VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2019 Kristián Bučko

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE MODELU OMG/DDS A ANALÝZA KOMUNIKACE
IMPLEMENTATION OF OMG/DDS MODEL AND COMMUNICATION ANALYSIS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Kristián Bučko**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jiří Pokorný**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Kristián Bučko ***ID:*** 195287
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Implementace modelu OMG/DDS a analýza komunikace

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce bude navrhnout reálný model datově orientované komunikace (publish-subscribe). Nejprve bude provedena analýza dostupných implementací DDS (Data Distribution Service) a OMG (Object Management Group). Práce bude zaměřena na implementaci Connext DDS od společnosti RTI, která bude otestována na mikropočítačích Raspberry Pi případně na dalších UNIX systémech za použití virtualizačních nástrojů. Následně bude provedeno porovnání zabezpečené a nezabezpečené komunikace. Dále bude provedeno hodnocení bezpečnosti a zranitelností pomocí dostupných SW (např. Kali Linux).

DOPORUČENÁ LITERATURA:

[1] SCHLESSELMAN, Joseph M.; PARDO-CASTELLOTE, Gerardo; FARABAUGH, Bert. OMG data-distribution service (DDS): architectural update. In: Military Communications Conference, 2004. MILCOM 2004. 2004 IEEE. IEEE, 2004. p. 961-967.

[2] CALVO, Isidro, et al. Towards a OMG DDS communication backbone for factory automation applications. In: Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on. IEEE, 2011. p. 1-4.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 27.5.2019

Vedoucí práce: Ing. Jiří Pokorný
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

V bakalárskej práci je predstavenie OMG DDS špecifikácie, teoretické porovnanie troch najrozšírenejších implementácií OpenDDS, Vortex Opensplice a taktiež RTI Connext. V práci sú taktiež popísané dôležité rozdiely medzi dátovo zameranou publish-subscribe a objektovo zameranou client-server (napr. CORBA) komunikáciou, tiež aj využiteľnosť týchto dvoch typov v real-time systémoch. Implementácia RTI Connext je prakticky demonštrovaná na rôznych typoch v scenároch nezabezpečenej a zabezpečenej komunikácie, kde sú samotné komunikácie odchytené vo wiresharku. V zachytených prenosoch je zhodnotená bezpečnosť. Boli upravené kódy publishera a subscribera podľa potreby merania, ktorým je analyzovaná latencia komunikácie medzi rôznou veľkosťou správ. Pre spracovanie časových razítok a priemerovanie hodnôt bol vytvorený skript v jazyku python. Nakoniec je spravené kompletné porovnanie týchto scenárov a ich grafické zobrazenie.

KĽÚČOVÉ SLOVÁ

Zabezpečená, nezabezpečená, komunikácia, analýza, autentifikácia, DDS, OMG, RTPS, RTI, OpenSSL

ABSTRACT

In bachelor´s thesis there is an introduction of OMG DDS specification, theoretical comparison of three most widespread implementations OpenDDS, Vortex Opensplice and RTI Connext, too. Important differences between data-focused publish-subscribe and object-focused client-server (e.g. CORBA) communication are also described in the thesis, as well as the usefulness of these two types in real-time systems. Implementation of RTI Connext is practically demonstrated on various types in scenarios of unsecured and secured communication where the communications themselves are caught in wireshark. The safety is evaluated in captured transfers. The publisher and subscriber codes were adjusted according to a need of measurement by which a communication latency between various sizes of a message is analyzed. A script in Python programming language was created for processing of timestamps and averaging values. A complete comparison of these scenarios and their graphical visual display are done in the end.

KEYWORDS

Secure, unsecure, communication, analysis, authentication, DDS, OMG, RTPS, RTI, OpenSSL

BUČKO, Kristián. *Implementace modelu OMG/DDS a analýza komunikace*. Brno, 2019, 44 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Jiří Pokorný

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Implementace modelu OMG/DDS a analýza komunikace" vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno . . . .

podpis autora

POĎAKOVANIE

Rád by som poďakoval vedúcemu bakalárskej práce pánovi Ing. Jiřímu Pokornému, za jeho odborné rady, ochotu, trpezlivosť a čas strávený pri konzultovaní tejto práce.

Brno ....

podpis autora

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popsaný v tejto bakalárskej práci bol realizovaný v laboratóriách podporených projektom SIX; registračné číslo CZ.1.05/2.1.00/03.0072, operačný program Výzkum a vývoj pro inovace.

Brno ....

podpis autora

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

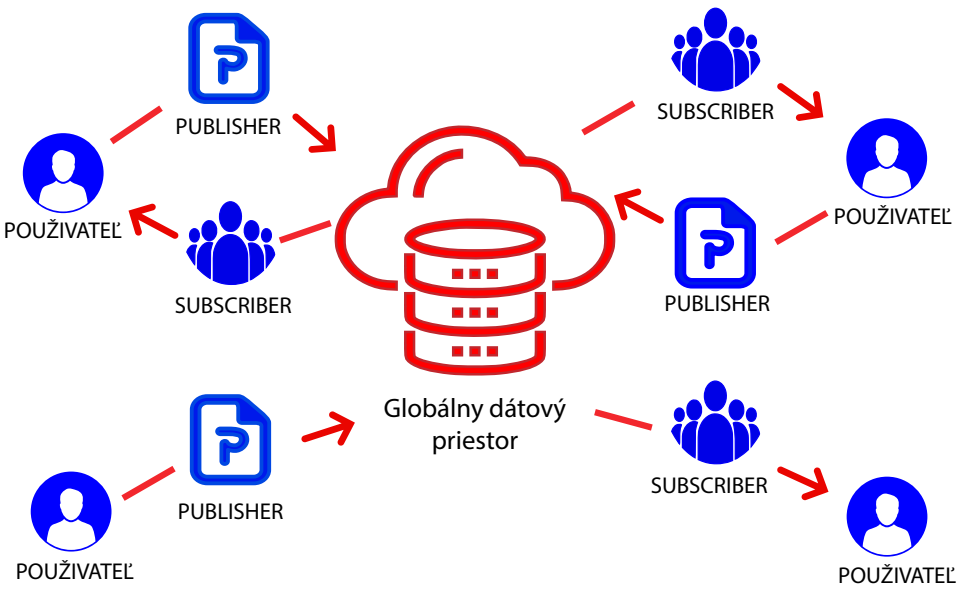
Obsah

Zoznam obrázkov

Úvod

DDS (služba distribúcie dát) je middleware protokol a štandard API pre prenos dát pomocou modelu publish-subscribe zo skupiny Object Management Group (OMG). Existujú rôzne opensource a komerčné implementácie štandardu DDS, ktoré poskytujú API a služby pre distribúciu dát [1].

Real-Time aplikácie (aplikácie v reálnom čase) často potrebujú byť distribuované do viacerých výpočtových uzlov. Dôvodmi sú distribúcia výpočtového výkonu do miest, kde je potrebný, zjednodušenie designu, manažment alebo údržba aplikácie. Dôležitou súčasťou všetkých distribuovaných aplikácií je komunikácia medzi komponentmi aplikácie na rôznych uzloch. V prípade real-time aplikácií je komunikácia podmienená dočasnými obmedzeniami ako napríklad deadliny. Často musia byť aplikácie odolné proti chybám a musia podporovať nadbytočnosť (redundancy) v ich architektúre. Ďalšou bežnou požiadavkou je dynamický charakter aplikácie kde sú uzly resp. komponenty spojené alebo odstránené z aplikácie v danom čase chodu (run-time). Tieto, často protikladné požiadavky sťažujú design a implementáciu komunikačnej časti aplikácie. Preto ľudia často tvoria aplikácie na rôznych komunikačných middleware platformách, ktoré majú na starosti komunikáciu.



Obr. 1: Dátovo zameraný aplikačný model Publish–Subscribe

Tradičné middleware platformy ako CORBA poskytujú transparentný prístup ďalekým objektom cez volania na diaľku (remote method calls). Keď aplikácia vy-

zve metódu na vzdialenom objekte, middleware automaticky zosériuje metódové parametre a pošle žiadosť do cieľového procesu, kde je objekt umiestnený. Výpočet sa deje vzdialene, potom sú výsledky poslané naspäť. Aj keď to zjednodušuje tvorbu distribuovaných aplikácií, existuje mnoho aplikácií, ktoré nemôžu byť efektívne implementované na tomto request-response modeli (žiadosť-odpoveď modeli) [2].

