TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Sandra Kukk 179275IABB

LAHINGUSIMULATSIOONI VBS3 (VIRTUAL BATTLESPACE 3) ASUKOHA EDASTUSPROTOKOLLI DIS (DISTRIBUTED INTERACTIVE SIMULATION) LIIDESTUSE REALISEERIMINE

Bakalaureusetöö

Juhendajad: Priit Järv

PhD

Tarmo Aia

Eesti kaitseväe küberväejuhatuse info- ja kommunikatsiooni-tehnoloogia keskuse projektijuht

Tallinn 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Sandra Kukk

17.05.2020

Annotatsioon

Antud lõputöö eesmärgiks oli välja arendada süsteem, mis võtab DIS protokollist vastu informatsiooni ning edastab selle Google Protocol Buffers formaadis üle REST API liidese teistele süsteemidele. Eesmärgi valideerimiseks kasutati kindlate stsenaariumite testimist.

Antud lõputöö tulemusena valmis prototüüp, mis võttis DIS implementatsioonist vastu PDU informatsiooni ning edastas selle Protobufi formaadis REST API liidesega serverisse. Töö käigus valminud rakendust on võimalik edasi arendada. Loodud lahendust saab rakendada reaalse implementatsiooni arendusel, mis on mõeldud Eesti kaitseväes simulatsioonisüsteemide andmete kasutamiseks koos reaalsete andmetega ja hilisemaks analüüsiks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 20 leheküljel, 6 peatükki, 7 joonist, 1 tabelit.

Abstract

Implementing DIS (Distributed Interactive Simulation) protocol interface for VBS3 (Virtual Battlespace 3) location service

The aim of this thesis was to develop a system, that receives information from the DIS protocol and transmits it serialized over the REST API to other systems. To validate this aim, specific scenarios were tested.

As a result of this thesis, a prototype was made, which received PDU information from the DIS implementation and transmitted it in Protobuf format to a server with a REST API. This application can be further developed. The created solution can be applied in the production-ready implementation in the Estonian Defense Forces for the use of simulation system data together with real data and for later analysis.

The thesis is in Estonian and contains 20 pages of text, 6 chapters, 7 figures, 1 table.

Lühendite ja mõistete sõnastik

AKIT	Andmekaitse ja infoturbe leksikon
API	*Application programming interface*, rakendusprogrammiliides, tagab reeglid ja vahendid rakendusprogrammi suhtluseks
APP-6	*Allied Procedural Publication 6*, väljaanne, mille eesmärk on tagada ja kehtestada NATO sümboolika standardid ja nõuded andmete kasutamisele teenusevahetus süsteemides, mis suudavad efektiivselt genereerida ühistes operatsioonipiltides sümboleid.
DARPA	*The Defense Advanced Research Projects Agency*, USA kaitseuuringutele ja uute militaartehnoloogiate väljatöötamisele spetsialiseerunud agentuur
DIS	*Distributed Interactive Simulation*, IEEE standard, mida kasutatakse militaarsimulatsioonis osalevate objektide omavaheliseks suhtluseks
HLA	*High Level Architecture*, IEEE standard, mida kasutatakse mitme simulatsiooni koostööks
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll, protokoll teabe edastamiseks arvutivõrkudes
HTTPS	*Hypertext Transfer Protocol Secure*, turvaline hüperteksti edastusprotokoll, turvaline protokoll autenditud ja krüpteeritud informatsiooni edastamiseks arvutivõrkudes
IEEE	*Institute of Electrical and Electronics Engineers*, Elektri- ja Elektroonikainseneride Instituut, maailma suurim rahvusvaheline elektrotehnikat arendav mittetulunduslik erialaorganisatsioon
Java	Ettevõtte Sun Microsystems ja vabavarakommuuni poolt arendatav programmeerimiskeel, mis kasutab erilist baitkood-tehnoloogiat ning mille jaoks on loodud eraldi Java virtuaalplatvorm, millega saab antud programmeerimiskeele kompileeritud produkte jooksutada väga erinevatel alusplatvormidel
JSON	*JavaScript Object Notation*, andmevahetusformaat
JWT	*JSON Web Token*, veebitõend, mida kasutatakse identiteedi informatsiooni saatmiseks ning kontrollimiseks
Küpsis	Tekstikujuline andmeplokk kliendi veebibrauseris, mida

	saadetakse määratud domeenile iga kord, kui klient teeb sinna päringu
LAN	*Local Area Network*, arvutivõrk, mis ühendab piiratud maa-alal, hoones jne asuvaid arvuteid ja võrguseadmeid
MOVES	*Modeling Virtual Environments and Simulation Institute*, USA-s asuva Mereväe aspirantuuri (*NPS*) Modeleerimise, Virtuaalsete keskkondade ja Simulatsioonide Instituut
NATO	*North Atlantic Treaty Organisation*, Põhja-Atlandi Lepingu Organisatsioon, sõjaline liit, põhineb kollektiivkaitsel, läbi mille liikmesriigid nõustuvad välise rünnaku korral vastastikust kaitset osutama
NPS	*Naval Postgraduate School*, USA-s asuv Mereväe aspirantuur
PDU	*Protocol Data Unit*, üle võrgu saadetav ja informatsiooni sisaldav pakett
RSA	Rivest–Shamir–Adleman, algoritm krüpteerimiseks ning dekrüpteerimiseks
SISO	*Simulation Interoperability Standards Organization*, Simulatsiooni Koostöövõime Standardite Organisatsioon, mis tegeleb simulatsioon ja mudelite koostöötalitlusega
SSL	*Secure Sockets Layer*, turvaprotokoll, mis tagab, et keegi ei saa jälgida veebilehele sissetulevat ja väljaminevat liiklust
TDD	*Test-driven development*, testimisel põhinev arendus
TENA	*Test and Training Enabling Architecture*, testimist ja treenimist võimaldav arhitektuur
VBS	*Virtual Battlespace*, militaarsimulatsioon
WAN	*Wide Area Network*, laivõrk, suure geograafilise ulatusega arvutivõrk

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva töö sisuks on Eesti Kaitseväes (edaspidi tellija) kasutatava matkesüsteemi Virtual Battlespace 3 (VBS3) liidestusprotokolli DIS ära kasutades tuua simulatsiooni objektide info kujule, mida on võimalik masinloetavalt töödelda teiste süsteemide poolt.

Probleem seisneb selles, et hetkel puudub standardiseeritud liides, mis muudaks VBS3 objektide info loetavaks teistele süsteemidele.

Eesmärgiks on välja arendada süsteem, mis võtab DIS protokollist vastu informatsiooni ning edastab selle serialiseerituna üle REST API liidese teistele süsteemidele. Serialiseerimisel kasutatakse Google Protocol Buffers lahendust. Tulemuse kontrollimiseks kasutatakse testimist viie stsenaariumi korral. Loodud arendust saab rakendada simulatsioonisüsteemide andmete kasutamiseks koos reaalsete andmetega ja hilisemaks analüüsiks.

Peatükis kaks kirjeldatakse DIS ja VBS3 tausta ning nende alternatiive. Järgmises peatükis tuuakse välja töös kasutatud tööriistad, keskkonnad ja metoodikad. Kahes viimases osas antakse ülevaade saadud tulemustest ning järeldustest.

2 Taust

Käesolevas peatükis antakse ülevaade DIS protokollist ja VBS matkesüsteemist ning nende alternatiividest. Lisaks tuuakse välja juba olemasolevad lahendused.

