
  //   TEMOTO_DEBUG("Attempting to add " << parent->name.c_str() << " as parent to " << name);
  // Make sure the item does not already have a parent
  if (parent_.expired())
  {
    parent_ = parent;
  }
  else
  {
    // TEMOTO_ERROR("Item already has a parent.")
  }
}
```

```
} // namespace emr
```

Lihtlitsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Meelis Pihlap,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

"Mitme roboti koostöö funktsionaalsuste väljatöötamine tarkvararaamistikule TeMoto",

mille juhendajad on Karl Kruusamäe ja Robert Valner

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.

Meelis Pihlap

20.05.2019