Aplikácie, ktorých prevádzka je dátovo-orientovaná, to znamená, že časť akcie alebo výpočtu je vykonaná, keď sú dáta pripravené, by boli vhodné na riešenie middleware platformami, ktoré bezproblémovo spravujú distribúciu dát od producentov k spotrebiteľom. Takéto aplikácie sú často navrhované podľa dátovo-zameraného Publish Subscribe modelu (DCPS), kde middleware vytvára koncept „globálneho dátového priestoru", ktorý je prístupný všetkým interesovaným aplikáciám (vid. obr. 1). Písanie aplikácií podľa tohto modelu má mnoho výhod. Najväčšou je asi to, že komunikačné požiadavky sú špecifikované aplikáciami v deklaratívnom spôsobe a middleware riadi výmenu dát automaticky podľa deklarácií [3].

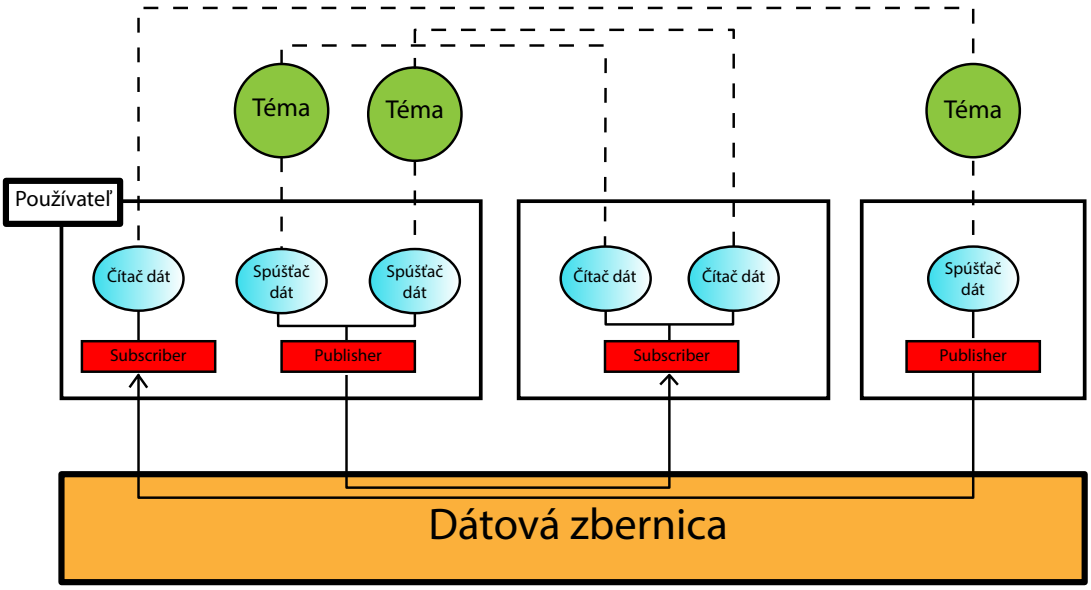
1 Teoretický úvod

V tejto časti práce prebehlo teoretické preskúmanie a analyzovanie dátovo a objektovo zameranej architektúry a OMG DDS modelov z ktorých boli najznámejšie implementácie vybrané a porovnané, taktiež popis najdôležitejšieho protokolu RTPS pre dátovo orientovanú komunikáciu publish-subscribe. Dôraz bol kladený na porovnanie komerčnej a voľnošíriteľnej (opensource) implementácie, hlavne čo sa týka typu komunikačných modelov, transportu a latencie.

1.1 Dátovo zamerané vs. objektovo zamerané (Data-Centric vs. Object-Centric)

Základom k porozumeniu potreby tohto nového štandardu je náhľad na fundamentálne architekturálne rozdiely medzi dátovo zameranou a objektovo zameranou komunikáciou v distribuovanom real-time systéme.

DDS je prirodzeným protikladom k existujúcemu, už dobre známemu CORBA modelu. V DDS pristupujeme k metódovým žiadostiam na vzdialených objektoch cez rozhranie, ktoré je definované v Interface Descriptor Language (IDL) (popisný jazyk rozhrania). V CORBA sú dáta komunikované nepriamo cez argumenty v metódových žiadostiach alebo cez ich spätné hodnoty.



Obr. 1.1: Zjednodušený diagram prvkov v rámci DDS domény.

Avšak vo veľkom počte real-time aplikácií je komunikačná predloha často modelovaná ako dátovo zameraná výmena, kde aplikácie publikujú stream dáta, ktoré sú potom prístupné vzdialeným aplikáciám, ktoré o dáta majú záujem. Hlavným bodom záujmu je efektívna distribúcia dát s minimálnou extra aktivitou naviac a potreba zväčšenia na stovky alebo tisícky odberateľov robustným spôsobom [3].

1.1.1 Publish-Subscribe protokol

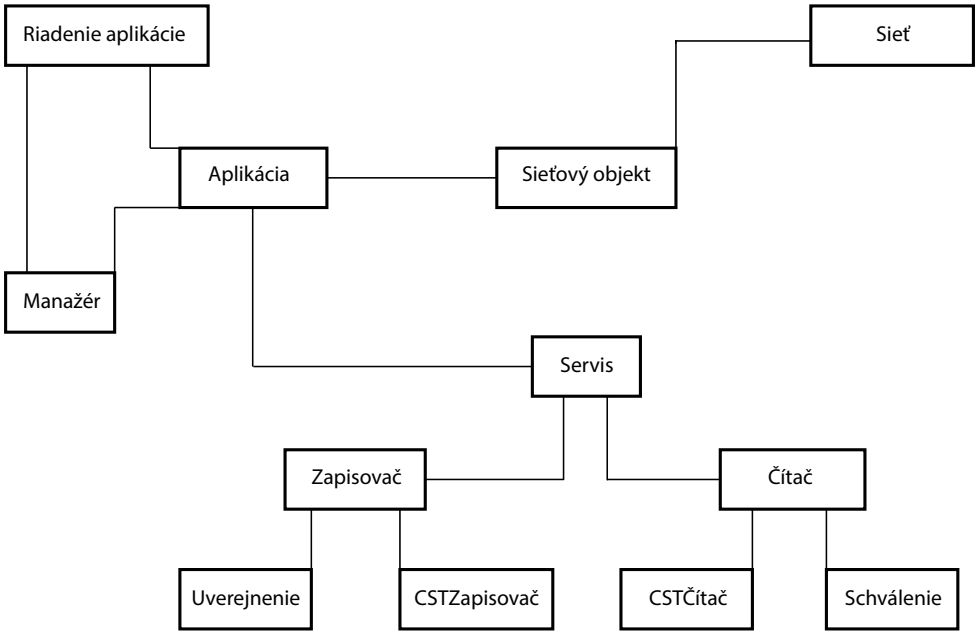
Klasický Publish-Subscribe model bol protokolom RTPS doplnený o časové parametre a parameter určujúcí režim doručovania dát. Distribučný model Publish-Subscribe využíva medzivrstvu k vytvoreniu komunikačných uzlov:

- Publisher, uzol, ktorý sa stará o produkciu dát. Publisher pomocou medzivrstvy registruje názov *(topic)* a type *(type name)* dát, ktorá bude publikovať *(issue)*. Data sú produkované tak rýchlo, ako to umožňuje technológia použitá k ich získavaniu (napr. čidlo vydáva data každých 10ms). Neexistuje žiadny parameter nastavujúci rýchlosť publikovania dát. Na sieti sa môžu vyskytovať Publisheri, ktorí produkujú zhodné data (napr. data záložných čidiel) [4].

- Subscriber, uzol, ktorý odoberá data. Subscriber pomocou medzivrstvy registruje názov *(topic)* a type *(type name)* dát vždy jedinej publikácie, ktorú chce odoberať. Subscriber definuje dva časové parametre: *minimum separation* a *deadline*. Minimum separation je časový interval meraný od poslednej prijatej správy. Po dobu trvania intervalu niesu prijímané nové data. Tento časový úsek slúži Subscriberovi k spracovaniu predošlej prijatej publikácie. Po uplynutí tejto doby Subscriber začne opäť prijímať data [4].

1.1.2 DDS Quality of Service Policy Set

Primárnym diskriminátorom medzi DDS a inými postupmi je unikátny Quality of Service (QoS) podmienkový súbor (policy set). Tento súbor obsahuje 21 kľúčových parametrov, ktoré umožňujú dynamickú, laditeľnú a škálovateľnú real-time sieť. Avšak tento veľký set tiež poukazuje na možnosť zmätku v tom, ktoré nastavenia umožnia optimálny výkon v danej konfigurácii. Aj keď majú špecifické implementácie DDS už prednastavené nastavenia (alebo niekedy nemusia plne podporovať všetky DDS QoS), je to stále v štádiu aktívneho výskumu.