2.1 Edastusprotokoll DIS

Distributed Interactive Simulation on jaotatud interaktiivne simulatsioon. Sõna "jaotatud" on selgitatud 1995. aastal välja antud teoses "*Distributed Interactive Simulation of Combat*" järgnevalt: "Jaotatud interaktiivne simulatsioon on selles mõttes "jaotatud", et üleriigiliselt mitmetes treeningpaikades asuvad simulatsiooni arvutid on ühendatud lokaalvõrguga, mis omakorda võivad olla seotud laivõrguga". Interaktiivne pool tuleneb sellest, et inimeste tegevused mõjutavad simulatsiooni käekäiku. Osalejad saavad võidelda nii arvuti kui ka teiste osavõtjate vastu. [32, lk 2-3]

DIS-i eelkäijaks oli SIMNET, mis loodi 1980ndatel ja rahastati DARPA poolt. 90ndate alguses arenes sellest välja DIS. Tänapäeval tegeleb standardi muutmisega ning parandamisega SISO ehk Simulatsiooni Koostöövõime Standardite Organisatsioon. Standard on defineeritud ka IEEE Standard 1278 all. [40, lk 1-2] 2020. aasta mai seisuga on välja antud seitse DIS versiooni. Hetkel arendatakse kaheksandat versiooni. [28]

DIS standard on edastusprotokoll, mille abil saavad simulatsiooni mudelid üksteisega suhelda üle LAN, WAN ja muude meediate. Kommunikatsioon toimub sõnumite abil, mida kutsutakse PDU (*Protocol Data Unit*) pakettideks. [42, lk 26] DIS seitsmenda versiooni seisuga on kokku 72 erinevat sorti PDU-d [18, lk 50]. Pakettides olev info sõltub tüübist. Osad annavad teavet asukoha, teised raadioside kohta. Näiteks relva laskmisel võetakse kasutusele *Fire PDU*, mis näitab, kes keda lasi ning milline laskemoon oli kasutusel. [42, lk 27, 32] Kõik PDU paketid sisaldavad samataolist päist, kus määratakse ära näiteks, millal pakett saadeti ja milline on protokolli versioon (Joonis 1). Lisaks tavalisele päisele, sisaldab näiteks *Fire PDU* ka infot sihtmärgi, asukoha ning laskja kohta (Joonis 2). DIS protokoll on võetud kasutusele peamiselt

sõjaväeliste simulatsioonide läbiviimiseks [13, lk 1]. Näiteks rakendatakse seda õhutõrje simulatsioonis MACE (*Modern Air Combat Environment*) [25]. DIS protokolli kasutamiseks on loodud erinevaid rakendusi. Näiteks ettevõte RedSim 2 pakub tarkvara DIS-põhiste rakenduste planeerimiseks ning testimiseks [37].

Field Name	Data Type	Purpose
Protocol Version	8 bit enumeration	Version of DIS; often 6 or 7
Exercise ID	8 bit unsigned integer	ID if this simulation
PDU Type	8 bit enumeration	Type of message
Protocol Family	8 bit integer	Group PDU belongs to
Timestamp	32 bit unsigned integer	Time PDU was sent
Length	16 bit unsigned integer	How many bytes in message
PDU Status	8 bit record	Used to add information to certain PDUs
Padding	8 bits unused	Unused, save for later designs

Joonis 1. PDU päises olev info [26].

Item Name	Bit Length	Opt	Opt Ctl	Rpt	Rpt Ctl
PDU Header Record	96				
Firing Entity Id Record	48				
Target Entity Id Record	48				
Munition Id Record	48				
Event Identifier Record	48				
Fire Mission Index Field	32				
Location in World Record	192				
Burst Descriptor Record	128				
Velocity Record	96				
Range Field	32				

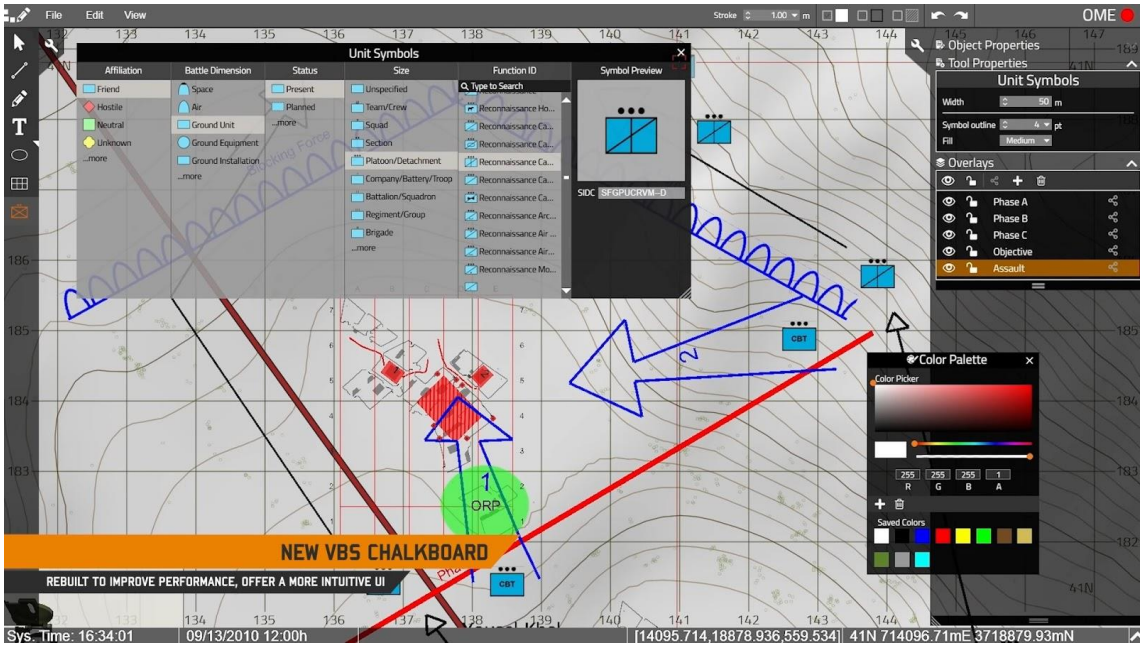
Joonis 2. *Fire PDU* komponendid [2].

2.2 Matkesüsteem VBS3

Virtual Battlespace 3 on ettevõtte *Bohemia Interactive Simulations* poolt arendatud simulatsioonitarkvara. Platvormi abil saab virtuaalses maailmas läbi viia enda

tingimuste järgi militaarseid treeningharjutusi, näidisstsenaariumeid ning pärast saadud tulemusi analüüsida. [4]. Antud lahendus sisaldab ka eelnevalt mainitud DIS edastusprotokolli. Joonisel 3 kuvatakse objektide sisestamist stsenaariumi ning joonisel 4 NATO APP-6 sümbolite kasutamist missiooni planeerimisel.

Joonis 3. VBS3 objektide sisestamine stsenaariumisse [5].



Joonis 4. NATO APP-6 sümbolite kasutamine missiooni planeerimisel [5].

VBS3 on evitatud sõjaväes üle viiekümnes riigis [29]. Sinna alla kuulub ka Eesti kaitsevägi. Näiteks Kalevi jalaväepataljoni operatiivsektsiooni ülema ja kapteni Ivo Peetsi arvates sobib VBS hästi sõduritele ja ülematele koostöö harjutamiseks [35, lk 55]. Lisaks on VBS-i lõimitud vähemalt kolmekümnesse projekti. Itaalia firma Vitrociset poolt loodud lahendus *Forest Fire Area Simulator* ehk Metsatulekahju Piirkonna Simulaator on kasutanud seda metsatulekahjude matkimiseks ja treeninud selle abil riikliku metsanduskorpust [3]. Eestis on võimalik kasutada VBS3 süsteemi Kaitseväe Akadeemia matkekeskuses [8]. Peale kaitseväe on süsteemi kasutanud ka pääste- , piirivalve- ja politseiamet [35, lk 55].

Lahingu läbiviimine arvutis aitab kokku hoida aega ja raha. Esiteks kulub vähem aega harjutuste planeerimiseks ja koordineerimiseks. Samuti saab ühe päeva jooksul saab läbi viia mitu protseduuri. 3D ja 2D vaade aitab administraatoril mängu paremini jälgida ning juhtida. [35, lk 53, 55] Juhul kui lahingu jaoks vajalikku varustust või muud ressurssi pole, on neid võimalik genereerida simulatsioonis. Simulatsiooni lõppedes on võimalik anda harjutusele ja igale osalejale individuaalset tagasisidet. Kapten Ivo Peets sõnab Sõdurilehes järgnevalt: “Näiteks on simulaatoris võimalik pärast harjutuse sooritust näidata filmilindilt, millised probleemid ilmnesid, paralleelselt kuvades selle olukorra õige lahendus. See võimaldab ülematel tagasisidestamisel minna detailidesse, et hiljem maastikul toimuvatel harjutustel vältida ohtlikke olukordi” [35, lk 55].

2.3 Avatud lähtekoodiga lahendused

Tellija poolt oli oluline, et lahendus oleks vabavaraline ehk litsentsivaba. Samuti pidi lahendus olema avatud lähtekoodiga, et rakendust oleks võimalik arendada ja siduda erinevate kaitseväe süsteemide külge. Üheks olemasolevaks lahenduseks on firma VT MAK poolt arendatud API tööriistakomplekt VR-Link, millega on võimalik enda rakendus muuta ühilduvaks DIS või HLA (*Higher Level Architecture*) protokolliga [47]. Mainitud lahendus ei sobi aga tellijale, sest see on tasuline ning litsentseeritud. Seega väidab autor, et teadaolevalt avatud lähtekoodiga, vabalt kasutatav või sarnane mikroteenuse lahendus puudub.

2.4 Teised protokollid ja matkesüsteemid

Lisaks DIS protokollile on olemas HLA (*Higher Level Architecture*) ja TENA (*Test and Training Enabling Architecture*). Kui DIS puhul on andmete vahetamiseks mõeldud info defineeritud standardi sees (näiteks formaat), siis HLA ja TENA korral formaadid ning muud reeglid info vahetamiseks protokolli osa alla ei kuulu. [17, lk 1] HLA loodi selleks, et parandada protokolli DIS puudused. Nimelt oli USA kaitseministeeriumil vaja lahendust, mis toetaks simulatsioonide omavahelist koostöövõimet ning selle tagas uus HLA. [41, lk 1]. Steffen Straßburger on oma artiklis öelnud järgmist: "HLA peamine eesmärk oli tagada vaba arhitektuur, mis pakuks teenuseid koostalitusvõimeks ja taaskasutamiseks" [46, lk 5]. TENA oli kaitseministeeriumil vaja mitmete simulatsioonide koostöö testimiseks ja treenimiseks [30, lk 259].

Ka VBS platvormile leidub sarnaseid lahendusi. USA sõjaväes on kasutusel *Synthetic Training Environment* (STE), kus saab samuti läbi viia militaarharjutusi. Erinevalt VBS-ist on STE külge plaanitud arendada elektrooniline ja küberruumis toimuv sõjapidamine. [22] USA ettevõte FAAC pakub peale sõjaväelise treeningu ka simulatsioone päästetöötajatele. Näiteks on neil võimalik praktiseerida tuletõrje- ja kiirabiautoga sõitmist. [12] Antud toode kasutab ka DIS standardit [11]. Ühendkuningriikides on väljaõppe saanud 15 000 inimest tänu firma NSC poolt loodud süsteemile *Unit Based Virtual Training* [31]. Lisaks varasemalt mainitud ettevõtetele pakuvad militaarseid simulatsioone ka veel järgmised firmad: Treality SVS, Bagira Systems, Saab AB, Aechelon Technology, Virtual Heroes, SimCentric Technologies jne [27, lk 40]. Eesti kaitsevägi kasutab Tapal lahingumasina CV90 treeninguteks ettevõtte eSim poolt loodud simulatsiooni Steel Beasts. Mainitud lahendusele ei realiseeritud käesolevas lõputöös liidest, sest seda kasutatakse ainult soomukijuhtide koolitamiseks. Autorile teadaolevalt ei plaani Eesti kaitsevägi lähitulevikus VBS süsteemilt üle minna ning seega keskenduti töös VBS liidestusele.

3 Metoodika

Antud peatükis esitatakse töö metoodika. Esimeses osas tuuakse esile nõuded ning teises osas arhitektuur. Järgmistes alapeatükkides kirjeldatakse ja põhjendatakse kasutatud lahendusi ja tööriistu.

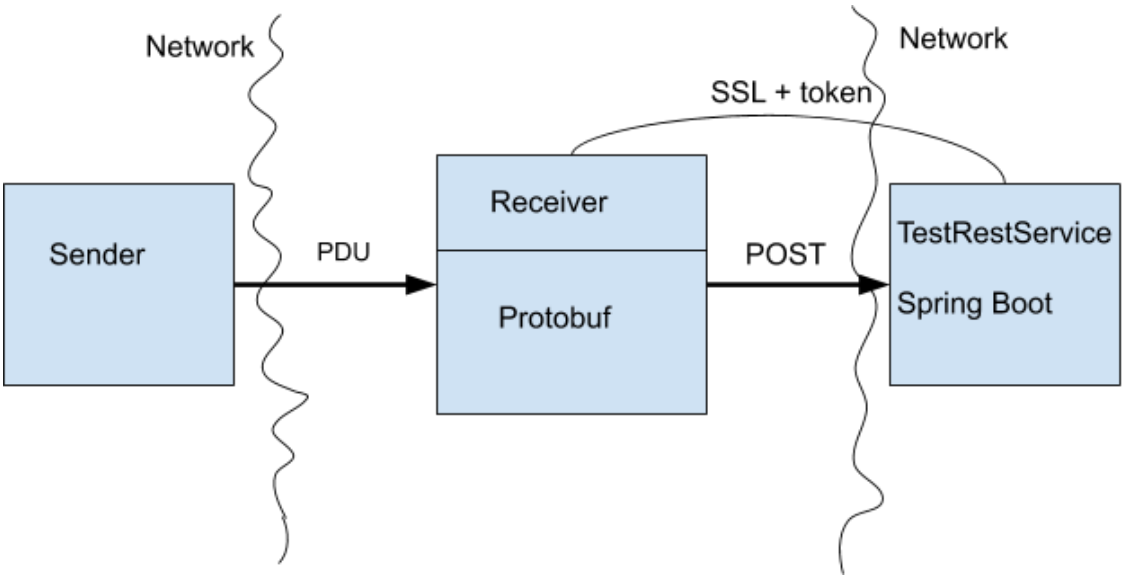
3.1 Nõuded

Süsteemi eesmärgiks on välja arendada süsteem, mis võtab DIS protokollist vastu informatsiooni ning edastab selle serialiseerituna üle REST API liidese teistele süsteemidele. Lisa 1 sisaldab kaitseväe poolt ettemääratud nõudeid.

3.2 Valikud ja arhitektuur

Autori enda valida oli programmeerimiskeel, arendus- ja versioonhalduskeskkond. Samuti oli vaba valik SSL tehnoloogia, tokeni lahenduse ning täpse DIS implementatsiooni kasutamine. Ülejäänud valikud tulenesid kaitseväe nõuetest (Lisa 1).

Enne arendamise alustamist pandi paika esialgne abstraktne arhitektuur. Kokku koosneb lahendus kolmest plokist (Joonis 5). Klassid Sender ning TestRestService on vajalikud protoüübi ning testide jaoks. Sender on klass, mis saadab PDU sõnumeid konfigureerimisfailis täpsustatud pordile ja IP-aadressile. Receiver klass kuulab ja võtab need vastu. Sõnum serialiseeritakse Protobuf formaati ning pärast autentimist ning autoriseerimist saadetakse sõnum POST meetodiga klassi nimega TestRestService.



Joonis 5. PDU sõnumi saatmine.

3.3 Keskkonnad ja keel

Rakenduse tegemiseks kasutati keskkondi IntelliJ IDEA ja GitLab ning programmeerimiskeelt Java autori varasemate kogemuste tõttu. IntelliJ IDEA on arenduskeskkond, mis sisaldab mitmeid mugavaid tööriistu, näiteks silurit (*debugger*) ja versioonikontrolli [19]. Java puhul kasutati versiooni 11, sest see on kõige viimasem *Long-Term-Suppor*t (LTS) ehk pikaajalise hooldusega versioon [34]. Versioonihalduskeskkonnas GitLab hallati tervet projekti ning koodi. Juurdepääsuõiguse koodi repositooriumile annab autor taotluse esitamisel[1]. Hilisemaks implementatsiooniks kirjeldati projekti lehel, kuidas programmi käivitada ning millega peaks koodi muutmisel arvestama. Näiteks loetleti *Issues* lehel kõik PDU tüübid, mida avatud lähtekoodiga DIS implementatsioon sisaldab.