Obr. 1.2: Objektový model, ktorý je základom protokolov RTPS.

1.1.3 Porovnanie s distribuovanou zdieľanou pamäťou (Distributed Shared Memory)

Ďalšou požiadavkou real-time aplikácií je potreba kontroly QoS vlastností, ktoré ovplyvňujú predvídateľnosť, overhead a použité zdroje. Distributed Shared Memory je klasickým modelom, ktorý ponúka dátovo zamerané výmeny. Avšak tento model je zložitý a „neprirodzený“ na efektívne používanie cez internet.

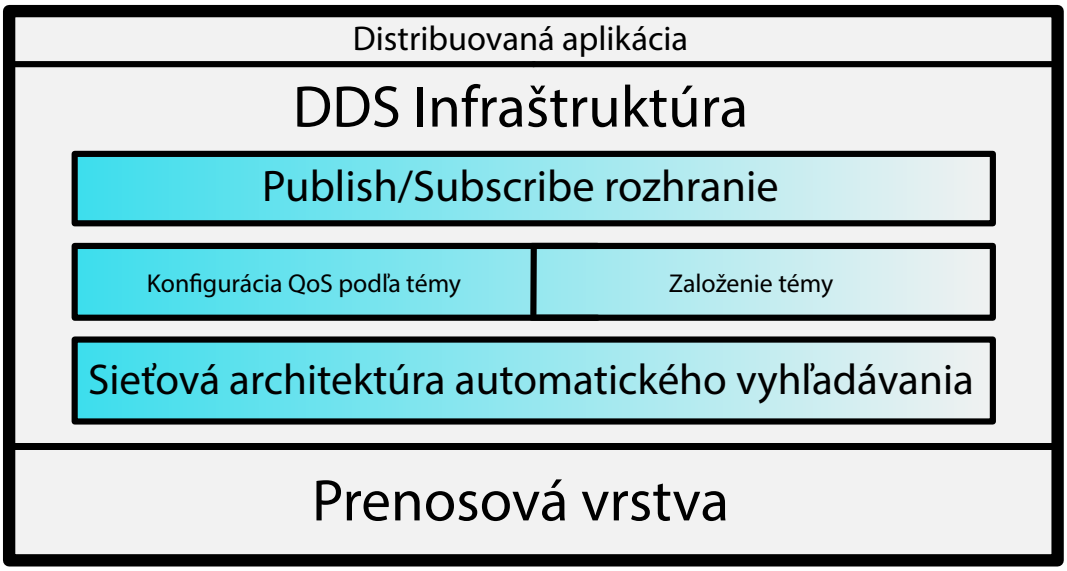
Tým pádom sa vo veľkom počte real-time aplikácií stal populárny iný model – Data-Centric Publish Subscribe (DCPS). Aj keď existuje niekoľko komerčných a in-house (privátnych) vyvinutých aplikácií, ktoré ponúkajú tento typ možnosti, do dnes sa nevytvoril žiaden generalizovaný štandard dátovej distribúcie. Tým pádom, žiadne bežné modely priamo nepodporujú dátovo zameraný systém na výmenu informácií. OMG DDS je pokusom vyriešiť túto situáciu. Špecifikácia tiež definuje operácie a QoS atribúty, ktoré tieto objekty podporujú a rozhrania, ktoré môže aplikácia použiť, aby bola upozornená na zmenu dát alebo čakať na špecifické zmeny.

1.1.4 Porovnanie s HLA RTI

HLA, tiež známe ako OMG Distributed Simulation Facility, je štandard z IEEE aj OMG. Popisuje dátovo zameranú publish-subscribe facilitu a dátový model. OMG špecifikácia je IDL špecifikácia a môže byť mapovaná na viacerých transportoch.

Špecifikácia popisuje niektoré z požiadavok dátovo zameraného publish-subscribe: aplikácia využíva publish-subscribe rozhranie na interakciu s middlewareom, obsahuje dátový model a podporuje obsahovo zamerané (content-based) odbery.

HLA dátový model podporuje špecializačnú hierarchiu ale nepodporuje agregačnú hierarchiu. Súbor definovaných typov sa nemôže časom rozšíriť. Dátové prvky sú nepísané a neusporiadané, sú to čisté sekvencie oktetov. HLA tiež neponúka obecné QoS facility.



Obr. 1.3: Infraštruktúrna vrstva DDS, ktorá umožňuje rôznym typom aplikácií, aby so sebou komunikovali.

DDS vytvára veľmi jednoduchú architektúru dátovej komunikácie, umožňuje veľmi komplexné dátové vzory. Témy dovoľujú endpoint nódom tzv. koncovým uzlom ich oddelenie od seba, nódy/uzly môžu vstúpiť a opustiť distribuovanú aplikáciu dynamicky. DDS je dátovo zameraný – všetky QoS parametre môžu byť zmenené cez správu. Táto správová konfigurabilita je kľúčom k podpore komplexných dátových komunikačných vzorov.

1.2 Porovnanie implementácií

Pre najlepšie pochopenie DDS implementácie si porovnáme opensource framework od firmy Vortex a spomínaný decentralizovaný model RTI. Aj keď OpenSplice vo svojich počiatočných verziách podporoval iba broadcast a multicast transporty [5], novšie verzie podporujú aj unicast transport. Dôvodom je efektívne zameranie sa

na WAN medzispoje (interkonekcie). OpenSplice funguje na federovanom modeli. Z pohľadu vyvíjania aplikácie to umožňuje jasnú separáciu aplikácií, ktoré bežia v oddelenom užívateľskom prostredí, od konfiguračných a komunikačných detailov DCPS. Na druhej strane, z komunikačného uhlu pohľadu federovaná architektúra dovoľuje data bundling (dátové zoskupovanie), ktoré OpenSplice automaticky využíva. To môže ale spôsobovať problémy. DCPS daemon spôsobuje potenciálny komunikačný zádrhel/prekážku pri vysokých dátových rýchlostiach. Na kompenzáciu tohto problému OpenSplice tiež podporuje decentralizovaný model, aj keď je to menej optimizované a nie je to východiskové nastavenie.

Obr. 1.4: Decentralizovaná architektúra – Zabudované vlákna na zvládnutie komunikácie, spoľahlivosť, QoS atď...

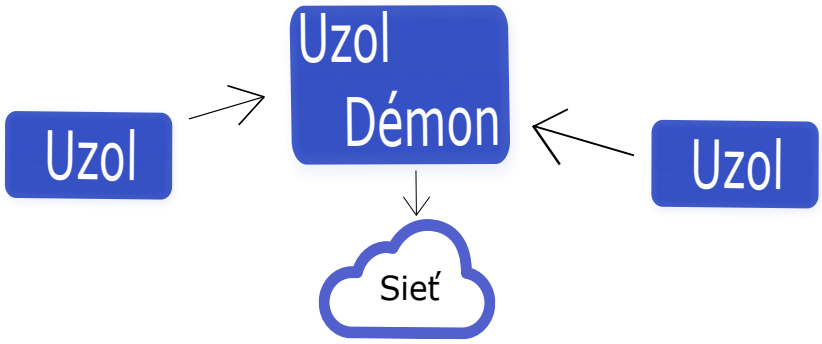
RTI podporuje unicast a multicast transport, ale nepodporuje broadcast. Hlavným rozdielom od OpenSplice je distribuovaný architektúrny model. RTI pracuje na decentralizovanej architektúre, kde sú aplikácie sebestačné (osobitný daemon nie je potrebný), tým pádom je latencia znížená. Spomínaný problém, ktorý sa vyskytuje v OpenSplice je v RTI nemožný. Hlavnou nevýhodou RTI je ale to, že decentralizované implementácie sťažujú využívanie optimizácií dátového zoskupovania medzi viacerými DDS aplikáciami, ktoré pracujú na jednom uzle. To je hlavným dôvodom, prečo táto možnosť nie je vo východiskových nastaveniach RTI. Dôsledkom toho je zníženie možných benefitov škálovateľnosti/škálovania (scalability), predovšetkým pre výmenu krátkych dátových sekvencií. RTI narozdiel od OpenSplice automaticky podporuje štandardný protokol ako jediný podporovaný protokol.