3.4 Avatud lähtekoodiga DIS implementatsioonid

Üks tuntumaid avaliku lähtekoodiga DIS lahendusi on Open-DIS projekt, mida arendab USA-s asuv Mereväe aspirantuuri (NPS) Modeleerimise, Virtuaalsete keskkondade ja Simulatsioonide Instituut (MOVES). Esialgselt tekkis avatud lähtekoodiga DIS järele vajadus siis, kui instituut arendas mängumootorit Delta3D. Brutzman, Grant ja

[1] https://gitlab.cs.ttu.ee/sankuk/disprojekt

McGregor toovad põhjuseks järgneva: “Enamus tänastest C++ DIS rakendustest on olnud kaubanduslikud tooted või ettevõttesisestes projektides”. Open-DIS lahendustele kehtib ka BSD 3-osaline tarkvaralitsents[1], tänu millele võib koodi vabalt kasutada. [6, lk 1-2] Avaliku info kohaselt on Open-DIS koodi kasutanud peamiselt NPS, aga ka Kanada valitsus [33]. Käesolevas töös kasutatakse Open-DIS Java implementatsiooni koodirepositooriumi. Lisaks Open-DIS lahendusele on avatud lähtekoodiga ka projekt KDIS. Autor ei valinud mainitud repositooriumi, sest implementatsioon oli tehtud programmeerimiskeeles C++ [20].

3.5 Google Protocol Buffers

Protocol Buffers ehk lühidalt Protobuf on Google poolt arendatud meetod andmete serialiseerimiseks. Võrreldes märgistuskeelega XML, on Protobuf 3-10 korda väiksem ja 20 kuni 100 korda kiirem. [14] Esmalt on vaja teha *proto* fail, millega kirjeldatakse sõnumitüübid. Dokumentatsiooni Java juhendis on edasist protsessi kirjeldatud järgnevalt: “Sellest loob *protocol buffer compiler* klassi, mis implementeerib automaatset kodeerimist ja *protocol buffer* andmete parsimist efektiivse binaarse formaadiga. Genereeritud klass tagab *getter* ja *setter* meetodid *protocol buffer* väljade jaoks ning hoolitseb *protocol bufferi* kui elemendi lugemise ja kirjutamise detailide eest”. [16] Protobufi kasutamise põhjust kirjeldab Eesti kaitseväe küberväejuhatuse info- ja kommunikatsioonitehnoloogia keskuse arhitektuuri- ja arendussektsiooni projektijuht kapten Tarmo Aia järgnevalt: „Kaitseväe süsteemides on andmemahtude väikesena hoidmine üliolulise tähtsusega, sest kasutatavad sidekommunikatsiooni liinide mahud on tihtipeale väikesed ning lisaks pakub Protobuf võimaluse seda kasutada erinevate programmeerimiskeelte vahel, selletõttu on see ka Eesti kaitseväes laialdaselt kasutuses" (Isiklik kirjavahetus, 17.05.2020).

3.6 REST ja API

Andmekaitse ja infoturbe leksikoni kohaselt on REST (*Representational State Transfer*) tarkvaarenduses olev arhitektuuristiil [1]. Lahendus koosneb kliendi ja serveri poolest. Lihtsustatult kirjeldades asuvad serveris ressursid ning klient saab küsida sellelt

[1] https://opensource.org/licenses/BSD-3-Clause

andmeid. REST-i kohta kehtib kuus printsiipi. Näiteks nagu eelnevalt öeldud peab süsteem sisaldama klienti ning serverit. Samuti peab kogu suhtlus olema olekuvaba ehk server ei tohi salvestada kliendi sessiooni kohta infot. [23, lk 3, 5-6] API (*Application Programming Interface*) on rakendusliides, mis kujutab reeglite ja vahendite kogumit programmi suhtluseks [1]. API, mis allub REST-i põhimõtetele, kutsutakse *restful* API-ks [23, lk 5]. REST API kasutab andmete manipuleerimiseks HTTP (*HyperText Transfer Protocol*) päringuid. Neli kõige enimkasutavat toimingut on GET, PUT, POST ja DELETE. Näiteks andmete küsimiseks kasutatakse GET meetodit. [39, lk 18, 33]

Töös realiseeritakse REST teenus Java-põhise raamistikuga Spring Boot. Autor valis mainitud vahendi, sest sellega oli mugav ja lihtne eraldiseisvat rakendust ehitada. Nimelt on rakenduse püsti panemiseks vaja minimaalselt tegeleda konfigureerimisega. Kaitseväes kasutatakse REST arhitektuuri, sest see on standardne liides ning see võimaldab integratsiooni kõikide teiste süsteemidega.

3.7 HTTPS ja JWT

Üks ettemääratud nõuetest on turvaline andmevahetus (Lisa 1). Esiteks peab rakendus kasutama interneti kommunikatsiooni protokolli HTTPS, mis tagab suhtlemisel krüpteeritud infovahetuse. Krüpteerimist teostab SSL (*Secure Sockets Layer*) turbeprotokoll. [38]

Teiseks tuleb autentimist teha JWT ehk *JSON Web Token*-iga. JWT on tõend, mida kasutatakse identiteedi informatsiooni saatmiseks ning kontrollimiseks [24, lk 2]. Turvalisuse tagamiseks allkirjastatakse *token* salastatud *string*-iga või võetakse kasutusele avalik ning privaatvõti. JWT struktuur koosneb punktidega eraldatud päisest, *payload*-ist ehk sisust ning signatuurist. Päis kirjeldab *token*-i tüüpi ning allkirjastamise algoritmi. *Payload*-is asuvad *claims*-id ehk väited objekti kohta (näiteks *token*-i aegumisaeg). Viimaseks osaks on allkirjastamise teel tekkinud signatuur. Autentimise protsess on järgnev: kasutaja logib sisse ning kui logimine õnnestub genereeritakse JWT. Kui kasutaja üritab ligi pääseda teistele ressurssidele, siis antakse *token* päringuga kaasa ning rakenduse pooles kontrollitakse, kas tal on selleks juurdepääsuõigus [21]

3.8 Testid

Testimisel kasutatakse Spring Boot-i siseseid tööriistu ning testimisraamistiku nimega JUnit 4. REST-i päringuid kontrolliti platvormiga Firecamp, mis võimaldab hallata ning testida API-si. Alguses kasutati ka sarnast lahendust nimega Postman, kuid lõplikuks valikuks jäi Firecamp, sest erinevalt esimesest, toetas see Google Protocol Buffers formaati. Eesmärgi valideerimist kontrollitakse kindlate stsenaariumite testimisega. Neljas ning viies stsenaarium katab lõviosa rakenduse funktsionaalsusest.

Rakendus peab läbima järgmised teststenaariumid:

1. Kui kasutatakse ebaturvalist HTTP protokolli, siis tagastatakse juurdepääsuõiguse viga (Error 403 Forbidden).

2. Kui kasutatakse ebaturvalist HTTP protokolli ja sisestatakse valed sisselogimisandmed, siis tagastatakse juurdepääsuõiguse viga (Error 403 Forbidden).

3. Kui kasutatakse HTTPS protokolli ja sisestatakse valed sisselogimisandmed, siis tagastatakse juurdepääsuõiguse viga (Error 401 Unauthorized).

4. Kui kasutatakse HTTPS protokolli ja sisestatakse õiged sisselogimisandmed, siis tehakse POST päring koos PDU sõnumiga.

5. POST päringuga saadetud PDU sõnum kattub testandmetega.

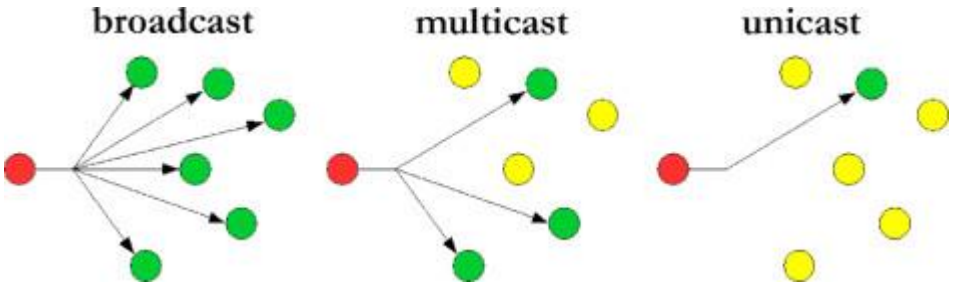
4 Tulemused

Antud peatükis esitatakse töö tulemused. Esimeses osas näidatakse, kuidas kasutatakse projekti Open-DIS implementatsioone ning teises osas tööriista Google Protobuf. Järgmine alapeatükk sisaldab REST API rakendamist. Viimasena selgitatakse HTTPS ja *token*-i lahendust ning tuuakse esile testide tulemused.

4.1 DIS implementatsioon

Protokolli DIS rakendamiseks kasutati projekti Open-DIS Java-põhise implementatsiooni klasse EspduReceiverNIO ja EspduSender. DIS-i jaoks tehti ka JSON konfigureerimisfail, kus määrati ära PDU pakettide saatmiseks vajalik port, IP-aadress ning PDU sõnumi maksimaalne suurus.

EspduSenderil põhineva klassi Sender-i roll on teha näidis PDU sõnum ning saata see konfigureerimisfailist loetud IP-aadressile. EspduSender toetab nii unicast-i, *broadcast*-i ja *multicast*-i. Jooniselt 6 on näha, kuidas *unicast*-i ehk üksikedastuse korral saadetakse pakett ühelt edastajalt ühele saajale, *broadcast*-is ehk eetriedastuses ühelt edastajalt kõikidele saajatele ning *multicast*-is ehk multiedastuses ühelt edastajalt teatud grupile saajatele [10].



Joonis 6. Üksik-, eetri- ja multiedastuse erinevused [36, lk 11].

Antud lahenduses toimus sõnumite saatmine üksikedastusena lokaalses võrgus, sest see oli ettemääratud (Lisa 1). Esiteks tegi Sender uue *Entity State* tüüpi PDU, mis on üks enam levinumatest tüüpidest, sest see annab infot tavalise objekti kohta. Täpsemalt loodi uus *Entity* tüüpi objekt, mis kirjeldas USA kolmanda generatsiooni tanki M1

Abrams. Järgmisena viidi objekt üle binaarsesse formaati, pandi datagrammi paketti ning saadeti varem konfigureerimisfailis defineeritud pordile ja IP-aadressile. Testimiseks ning kontrollimiseks korrati tegevust 50 korda ehk kokku saatis Sender 50 *Entity State* tüüpi paketti.

Autori poolt tehtud Receiver klass põhineb EspduReceiverNIO klassil, mille eesmärk on olla kuulaja. Kõigepealt määratleti andmete saamiseks *socket* ehk sokkel pordi ja IP-aadressiga. Nii kui Senderi poolt saadud datagramm jõudis kohale, võeti paketist info ning tehti selle põhjal uus PDU objekt. Viimasena tehti *postPdu* meetodiga serverile POST päring (lähemalt kirjeldatakse mainitud meetodit alapeatükis 4.3).

4.2 Google Protocol Buffers

Nagu alapeatükis 3.5 mainitud pidi esmalt looma faili, mis kirjeldaks *proto* faili ülesehitust. Struktuuri loomisel järgiti Protobufi dokumentatsiooni. Esmalt pidi määrama süntaksis *proto* versiooni, milleks oli antud lahenduses *proto3* (Joonis 7). Järgmisena täpsustati Java pakettide nimed ehk millisesse kausta ja mis nimega tekib *compiler*-i poolt genereeritud fail. *Proto* sõnum nimetati PduData-ks, sest see sisaldas PDU kohta saadud infot. Tagastama ainult PDU tüübi *string*-ina, sest tellija poolne Protobuf-i info formaat oli lõplikult paika panemata ning töö seisukohast lepiti kokku, et tehakse ainult implementatsiooni valideerimine ja edasine tegevus läheb lõputöö skoobist välja (Lisa 1). Järelikult kuulus ka *proto* sõnumi hulka ainult PDU tüüp. *Proto* struktuuri failis peab iga välja väärtus olema unikaalne number (numbrit kasutatakse binaarse formaadis identifitseerimiseks [15]). Seetõttu valis autor PDU tüübi taha kõige väiksema võimaliku väärtuse, milleks oli üks. Peale seda kasutati Protobuf-i *compiler*-it, et automaatselt genereerida *proto getter*-ite ja *setter*-ite klass.

```
syntax = "proto3";

package GeneratedProto;
option java_package = "GeneratedProto";
option java_outer_classname = "PduProto";

message PduData{
	string pdu_type=1;
}
```

Joonis 7. *Proto* faili ülesehitus.

4.3 REST API

REST API realiseerimiseks kasutati raamistiku Spring Boot ning tööriista RestTemplate. Spring Boot-is on võimalik meetodeid kirjeldada annotatsioonidega, mille tõttu on rakenduse püsti panemine lihtne. Näiteks kontrolleri määramiseks piisas klassi ette *@RestController* lisamisest.

Antud lahenduses kasutati POST päringut, mida kasutatakse ressursi loomiseks ning selle saatmiseks serverisse. POST-imise eest vastutas Receiver klassis olev meetod *postPdu* ning serveri kontroller võttis need vastu. Esmalt defineeriti *postPdu* meetodis *endpoint* ehk lõpp-punkt, kuhu POST päring saadetakse. Järgmisena loodi uus RestTemplate objekt. RestTemplate on Spring raamistiku klass, mis pakub erinevaid malle HTTP meetodite tegemiseks [43]. Samuti lisati sellele ka Protobuf-i *converter* ehk teisendaja, mille abil sai lugeda ja kirjutada Protobuf formaadis sõnumeid. Kasutades genereeritud *proto* faili meetodeid, loodi uus *proto* sõnum ning PDU tüübiks pandi Receiver-ist kätte saadud tüüp ehk *Entity State* tüüp (Receiver-ist on lähemalt räägitud alapeatükis 4.1). Viimasena lisati HTTP päringu *header*-isse ehk päisesse varasemalt genereeritud *token* ning koos *proto* sõnumiga tehti POST. Kontrolleris täpsustati, et POST-i vastuvõtmise jaoks mõeldud meetod ootab sisu Protobuf-i formaadis. Rakenduses määrati kaks kontrollerit: TestController PDU sõnumite saatmiseks ning JwtAuthenticationController autentimise jaoks (autentimisest on lähemalt peatükis 4.4).

4.4 HTTPS ja JWT implementatsioon

HTTPS protokolli rakendamiseks pidi esmalt tekitama turvasertifikaadi. Kuna käesolevas lõputöös arendati prototüüpi, siis piisas hetkel autori enda poolt tehtud sertifikaadist. Esiteks loodi sertifikaadihoidla *keystore*, mida kasutati kliendi või serveri

privaatsete sertifikaatide talletamiseks. Teiseks genereeriti käsureal uus sertifikaat, mille aegumisajaks oli 365 päeva, allkirjastamise algoritmiks RSA (Rivest–Shamir–Adleman) ning suuruseks 2048 bitti. *Keystore* koos autori poolt allkirjastatud sertifikaadiga lisati rakenduse ressursside kausta. Sätete failis defineeriti *keystore* andmefail ning selle juurdepääsuks mõeldud salasõna. Selleks et, POST meetod kasutaks samuti HTTPS protokolli, pidi konfiguratsiooni klassis uue RestTemplate-i loomisel lisama külge ka SSL valideerimise.

JWT ehk *JSON Web Token*-i kasutamise protsess põhineb lehekülje Javainuse juhendi põhjal ning on järgnev:

1. Tehakse POST kasutajanime ja salasõnaga */authenticate endpoint*-i
2. Kontrollitakse, kas POST päringuga tuli kaasa JWT
3. *Token*-i puudumisel see genereeritakse
4. Serveri pool kontrollitakse kasutajanime ja vastava salasõna olemasolu
5. Kui kasutaja puudub tagastatakse *error,* et kasutaja on registreerimata
6. Kui kasutaja on olemas ning valideeritud, genereeritakse automaatselt *token*
7. Antud prototüübi lahenduses salvestatakse *token* ajutisse faili
8. Receiver loeb failist *token*-i ning paneb selle PDU POST päringu *header*-isse.
9. PDU sõnumi POST päring saab toimida ainult juhul, kui *token* kehtib.

Protsessi käigus genereeriti *token*, mille kestvusajaks on 10 minutit. Kasutajate andmebaas oli antud prototüübis otse koodi kirjutatud ehk *hardcoded*. Kasutajate salasõna räsiti BCrypt räsifunktsiooniga. Rakenduse sätete faili pandi räsimise algoritmi jaoks mõeldud salasõna, mis on juhuslikult genereeritud numbritest ja tähtedest koosnev *string*. Samuti konfigreeriti rakendust selliselt, et volituseta kasutajale tagastatakse HTTP staatus 401.