Obe systémy podporujú všetky komunikačné modely. Obe taktiež majú peer-to-peer prístup. Hlavným rozdielom je distribuovaná architektúra adoptovaná na výmenu dát. OpenSlice funguje na federovanom modeli a automaticky podporuje dátové zoskupovanie. RTI používa decentralizovaný model a nepodporuje automatické zoskupovanie. Môže byť ale pridané rozšírenie pre dátové zoskupovanie [6]. Tento rozdiel relevantne ovplyvňuje výkon, predovšetkým pri vysokých dátových

rýchlostiach a ak je DDS podpora použitá v situáciách zdrojovej saturácie/zdrojového presýtenia (resource saturation).

Obr. 1.5: Federovaná architektúra – Samostatný démonový proces na zvládnutie komunikácie, spoľahlivosti, QoS atď...

Federované modely ľahšie umožňujú lokálne optimizácie, napríklad môžu zlepšiť systémový výkon za výmeny malých dát. Avšak, v niektorých prípadoch, prídavný overhead môže byť problém, napríklad ak je overhead potrebný na vykonanie lokálnych operácií dátového zoskupovania. Centralizované modely majú tendenciu horšieho výkonu, ak sú ďaleko od nasýtenia/saturácie, ale majú lepší výkon pri vysokých dátových rýchlostiach a veľkých správach. Vo všeobecnosti prezentujú viac predvídateľný výkonový trend [7].



Obr. 1.6: Centralizovaná architektúra – Jeden samostatný démonový proces pre doménu.

1.2.1 Enterprise Architect

Na súčasnom trhu môžeme nájsť celú radu kvalitných produktov pre prácu s jazykom UML, špecialne s UML diagramom. Väčšinou sa jedná o veľmi rozsiahle a

profesionálne produkty, pre ich úspešné a efektívne používanie je potrebné absolvovať školenie či pomerne dlhé štúdium. Výsledkom nasadenia týchto systémov však býva rapídne urýchlenie práce, najmä prvotného návrhu systému. Jedným z týchto nástrojov je Enterprise Architect [8].

Všeobecné informácie

Jedná sa o nástroj používaný pro modelovanie s pomocou UML, ktorý je vyvíjaný austrálskou spoločnosťou Sparx Systems. Program podporuje platformy Linux a Windows. Enterprise Architect umožňuje podporiť a zjednodušiť celú fázu vývoja software, od definovania žiadostí na systém, cez design až po prípravu testovania. Program sa dodáva v niekoľkých verziách, či už trial verzia alebo platené profesionálne a korporátne verzie. Ich odlišnosti spočíva najmä v miere podpory tímovej spolupráce.

Enterprise Architect sa vyznačuje podporou modelov, medzi nich patrí Business Process Model, Class Model, Use Case Model, Sequence Model, Activity Model a Component Model. Nástroj vytvára výstupy vo formáte RTF pre dokumentáciu, ale aj vo formáte XMI, ktorý je určený pre budúcu spoluprácu s ostatnými programami [8].

Popis funkcií

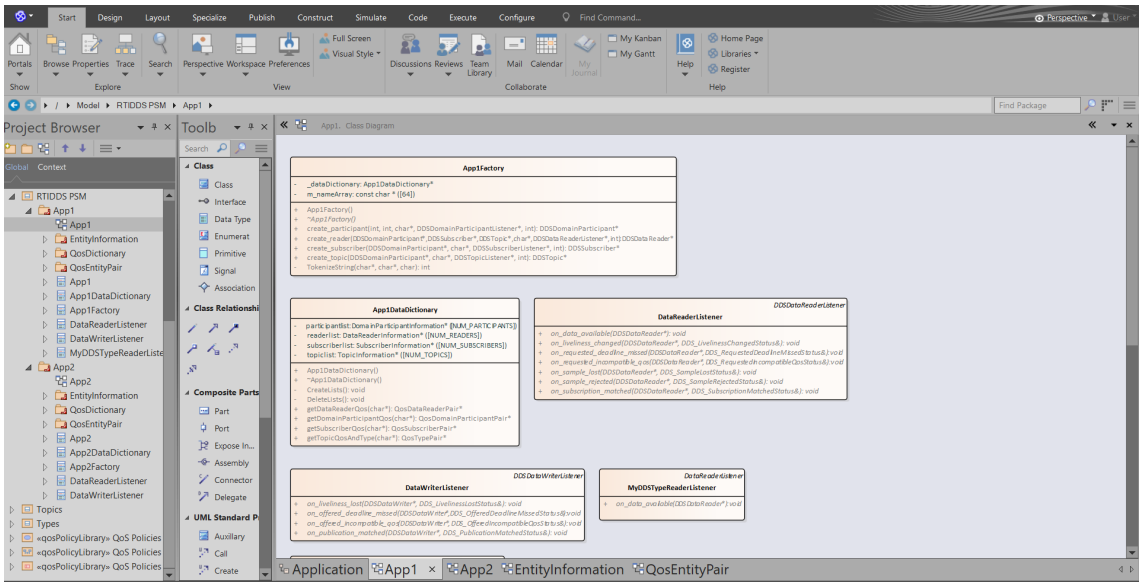
Bolo už spomenuté, že Enterprise Architect má za úlohu zjednodušiť fázu vývoja software. Teraz sa zameriame na podrobnejší popis jednotlivých funkcií a nástrojov. Hlavnou úlohou je modelovanie, pričom tento nástroj podporuje tvorbu diagramov UML, takže diagramové správanie a štruktúru diagramov. Ďalšími funkciami sú modelovanie obchodných procesov a dátové modelovanie, ktoré umožňuje vytvorenie logického a fyzického dátového modelu. Do modelovania je zaradené i mapovanie štruktúry modelov.

Enterprise Architect ponúka možnosť automatického generovania kódu (*Code engineering*), do toho patrí popredné inžinierske kódovanie (*Forward code engineering*) a spätné inžinierske kódovanie (*Reverse code engineering*). *Forward code engineering* umožňuje generovať zdrojový kód z už vytvorených modelov. Bohužiaľ pri vývoji sa *forward code engineering* príliš nevyužíva, pretože jeho použitie je podmienené synchronizáciou modelov, čo je pre zdroje veľmi náročné. Enterprise Architect umožňuje generovať zdrojový kód v celej rade objektovo orientovaných jazykov, konkrétne C, C++, Java, C#, PHP, Visual Basic a Delphi. *Reverse code engineering* umožňuje získať triedny diagram alebo fyzický dátový model z naimplementovaných tried alebo z vytvorenej databázy. Používajú sa rovnaké programovacie jazyky ako u *forward code engineeringu*.

Framework Enterprise Architect podporuje hneď niekoľko rôznych typov testovania, ako Unit testy, integračné testy, systémové testy či akceptační testy [8].

Pre túto prácu je tiež veľmi dôležité, že Enterprise Architect podporuje celú radu rôznych mechanizmov pre import a export modelov reprezentovaných štandardom XMI. Táto vlastnosť umožňuje používateľom spracovávať informácie vytvorené v iných nástrojoch a prenášať informácie o modeloch medzi Enterprise Architect a ďalšími nástrojmi.

Pre lepšiu predstavu je nástroj Enterprise Architect zobrazený na obr. 1.7.



Obr. 1.7: Ukážka nástroja Enterprise Architect

1.2.2 Framework RTI Connext DDS

RTI Connext DDS je prvá platforma, ktorá je rýchla, škálovateľná a odolná na pripojenie k softvéru navrhnutá pre náročné požiadavky Industrial Internet of Things (IIoT). Poskytuje QoS s nízkou latenciou a v reálnom čase, potrebné na monitorovanie a riadenie procesov. Poskytuje nepretržitú dostupnosť a bezpečnosť nevyhnutnú pre kritické systémy. Je kompatibilný so štandardom DDS, ktorý podporuje interoperabilitu a otvorenú architektúru a znižuje náklady na životný cyklus [9].
Connext DDS obsahuje bohatú sadu možností optimalizovaných pripojení pre systémy IIoT.

- Správy knižníc zjednodušujú vývoj distribuovaných aplikácií pomocou rozhrania API na vysokej úrovni pre fronty publish-subscribe, request-reply a point-to-point.