4.5 Testide tulemused

Testide tegemiseks kasutati Java-le mõeldud tööriista JUnit4. Kõige olulisemaks peeti funktsionaalsuse ja integratsiooni teste. Nagu varasemalt mainitud, kasutati töö eesmärgi valideerimiseks teststsenaariume. Tabel 1 kirjeldab defineeritud testide elluviimist. Kaks esimest põhinesid protokollil HTTP. Nimelt testiti, kas HTTP ühendusega on võimalik serveri ressursside ligi pääseda. Näiteks kontrolliti, kas olemasolevat kasutajat saab autentida. Ülejäänud stsenaariumi testid kasutasid SSL protokolli. Esmalt testiti autentimise päringuid õigete ja valede sisselogimisandmetega. Järgmisena kontrolliti lahenduse põhifunktsionaalsust ehk kas õigete logimisandmetega ning *token*-iga saadakse teha POST päring PDU sõnumiga ning kas antud sõnum klapib Sender klassist teele pandud infoga. Lisaks teststsenaariumitele, tehti osaliselt teste ka Spring Boot-i konfiguratsiooni ning *token*-i tegemiseks mõeldud klassidele.

Tabel 1. Stsenaariumite testimine

1	Kasutatakse ebaturvalist HTTP protokolli, siis tagastatakse juurdepääsuõiguse viga (Error 403 Forbidden).	Loodi uus JSON objekt olemasoleva kasutaja infoga, mis saadeti POST päringuga /authenticate lõpp-punktile. Test tagastas HTTP 403 Forbidden vea, mis tähendas, et ressursile oli üle HTTP juurdepääs keelatud.
2	Kasutatakse ebaturvalist HTTP protokolli ja sisestatakse valed sisselogimisandmed, siis tagastatakse juurdepääsuõiguse viga (Error 403 Forbidden).	Loodi uus JSON objekt registeerimata kasutaja infoga, mis saadeti POST päringuga /authenticate lõpp-punktile. Test tagastas HTTP 403 Forbidden vea, mis tähendas, et ressursile oli üle HTTP juurdepääs keelatud.
3	Kasutatakse HTTPS protokolli ja sisestatakse valed sisselogimisandmed, siis tagastatakse juurdepääsuõiguse viga (Error 401 Unauthorized).	Loodi uus JSON objekt registeerimata kasutaja infoga, mis saadeti POST päringuga /authenticate lõpp-punktile. Test tagastas HTTP 401 Unauthorized vea, mis tähendas, et kasutajal puudusid õigused POST päringu tegemiseks.
4	Kasutatakse HTTPS protokolli ja sisestatakse õiged sisselogimisandmed, siis tehakse POST päring koos PDU sõnumiga.	Olemasoleva kasutaja info saadeti /authenticate lõpp-punktile. Test tagastas HTTP staatuse 201 Created. Pärast seda genereeriti kasutajale token. Loodi näidis PDU tüüp ning see koos token-iga sisestati Receiver-i postPdu meetodisse. Serveri POST meetod tagastas Protobuf-i formaadis sõnumi, millest loeti saadetud tüüp. Antud testi tulemuseks oli HTTP staatus 201 Created ning PDU tüüp kattus esmalt defineeritud testandmetega.
5	POST päringuga saadetud PDU sõnum kattub testandmetega.	Tehtud etapid olid samad, mis neljandas stsenaariumis. Peale POST päringut kontrolliti, kas PDU tüüp kattus esmalt defineeritud testandmetega.

5 Analüüs ja järeldused

Lõputöö eesmärk sai täidetud. Süsteem võtab DIS implementatsioonist vastu PDU informatsiooni ning edastab selle serialiseerituna (Protobufi formaadis) üle REST API liidese serveri poolde. Arenduses käigus täideti kaitseväe poolt püsitatud nõuded. Autori enda valikute puhul arvestati sellega, et valitud vahendid oleksid usaldusväärsed ning võimalusel valdkonnas standardsed. Nagu peatükis 2.3 mainitud, on lahendus autorile teadaolevalt unikaalne. Töö tulemusena valmis rakendus, mida saab lihtsalt implementeerida, sest see on eraldiseisev.

5.1 Kitsendused ja piirangud

Lahendus sisaldab kitsendusi ning piiranguid, sest tegemist on prototüübiga. Reaalsetes keskkonna tingimustes võivad tekkida näiteks jõudluse, võrgu või koormuse probleemid. Hetkel sisaldab rakendus sisse kirjutatud ehk *hardcoded* koodi (kasutajate andmebaas, räsimise algoritmi salasõna). Antud prototüübi andmebaasis on ainult üks kasutajanime ning salasõna paar. Rohkemate kasutajate registreerimiseks tuleks luua sisse- ja väljalogimis vaated ning kontrollerid.

5.2 Töö raskused

Algselt oli plaan kasutada protsessi *Test Driven Development* (TDD) ehk testimisel põhinevat arendust. DIS protokolli implementeerimisel katsetati mitme erineva lahendusega ja seejärel valiti nende seast sobivaim. Näiteks sai DIS sõnumite saatmiseks vähemalt viie erineva klassi vahel valida ning sama kehtis ka info vastuvõtmise kohta. Lisaks polnud ühegile mainitud klassile tehtud teste. DIS protokolli kasutatakse üsna kitsas valdkonnas ning kui seda rakendatakse, siis on enamikul juhtudel selle lähemalt uurimiseks vaja maksta. Näiteks VBS3 simulatsiooni jaoks on vaja osta aastane litsents, RedSim 2 nõuab ühele kasutajale mõeldud DIS PDU Recorder litsentsi eest 2750 dollarit [37]. Antud teema mõistmine ning selle kohta uurimine võttis aega DIS protokolli keerukuse ning eelnevalt mainitud põhjuste tõttu. Ajaliste piirangute pärast pidi autori õppimiskõver olema üsna järsk ning selle arvelt kannatasid testid. Kogu lahenduse töötamiseks pidi kõik kolm arhitektuuri plokki (Joonis 5) töötama samal ajal ning pideva katsetamise teel oli igale versioonile testide tegemine

ajamahukas. Testide tegemine ei õnnestunud algselt planeeritud mahus, aga tellija poolt esitatud funktsionaalsused said täidetud.

5.3 Lahenduse muutmine

Juhul kui tegemist poleks olnud prototüübiga, vaid tootmisesse mineva koodiga (*production code)* tuleks teha paar asja teisiti. Testandmed sai genereeritud Open-DIS andmete pealt, kuid selgus, et see protsess oli ajamahukas ning otstarbekam oleks olnud matkekeskusest andmed küsida, kuid lõputöö ajaraami tõttu polnud see enam võimalik. Sel juhul oleks olnud parem ettekujutus DIS sõnumite saatmisest ning vastuvõtmisest täpsemalt VBS3 simulatsioonis. Järgmisena kasutaks TDD meetodit nii nagu alguses oli planeeritud. Sisse- ja väljalogimise jaoks looks vaated ning vajalikud kontrollerid. Sisse kirjutatud koodi ehk *hardcoded* koodi asendaks meetodite või klassidega. Näiteks kasutajate andmebaasiks kasutaks andmebaasisüsteemi SQLite, sest selle jaoks pole serverit vaja püsti panna [45]. Rakenduse käivitamisel genereeriks uue ja unikaalse salasõna räsimise algoritmiks. Samuti ei salvestaks autor autentimise *token*-it ajutisse faili.

5.4 Alternatiivid

Alternatiivina saaks kasutada erinevaid Open-DIS implementatsioone, sealhulgas C-keele, Javascript-i või Python-i põhjalist repositooriumi. Teiseks oleks võimalik valida mitme Sender-i ja Receiver-i klassi vahel. Juba ainuüksi Open-DIS sisaldab vähemalt nelja näidisklassi PDU sõnumite saatmiseks. Need kõik edastavad PDU kohta infot, kuid kasutavad selleks erinevaid meetodeid. JWT info saaks salvestada faili asemel HTTP-küpsise sisse. RestTemplate asemel oleks võimalik kasutada Spring raamistiku klassi WebClient. Võrreldes RestTemplate-iga on WebClient *non-blocking* ehk REST-i päringu saatmisel ei pea ootama sellelt vastust, et rakenduse järgmist ülesannet täita [44]. Tasuline ja litsentseeritud alternatiiv oleks varasemalt mainitud API tööriistakomplekt VR-Link.

5.5 Töö õnnestumised

Arendamisel oli palju positiivseid momente. Spring Boot-i raamistiku kasutamine oli mugav ja lihtne, sest piisas annotatsioonide lisamisest ning suur osa konfigureerimisest

tehti automaatselt ära. Autori jaoks oli huvitav enda poolt allkirjastatud SSL sertifikaatide loomine, sest varasem kokkupuude nendega puudus. Töö käigus õppis autor rohkem tundma IntelliJ IDEA keskkonda ja Java programmeerimiskeelt (näiteks meetodid ning klassid, mida oli vaja üle võrgu suhtlemiseks). Protobufi kooskõlastamine REST-i ja RestTemplate-iga osutus alguses raskeks, kuid lõpus saadi kõik tööle tänu Protobufi formaadi teisendamisele.

5.6 Tulemuste usaldusväärsus

Lõputöö käigus tehti kindlaid samme selleks, et tulemused oleksid usaldusväärsed. Töös on kasutatud tööriistu ning süsteeme, mis on tarkvaraarenduses *de facto standard*, näiteks Java rakendustes on nendeks Spring raamistik ning REST-i realiseerimine. Samuti täideti kõik kaitseväe poolt püstitatud nõuded, v.a punkti 10 puhul ei suudetud täita nõuet 100% katvusega (Tabel 1).

Tabel 2. Nõuete realiseerimine.

1	Liidese realiseerimiseks kasutatakse REST arhitektuuri stiili.	Spring Boot raamistik, RestTemplate tööriist
2	Sõnumite serialiseerimiseks kasutatakse Google Protocol Buffers protokolli.	DIS sõnum pandi Protobufi formaati
3	Autentimiseks kasutatakse *token*-i lahendust.	Kasutati *JSON Web Token*-it
4	Autoriseerimiseks kasutatakse SSL tehnoloogiat.	Suhtlus üle HTTPS protokolli
5	Rakenduses kasutatakse JSON konfiguratsiooni faili, kus määratakse IP ja port.	Loodi Config.json fail vajalike väljadega
6	Tarkvara haldamiseks kasutatakse versioonihaldus keskkonda.	Versioonihalduseks kasutati GitLab keskkkonda
7	Protokolli DIS realiseerimiseks kasutatakse avaliku lähtekoodiga DIS implementatsioone.	DIS sõnumeid võttis vastu ning saatis Open-DIS klassid
8	DIS implementatsioonis tagastatakse ainult PDU tüüp stringi kujul.	Prototüübis saatis Sender klass DIS objekti tüübi väärtust
9	Sõnumite saatmine toimub üksikedastusena lokaalses võrgus	Edastuseks kasutati lokaalhosti
10	Töö valmidus valideeritakse testidega.	Valideerimiseks testiti nelja stsenaariumit
11	Testid kirjutatakse autori poolt tehtud koodile.	Testides kontrolliti rakenduse funktsionaalsust.

5.7 Rakenduse edasiarendus

Töö edasiarendusena peaks PDU sõnumis olema peale tüübi ka muid väärtuseid, näiteks asukoha koordinaadid. Lisada võiks vaated ning kontrollerid sisse- ja väljalogimiseks. Kasutajate info tuleks salvestada andmebaasi ning *token*-i kasutamisel väldiks faili salvestamist. Samuti peaks testima eetri- ja multiedastust, näiteks virtuaalsete arvutitega. Tellija poolt piisas esialgsest implementatsioonist, sest muidu oleks kogu töö olnud sel juhul suurema mahuga kui lõputöö.

5.8 Lahenduse kasutuselevõtt

Antud töö tulemusena valminud prototüüpi kasutatakse implementeerimiseks Eesti kaitseväe Küberväejuhatuses. Seal hakatakse rakendust modifitseerima, et viia see vastavusse kaitseväe süsteemide ja tingimustega. Loodud arendust rakendatakse simulatsioonisüsteemide andmete kasutamiseks koos reaalsete andmetega ja hilisemaks analüüsiks. Kuna tegemist on prototüübiga, on raske hinnata, kas lahendus tasub rahaliselt ära või mitte. Küll on aga võimalik järeldada, et antud lahenduse edasiarenduseks on olemas tööjõu ning tehnoloogiline ressurss [9].

6 Kokkuvõte

Käesolevas lõputöös realiseeriti militaarsimulatsiooni VBS3 asukoha edastusprotokolli DIS liidestust.

Probleem seisnes selles, et Eesti kaitseväel puudus liides, mis muudaks VBS3 objektide info loetavaks teistele kaitseväe süsteemidele.

Lõputöö eesmärgiks oli mainitud probleemile luua lahendus rakenduse kujul. Selle täitmiseks pandi paika süsteemi arhitektuur. Rakenduse tegemisel täideti kaitseväe poolt esitatud nõudeid. Prototüübi tegemisel kasutati avatud lähtekoodiga DIS implementatsioone ning Spring raamistikke. Eesmärgi valideerimiseks testiti viie eelnevalt defineeritud stsenaariumi läbimist.

Töö tulemusena arendati välja süsteem, mis võtab DIS protokollist vastu PDU sõnumi ning edastab selle Protobufi formaadis üle REST API liidese teistele süsteemidele. Tööd on võimalik edasi arendada. Antud lahenduse skoobis oli nõue tagastada ainult PDU tüüp, kuid peale selle saaks tagastada ka teisi väljasid, näiteks asukoha koordinaate.

Lahendus on autorile teadaolevalt unikaalne.

Kasutatud kirjandus

[1] AKIT (Andmekaitse ja infoturbe leksikon) [WWW] https://akit.cyber.ee/term/10843%20AKIT%20REST (11.05.2020)

[2] Arizona, Ft. Huachuca Joint Data Base Elements (JDBE). – *Fire PDU*. [WWW] The DIS Data Dictionary (07.05.2020)

[3] Bisimulations koduleht. – *Forest Fire Area Simulator, Vitrociset and Italian Forestry Corps*. [WWW] https://bisimulations.com/company/customer-showcase/forest-fire-area-simulator (05.05.2020)

[4] Bisimulations koduleht. – *Virtual Desktop Training & Simulation Host* https://bisimulations.com/products/vbs3 [WWW] (05.05.2020)

[5] Bohemia Interactive Simulations. – *What Is VBS3: Versatile Desktop Training & Simulation Software*, 2018. [Video] https://www.youtube.com/watch?v=ad_xFWtutNY (08.05.2020)

[6] Brutzman, D., Grant, J., McGregor, D. – *Open-DIS: an open source implementation of the DIS Protocol for C++ and Java*, 2008, 1-2. [Online] Dudley Knox Library (10.05.2020)

[7] Dahmann, S., J., Fuijmoto, M., R., Weatherly, M., R. – *The Department of Defence High Level Architecture, Proceedings of the 1997 Winter Simulation Conference, 1997, USA, GA, Atlanta, Detsember 7-10*. [Online] IEEE Xplore Digital Library (05.05.2020)

[8] Eesti Kaitsevägi. – *Kaitseväe akadeemia matkekeskuses toimus Kalevi jalaväepataljoni nooremallohvitseride matkeõppus*, 2019 . https://mil.ee/uudised/kaitsevae-akadeemia-matkekeskuses-toimus-kalevi-jalavaepataljoni-nooremallohvitseride-matkeoppus/ [WWW] (05.05.2020)

[9] Eesti Kaitsevägi. – *Küberväelased valmistasid terviseametile viiruseandmete infosüsteemi*, 2020 . https://mil.ee/uudised/kubervaelased-valmistasid-terviseametile-viiruseandmete-infosusteemi/ [WWW] (17.05.2020)

[10] EIK Wiki. – *OSadmin mõisted*. [WWW] https://wiki.itcollege.ee/index.php/OSadmin_m%C3%B5isted (13.05.2020)

[11] FAAC koduleht. – *FAAC's Military Division*. [WWW] https://www.faac.com/military/about/ (08.05.2020)