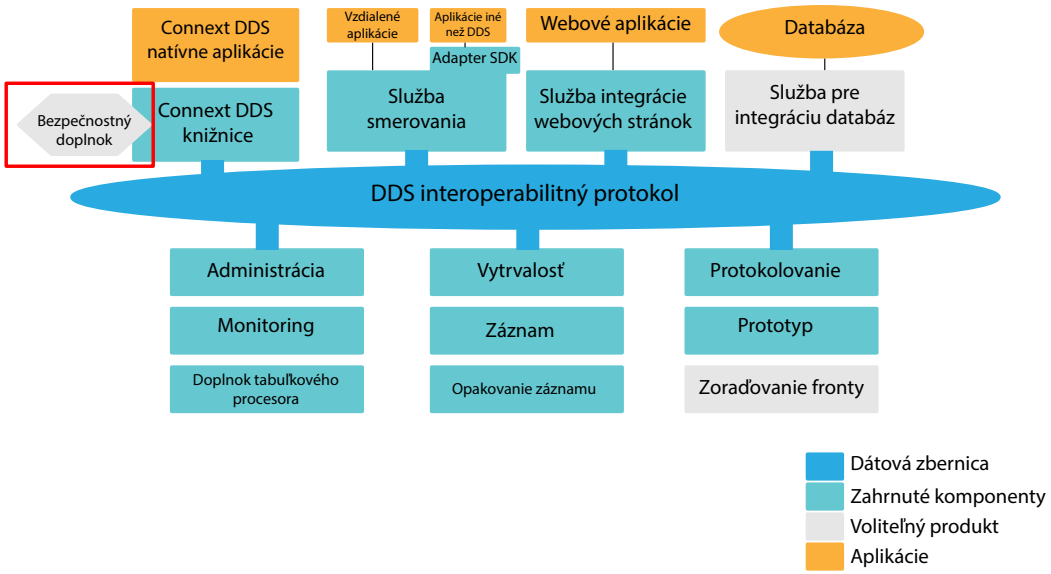
- Rámec adaptérov s prispôsobiteľnými príkladmi uľahčuje integráciu nemodifikovaných existujúcich aplikácií a zariadení.
- Výkonné nástroje urýchľujú integráciu systému, testovanie a ladenie.

Architektúra prostredia

Connext DDS je navrhnutý tak, aby riešil požiadavky na výkon, rozsah a odolnosť IIoT. Jeho architektúra je úplne decentralizovaná. Aplikácie sa automatický objavujú a komunikujú peer-to-peer. Na rozdiel od tradičných IT-centrických správ a integračných riešení, Connext DDS nevyžaduje žiadne centralizované sprostredkovateľské správy a služby. To umožňuje jednoduché umiestnenie a nasadenie pri odstraňovaní chýb a jednotlivých porúch [10].

Vzhľadom na dlhé životné cykly priemyselných systémov je zariadenie Connext DDS veľmi vhodné na postupné prijímanie. Poskytuje škálovateľnú, spoľahlivú a bezpečnú integračnú zbernicu, ktorá spája nové a existujúce aktíva. V prípade nových aplikácií, ako je napríklad analýza, predstavuje jednotnú API, ktorá oddeľuje logiku aplikácií od rozhraní, protokolov a dátových modelov špecifických pre zariadenie [9, 4].

Praktická časť tejto práce bude zameraná na zabezpečenie DDS komunikácie, takže hlavne na Security Plugin, ktorý môžeme vidieť na obr. 1.8.



Obr. 1.8: Ukážka sady nástrojov na vývoj a integráciu distribuovaných aplikácií.

Vlastnosti

Tento nástroj zjednodušuje aplikačnú a integračnú logiku s inovatívnou. Namiesto výmeny správ si softvérové komponenty zdieľajú dátové objekty. Aplikácie pracujú priamo na týchto objektoch (vytvárať, čítať aktualizácie a vymazávať). Vývojári sa nemusia zaoberať rozhraniami na odosielanie správ alebo sieťovými rozhraniami. Connext DDS spracováva podrobnosti o distribúcií a správe údajov vrátane serializácie a riadenia cyklu [9].

- Oddelenie - výrobcovia údajov sú nezávislí voči počtu spotrebiteľov a druhu spracovania, ktoré robia. To umožňuje pridanie a zmenu komponentov bez ovplyvnenia tých, ktoré už boli nasadené.
- Jednoduchá integrácia - definovaná dátovým modelom - nevyžaduje znalosť implementácie komponentov a nemusí sa robiť spätné konfigurovanie protokolov a správ.
- Robustnosť - framework udržiava zdieľaný stav systému. Neskoré a opätovne pripojené aplikácie sa automatický synchronizujú s aktuálnym stavom. To zaručuje, že aplikácie majú konzistentný pohľad na svet aj v dynamických a veľkých prostrediach.

1.2.3 OpenDDS

OpenDDS využíva pripojiteľnú transportnú architektúru, ktorá umožňuje prenos údajov prostredníctvom transportných implementácií aplikačného vývoja. Koncepčne je architektúra navrhnutá zo štandardu *Pluggable Protocols* od spoločnosti TAO. OpenDDS v súčasnosti podporuje prenosy TCP a UDP point-to-point, ako aj nespoľahlivé a spoľahlivé multicast prenosy.

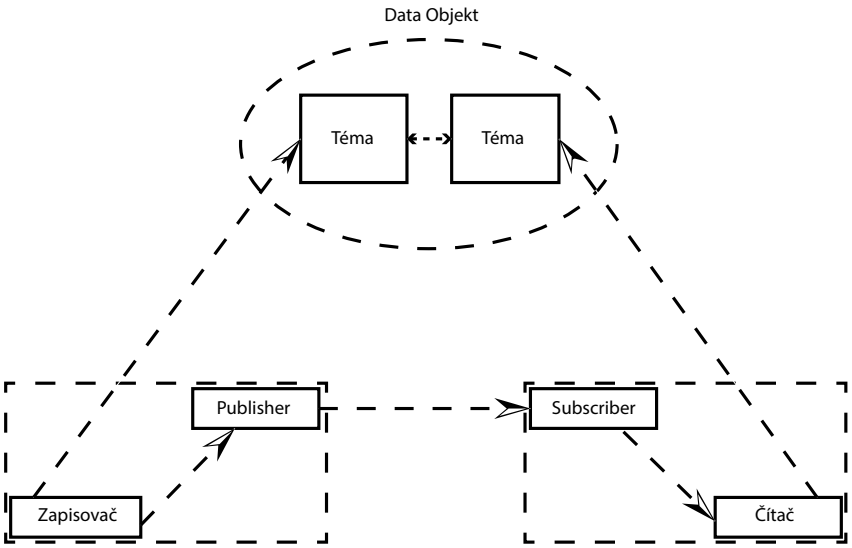
Táto architektúra s možnosťou pripojenia umožňuje používateľovi DDS optimalizovať inštaláciu DDS na základe požadovaného transportu homogénnej alebo heterogénnej povahy nasadenia aplikácie. Tieto voľby možno vykonať bez ovplyvnenia samotného kódu aplikácie.

Samotný kód je generovaný špecializovaným kompilátorom OpenDDS IDL. Jediný samostatný proces DCPS Information Repository funguje ako centrálny sprostredkovateľ, združujúci všetkých účastníkov (publisherov a subscriberov). OpenDDS používa CORBA model pre komunikáciu s procesom *DCPSInfoRepo*. Prenos údajov medzi publisherom a subscriberom je priamy medzi všetkými procesmi tejto implementácie. OpenDDS vytvára svoje vlastné vlákna, ktoré vznikajú pri odosielaní alebo prijímaní DDS dát [11].

Základné komponenty systému

Medzi hlavné základné komponenty tejto implementácie patria témy, publisher, subscriber, zapisovač (writer), čítač (reader). Na obrázku 1.9 je jasne vysvetlená štruktúra [12].

- Témy sú informácie o jednom type údajov a distribúcii a dostupnosti vzoriek.
- Publisher má na starosti kontrolu a obmedzenia toku dát zo zapisovača (DataWriter).
- Subscriber má na starosti kontrolu a obmedzenia toku dát z čítača (DataReader).
- Publisher môže mať neobmedzený počet zapisovačov.
- Subscriber môže mať neobmedzený počet čítačov.
- Zapisovač môže mať len jednu tému.
- Čítač môže mať len jednu tému.
- Téma môže mať viacero zapisovačov a čítačov.



Obr. 1.9: Jednoduchý koncepčný pohľad

Dátový tok

Aplikácia na vydavateľskej strane iniciuje tok údajov zapísaním hodnoty údajov do zapisovača. Zapisovač publikuje vzorky a pošle ich do súvisiaceho procesu. Každý priradený účastník odovzdá získanému vzorku svoje data, ktoré sú spojené s odosielaním. Tok sa ukončí, keď aplikácia na strane odberateľa načíta údaje z čítača.

Politika kvality služby (QoS) riadi tok údajov prostredníctvom systému. Téma, čítač, zapisovač, publisher a subscriber majú všetky politiky QoS. QoS na strane

publishera, zapisovača a témy kontrolujú údaje na strane odoslania. QoS na strane subscribera, čítača a témy kontrolujú údaje na strane príjemcu [12].

1.2.4 Vortex OpenSplice

OpenSplice DDS je implementácia z rady OMG DDS, ktorá sa vyznačuje vysokým výkonom a vysokou dostupnosťou, extrémnou škálovateľnosťou a predvídateľnosťou. OpenSplice DDS architektúra, je postavená okolo niekoľkých základných konceptoch napríklad efektívne využívanie zdieľaných zdrojov, ako je pamäť a vytváranie sietí, aby sa zaručila správna škálovateľnosť. Riadenie týchto zdrojov je zabezpečené riadením QoS. Na zabezpečenie maximálneho determinizmu slúži architektúra *pluggable-service*, ktorá je implementovaná v základnej funkcionalite DDS [1].

QoS politika

OpenSplice DDS podporuje prístup založený na údajoch, kde je na začiatku informačný systém, ovplyvňovaný príslušnou QoS politikou, ktorý je ďalej zdieľaný medzi viacerými stranami. Táto federovaná architektúra je bežná v existujúcich a plánovaný „koaličný“ vývojoch, kde viaceré strany spoločne implementujú celkový bojový systém. OpenSplice DDS poskytuje informačnú kostru, na ktorej sú všetky tieto aplikácie a je preto zodpovedná za poskytovanie správnych informácií každej aplikácii v správnom čase [13].

Sektor využitia

OpenSplice DDS sa využíva v oblasti obrany, ale jeho používanie tiež neustále rastie v iných oblastiach, ako je doprava, telekomunikácie a financie. Napríklad v kontexte Air Traffic Control (ATC) a v oblasti Air Traffic Management (ATM). OpenSplice DDS bol vybraný ako middleware na distribúciu letových údajov v organizácií CoFlight, čo je ďalšia generácia Európsky spracovateľ letových údajov. Všeobecne, OpenSplice DDS implementácia je vhodná technológia pre aplikačné domény, ktorá vyžaduje bohatú podporu politík QoS a vysoko výkonných a spoľahlivých štandardov založených na komerčných riešeniach [13].

1.3 Typy šifrovania

V dnešnej dobe sa cez internet a cez iné komunikačné médiá prenáša veľa dôležitých a tajných informácií. Je veľmi nevyhnutné tieto informácie a dáta chrániť. V tejto časti hovoríme o princípoch a základnom rozdelení šifrovania informácií. Šifrovanie

sa vo všeobecnosti delí na symetrické a asymetrické a my si ukážeme ako sa tieto dve metódy využívajú v praxi.

1.3.1 Symetrické šifrovanie

Symetrické šifrovanie je postup, ktorým jednoznačne zašifrujeme správu M (Message) pomocou kľúča K s (väčšinou) pevne danou dĺžkou na zašifrovaný text T, pričom zo zašifrovaného textu T dostaneme pôvodnú správu M len za podmienky, že poznáme pri šifrovaní použitý kľúč. Symetrické šifrovanie sa skladá z dvoch častí, zašifrovanie (Encryption) a dešifrovanie (Decryption), pričom platí:

$$\left|\begin{array}{l|c} \mathbf{E(M,\ K)\ =} & \mathrm{T} \\ \mathbf{D(T,\ K)\ =} & \mathrm{M} \end{array}\right|$$

pričom E a D je u väčšiny algoritmov rovnaká funkcia (teda používame rovnaký postup na šifrovanie aj dešifrovanie). Príkladom symetrického šifrovania je DES (Data Encryption Standard).

1.3.2 Asymetrické šifrovanie

Problém so symetrickým šifrovaním je v prenose kľúča. Kľúč K sa totiž musí preniesť cez nejaké médium. To bola v minulosti jedna z najväčších priorít medzinárodnej špionáže. Už vôbec nebolo možné kľúč preniesť cez elektronický kanál, ktorý je veľmi ľahko odpočúvateľný. Fyzický prenos je na druhej strane veľmi pomalý. Asymetrické šifrovanie tento problém rieši veľmi efektívne. Asymetrické šifrovanie je séria postupov, pri ktorých jednoznačne premeníme text T1 na text T2 pomocou kľúča Kn (n=1,2). Skladá sa z dvoch častí. Prvá časť (šifrovanie - encryption) premení text M na text T pričom použije kľúč K1 (väčšinou označovaný ako verejný kľúč - public key). Druhá časť (dešifrovanie - decryption) premení text T na text M, pričom sa použije kľúč K2 (väčšinou označovaný ako súkromný kľúč - private key). V zásade platí, že z K1 sa žiadnym matematickým postupom nedá získať K2. Súkromný kľúč K2 je kľúč, ktorý vlastní len človek, ktorému je správa určená. K1 je verejný kľúč, ktorý môže vlastniť ktokoľvek (daná osoba ho teda môže poskytovať na stiahnutie na internete). Text M zašifrovaný pomocou kľúča K1 sa teda dá dešifrovať len za pomoci kľúča K2, ktorý má len človek, ktorému je správa určená (z toho vyplýva, že text T na text M nemôže dešifrovať ani ten, kto ho zašifroval, pretože nemá súkromný kľúč K2, potrebný na túto operáciu). V skratke:

$$\left|\begin{array}{l|c} \mathbf{E(M,\ K1)\ =} & \mathrm{T} \\ \mathbf{D(T,\ K2)\ =} & \mathrm{M} \\ \mathbf{K2\ !=} & \mathrm{f(K1)} \end{array}\right|$$

Posledný riadok teda hovorí, že neexistuje funkcia f, ktorá ako argument dostane K1 a vráti hodnotu K2 [14].

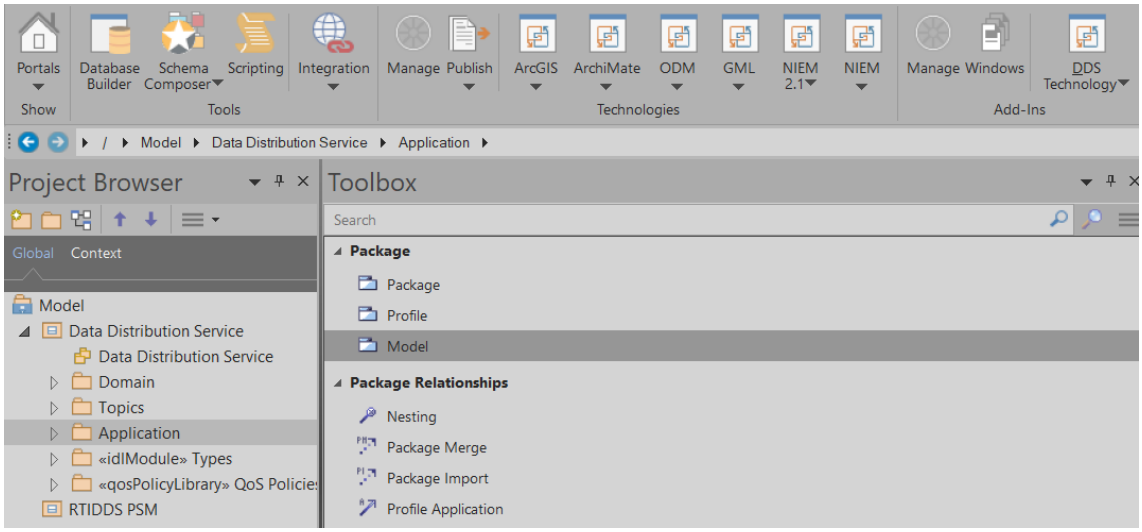
2 Praktická časť

V tejto časti je predvedené prepojenie DDS s Enterprise Architect a popísaný problém ktorý vznikol. Ďalej sa už pracovalo výhradne s implementáciou RTI Connext od firmy RTI. Prebehlo implementovanie softvéru, generovanie kódov pre nezabezpečenú a zabezpečenú aplikáciu, kde prebehlo otestovanie a analýza pomocou wiresharku. V ďalšej časti prebehlo meranie odozvy medzi aplikáciou publishera a subscribera, vytvorenie skriptu pre spracovanie nameraných výsledkov, hodnotenie bezpečnosti v zabezpečenej aplikácií a ukážka porovnanie latencie meraných prenosov.