[12] FAAC koduleht. – *Realistic, Emotionally Vivid EMS Training Simulations*. [WWW] https://www.faac.com/simulation-training/public-safety/ambulance-rescue-driver-training/ (08.05.2020)

[13] Fitzsimmons, A., E., Fletcher, D. – *Beyond DoD: Non-Defense Training and Education Applications of DIS*, 1995, 1. [Online] IEEE Xplore Digital Library (05.05.2020)

[14] Google Developers. – *Developer Guide.* [WWW] https://developers.google.com/protocol-buffers/docs/overview (11.05.2020)

[15] Google Developers. – *Language Guide (proto3).* [WWW] https://developers.google.com/protocol-buffers/docs/proto3 (12.05.2020)

[16] Google Developers. – *Protocol Buffer Basics: Java*. [WWW] https://developers.google.com/protocol-buffers/docs/javatutorial (11.05.2020)

[17] Hill, F. – *The DIS protocol - new and improved. SpringSim '09: Proceedings of the 2009 Spring Simulation Multiconference, 2009, USA, CA, San Diego, Märts 22-27*. [Online] ACM Digital Library (05.05.2020)

[18] IEEE Computer Society. – *1278.1-2012 IEEE Standard for Distributed Interactive Simulation - Application Protocols*, 2012, 50. [Online] IEEE Xplore Digital Library (04.05.2020)

[19] JetBrains koduleht. – *What's New in IntelliJ IDEA 2020.1*. [WWW] https://www.jetbrains.com/idea/whatsnew/#java (10.05.2020)

[20] Jones, K. – *What Is KDIS*. [WWW] https://sourceforge.net/p/kdis/wiki/Main_Page/history (15.05.2020)

[21] Jwt koduleht. – *Introduction to JSON Web Tokens*. [WWW] https://jwt.io/introduction/ (12.05.2020)

[22] Kauchak, M. – *STE-CFT: Accelerating Training Modernization*, 2019. [WWW] https://militarysimulation.training/articles/ste-cft-accelerating-training-modernization/ (08.05.2020)

[23] Lange, K. – *The Little Book On Rest Services*, Kopenhaagen, 2016, 3, 5-6 [E-Raamat] https://www.kennethlange.com/ (11.05.2020)

[24] M. Jones, B. Campbell, C. Mortimore. – *JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants*, 2015, 2. [Online] IETF Datatracker andmebaas (12.05.2020)

[25] McGregor, D. – *Example DIS Applications*, 2018. [WWW] https://github.com/open-dis/DISTutorial/wiki/example-dis-applications (05.05.2020)

[26] McGregor, D. – *PDU Headers*, 2018. [WWW] https://github.com/open-dis/DISTutorial/wiki/PDU-Headers (07.05.2020)

[27] MST Magazine. – *2019 MS&T Awards - Industry's Choices* – MST Magazine, nr 4/6, 2019, 38-39. [E-ajakiri] https://bluetoad.com/publication/?m=34018&i=617676&p=2&ver=html5 (08.05.2020)

[28] Murray, R. – *DIS or Cut Bait: the Push to the First Draft*, 2020. [Online] SISO Digital Library (04.05.2020)

[29] Mussprat, A. – *VBS STE: The future of simulated training*, 2018. [WWW] https://www.defenceiq.com/defence-technology/articles/vbs-ste-bohemia-simulations-on-creating-a-cloud-based-synthetic-training-environment (05.05.2020)

[30] Noseworthy, J., R. – *The Test and Training Enabling Architecture (TENA) — Supporting the Decentralized Development of Distributed Applications and LVC Simulation, 2th 2008 IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, 2008, BC, Kanada, BC, Vancouver, Oktoober 27-29*, lk 259. [Online] IEEE Xplore Digital Library (07.05.2020)

[31] NSC koduleht. – *UBVT*. [WWW] https://www.nsc.co.uk/training/ubvt/ (08.05.2020)

[32] Office of Technology Assessment, The United States Congress. – *Distributed Interactive Simulation of Combat*, 1995, 2-3. [Online] Kaitse Tehnilise Teabe Keskus (DTIC) https://apps.dtic.mil/docs/citations/ADA336692 (05.05.2020)

[33] Open-DIS koduleht. – *An open source implementation of the Distributed Interactive Simulation protocol*. [WWW] http://open-dis.org/ (11.05.2020)

[34] Oracle koduleht. – *Oracle Java SE Support Roadmap*. [WWW] https://www.oracle.com/java/technologies/java-se-support-roadmap.html (10.05.2020)

[35] Peets, I. – *Virtuaalne lahinguruum – reaalsusest reaalsem 3* – Sõdurileht, nr 2, 2018, 53, 55. [E-ajakiri] https://dea.digar.ee/cgi-bin/dea?a=d&d=AKsodur201804.2.23 (05.05.2020)

[36] Ramos, M., A., Masiero, P., Penteado, R., Braga, T., V., R., – *Extending statecharts to model system interactions*, 11, 2015. [WWW] https://www.researchgate.net/publication/280610232_Extending_statecharts_to_model_system_interactions (13.05.2020)

[37] RedSim2 koduleht. – *DIS PDU Recorder*. [WWW] http://www.redsim.com/products/dis-pdu-recorder.html (14.05.2020)

[38] Rodriguez, M. – *HTTPS Everywhere: Industry Trends and the Need for Encryption*, 2018. [WWW] https://www.researchgate.net/publication/325835233_HTTPS_Everywhere_Industry_Trends_and_the_Need_for_Encryption (12.05.2020)

[39] Ruby, S., Richardson, L., Amundsen, M. – *RESTful Web APIs*, 2013, 18, 33. [Online] O'Reilly's for Higher Education (15.05.2020)

[40] Ryan, P., Ross, P., Oliver, W. – *Distributed Interactive Simulation Revisited: Capabilities of the Revised IEEE Standard*, 2018, 1-2 [WWW] https://www.researchgate.net/publication/328418763_Distributed_Interactive_Simulation_Revisited_Capabilities_of_the_Revised_IEEE_Standard (04.05.2020)

[41] Ryan, P., Zalcman, L. – *The DIS vs HLA Debate: What's in it for Australia?,* 2003. [WWW] https://www.researchgate.net/publication/268059885_The_DIS_vs_HLA_Debate_What's_in_it_for_Australia (07.05.2020)

[42] SISO Product Development Group. – *Guide for: DIS Plain and Simple*, 2009, 26-27, 32. [Online] SISO Digital Library (04.05.2020)

[43] Spring Framework API dokumentatsioon. – *Class RestTemplate*. [WWW] https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html (13.05.2020)

[44] Spring Framework API dokumentatsioon. – *Interface WebClient*. [WWW] https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/reactive/function/client/WebClient.html (17.05.2020)

[45] SQLite koduleht. – *About SQLite*. [WWW] https://www.sqlite.org/about.html (14.05.2020)

[46] Straßburger, S. – *Overview about the High Level Architecture for Modelling and Simulation and Recent Developments*, 2006, 5. [WWW] https://www.researchgate.net/publication/251422110_Overview_about_the_High_Level_Architecture_for_Modelling_and_Simulation_and_Recent_Developments (07.05.2020)

[47] VT MAK koduleht. – *VR-Link: HLA & DIS Simulation Networking* [WWW] https://www.mak.com/products/link/vr-link#the-vr-link-api-is-stable (17.05.2020)

Lisa 1 – Kaitseväe poolt püstitatud nõuded

1. Liidese realiseerimiseks kasutatakse REST arhitektuuri stiili.
2. Sõnumite serialiseerimiseks kasutatakse Google Protocol Buffers protokolli.
3. Autentimiseks kasutatakse *token*-i lahendust.
4. Autoriseerimiseks kasutatakse SSL tehnoloogiat.
5. Rakenduses kasutatakse JSON konfiguratsiooni faili, kus määratakse IP ja port.
6. Tarkvara haldamiseks kasutatakse versioonihaldus keskkonda.
7. Protokolli DIS realiseerimiseks kasutatakse avaliku lähtekoodiga DIS implementatsioone.
8. DIS implementatsioonis tagastatakse ainult PDU tüüp stringi kujul.
9. Sõnumite saatmine toimub üksikedastusena lokaalses võrgus
10. Töö valmidus valideeritakse testidega.
11. Testid kirjutatakse autori poolt tehtud koodile.