2.1 Prepojenie DDS s Enterprise Architect

Spoločnosť Sparx Systems prináša softvérovým inžinierom low-cost vysokovýkonné UML modelovanie. V tejto časti práce si ukážeme výkon technológie MDG pre DDS a navrhneme aplikáciu na distribúciu dát v reálnom čase.

Po vytvorení projektu je potrebné načítať DDS technológiu, ktorú vykonáme v Addins Menu. Po tomto kroku môžeme vidieť, že EA je pre tento projekt povolený pre DDS a že paleta nástrojov obsahuje len príslušné konštrukcie DDS.



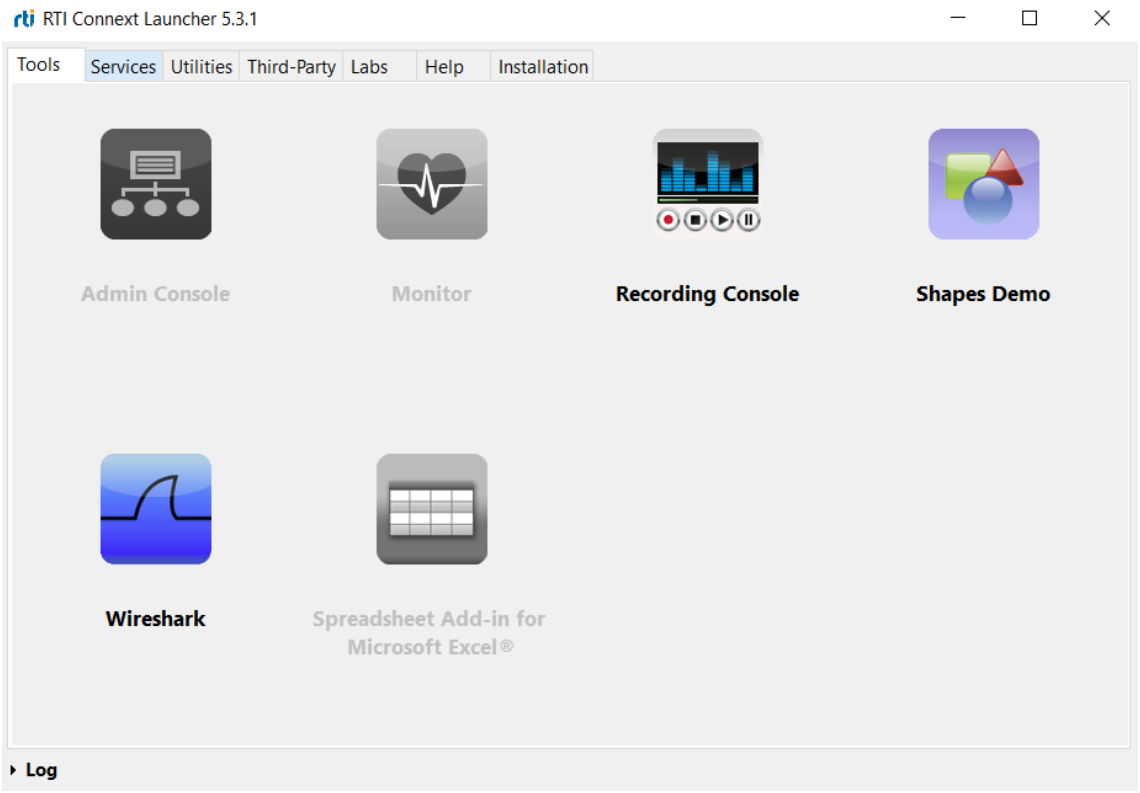
Možnosti správania vlastníctva QoS nám umožňujú automaticky vytvárať vlastnosti QoS pre každú z DDS entít, ktoré vytvoríme v našej aplikácii. Umožňuje tiež zobraziť vlastnosti QoS ako oddelenia v našom UML modeli. Možnosti DDS General riadia základné správanie technológie MDG. Keďže máme nový projekt DDS, musíme vytvoriť knižnicu pravidiel QoS, v ktorej som určil predvolené informácie

o QoS pre naše subjekty. Môžete vidieť, ako sú tu definované dáta QoS, ako hodnota označená - predvolená hodnota je prevzatá z knižnice QosPolicy. Tieto hodnoty môžu byť prepísané našou QosProperty.

Po rozbehnutí aplikácie vznikol problém v kompilácií, ktorý nešlo vyriešiť, keďže MDG DDS modul je už dlhšiu dobu neaktualizovaný, preto bol dôraz kladený na implementáciou DDS modulu od firmy RTI pomocou frameworku RTI Connext, ktorý bude popísaný a prakticky predstavený v nasledujúcej časti.

2.2 Implementácia pomocou RTI

Po poskytnutí akademickej licencie od firmy RTI som sa rozhodol prakticky overiť šifrovanú a nešifrovanú komunikáciu pomocou aplikácie RTI Connext obr. 2.1 s využitím programu Wireshark, ktorý je protokolový analyzér a paketový sniffer.



Obr. 2.1: Ukážka aplikácie RTI Connext Launcher

2.2.1 Knižnica OpenSSL

OpenSSL je softvérová knižnica pre aplikácie, ktoré zabezpečujú komunikáciu prostredníctvom počítačových sietí proti odpočúvaniu alebo potrebu identifikovať stranu

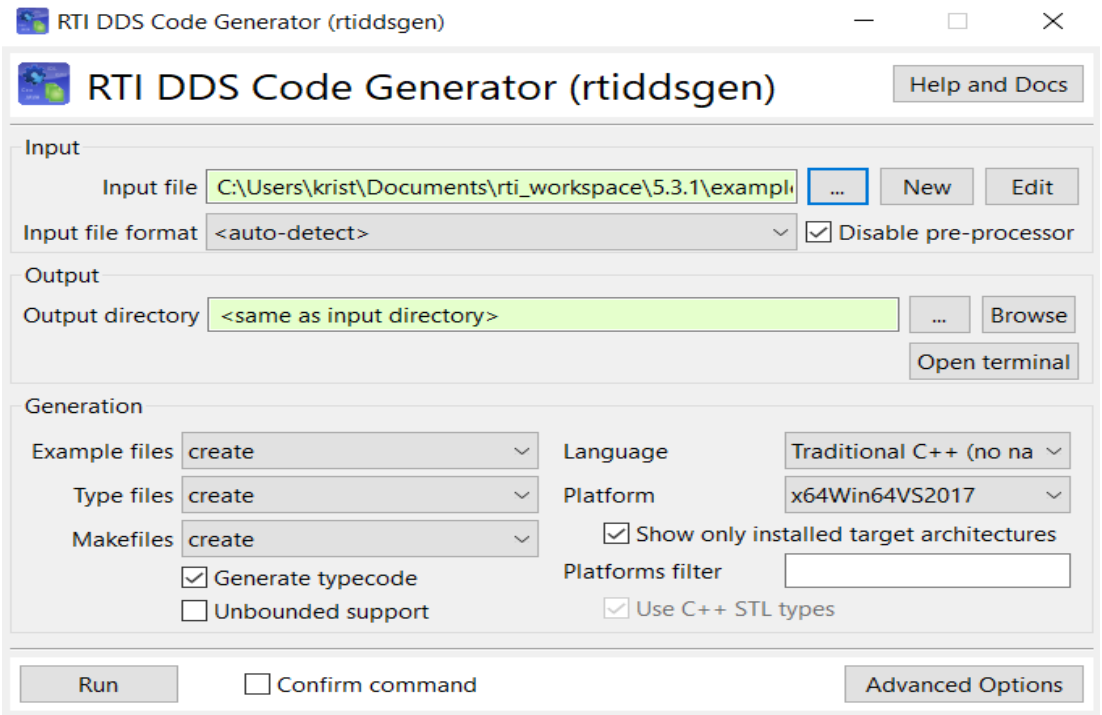
na druhom konci. Je široko používaná na webových serveroch na internete. Táto knižnica je nevyhnutná pre šifrovanú komunikáciu Publish – Subscribe.

OpenSSL obsahuje implementáciu protokolov SSL a TLS s otvoreným zdrojovým kódom. Knižnica jadra, napísaná v programovacom jazyku C, implementuje základné kryptografické funkcie a ponúka rôzne kryptografické operácie (generovanie kľúčov, podpisovanie dokumentov). Pomocou výmeny týchto dokumentov a kľučov si strany určia ako bude komunikácia vlastne vyzerať. K dispozícii sú balíky umožňujúce používanie knižnice OpenSSL v rôznych jazykoch.

Verzie sú k dispozícií pre väčšinu operačných systémov Unix (vrátane Solaris, Linux, MacOS, QNX, rôznych open-source BSD operačných systémov) a Microsoft Windows [15]. Pre našu šifrovanú ukážku bola použitá knižnica *openssl-1.0.2n* modifikovaná firmou RTI.

2.2.2 Generovanie kódov

Generátor kódov vytvára kódy potrebné pre definovanie a registráciu dátového typu používateľa RTI Connext DDS. Pri generovaní súborov je nutné poskytnúť opis dátových typov, konkrétne v súboroch IDL a XML. V práci sa konkrétne menila veľkosť dátového typu pre veľkosť správy na 1 MB z počiatočných 128 B. Je to nevyhnutný krok k tomu, aby bol zabezpečený plynulý prenos väčších správ medzi publisherom a subscriberom bez chýb. Pre predstavu vid. obr. 2.2.

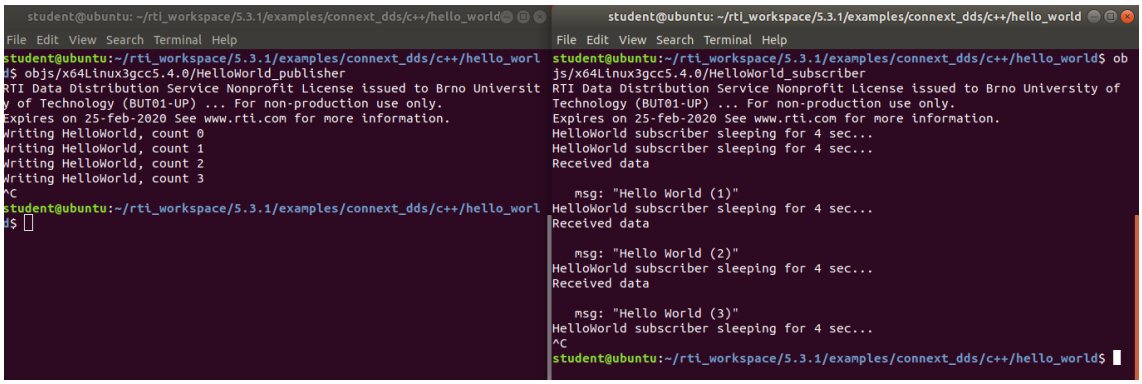


Obr. 2.2: Ukážka generovania kódov

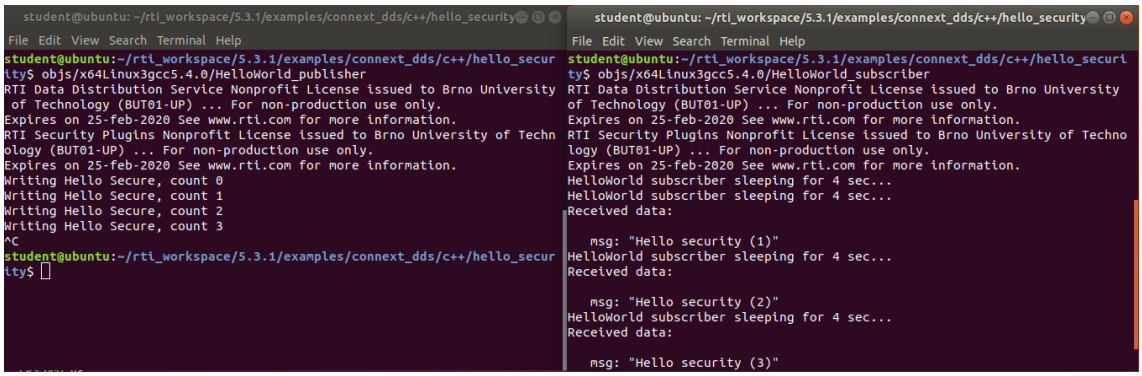
2.3 Prvotné otestovanie nešifrovanej a šifrovanej komunikácie

Po dôkladnom analyzovaní bol pre kompiláciu a spustenie týchto aplikácií využitý operačný systém Linux s platformou Ubuntu 18.04 64-bit vo virtuálnom prostredí VMware Workstation Pro. Cieľom bolo modifikovať a demonštrovať tieto aplikácie s následným odchytením komunikácie pomocou programu Wireshark. K úspešnej kompilácií bolo potrebné nainštalovať spomínanú *openssl* knižnicu priamo v operačnom systéme, doinštalovať zabezpečovací plugin *rti_security_plugin*, ktorý je súčasťou inštalačného balíka RTI, prepojiť prostredie RTI s doplnkovými balíkmi *openssl host*, *openssl target*, hlavičky a knižnice priamo z frameworku RTI pomocou systémových premenných.

V obr. 2.3 a 2.4 je na ľavej strane publisher a na pravej strane je subscriber už s výpisom zaslanej správy.



Obr. 2.3: Ukážka nezabezpečenej aplikácie



Obr. 2.4: Ukážka zabezpečenej aplikácie

2.3.1 Wireshark výsledky nezabezpečenej a zabezpečenej komunikácie

Na ukážkach môžeme vidieť rozdiely medzi týmito zachytenými prenosmi. V oboch ukážkach je spomínaný protokol na ktorom beží hlavná časť Publish – Subscribe komunikácie. Môžeme vidieť, že zabezpečená komunikácia obsahuje šifrované prvky ako *SEC_PREFIX*, *SEC_BODY* alebo *SEC_POSTFIX*.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
2	0.002574337	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
3	0.002663227	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
4	0.005496361	10.0.2.15	224.0.0.22	IGMPv3	54	Membership Report / Join group 239.255.0.1 for any sources
5	0.750299564	10.0.2.15	224.0.0.22	IGMPv3	54	Membership Report / Join group 239.255.0.1 for any sources
6	1.005555978	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
7	2.006384638	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
8	3.006987138	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
9	3.350692980	10.0.2.15	239.255.0.1	RTPS	862	INFO_TS, DATA(p)
10	3.353858239	10.0.2.15	239.255.0.1	RTPS	862	INFO_TS, DATA(p)
11	3.353968998	10.0.2.15	239.255.0.1	RTPS	862	INFO_TS, DATA(p)
12	4.007989787	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
13	4.361814183	10.0.2.15	239.255.0.1	RTPS	862	INFO_TS, DATA(p)
14	5.009451104	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
15	5.363472479	10.0.2.15	239.255.0.1	RTPS	862	INFO_TS, DATA(p)
16	6.011476521	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
17	6.366021938	10.0.2.15	239.255.0.1	RTPS	862	INFO_TS, DATA(p)
18	7.011631865	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
19	7.367285078	10.0.2.15	239.255.0.1	RTPS	862	INFO_TS, DATA(p)
20	8.012200387	10.0.2.15	239.255.0.1	RTPS	866	INFO_TS, DATA(p)
21	8.374867157	10.0.2.15	239.255.0.1	RTPS	862	INFO_TS, DATA(p)
22	22.674557767	10.0.2.15	224.0.0.22	IGMPv3	54	Membership Report / Leave group 239.255.0.1
23	23.533576719	10.0.2.15	224.0.0.22	IGMPv3	54	Membership Report / Leave group 239.255.0.1

Obr. 2.5: Ukážka nezabezpečenej komunikácie

No.	Time	Source	Destination	Protocol	Length	Info
15	3.581201398	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
16	4.005182228	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
17	4.581493206	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
18	5.005304857	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
19	5.012675535	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
20	5.013257257	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
21	5.014528986	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
22	5.015509805	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
23	5.015701822	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
24	5.016099637	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
25	5.016318016	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
26	5.016550942	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
27	5.016802291	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
28	5.017392200	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
29	5.017666501	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
30	5.017983408	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
31	5.018239039	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
32	5.018406695	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
33	5.018803001	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
34	5.019022582	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
35	5.019261149	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
36	5.019466631	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
37	5.117346701	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
38	5.120500154	10.0.2.15	239.255.0.1	RTPS	438	INFO_TS, SEC_PREFIX, SEC_BODY, SEC_POSTFIX
39	5.581740831	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
40	6.005732293	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
41	6.582144429	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
42	7.006032264	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
43	7.583134500	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
44	8.006671048	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
45	8.584125393	10.0.2.15	239.255.0.1	RTPS	1038	INFO_TS, DATA(p)
46	22.993429581	10.0.2.15	224.0.0.22	IGMPv3	54	Membership Report / Leave group 239.255.0.1
47	23.569457739	10.0.2.15	224.0.0.22	IGMPv3	54	Membership Report / Leave group 239.255.0.1

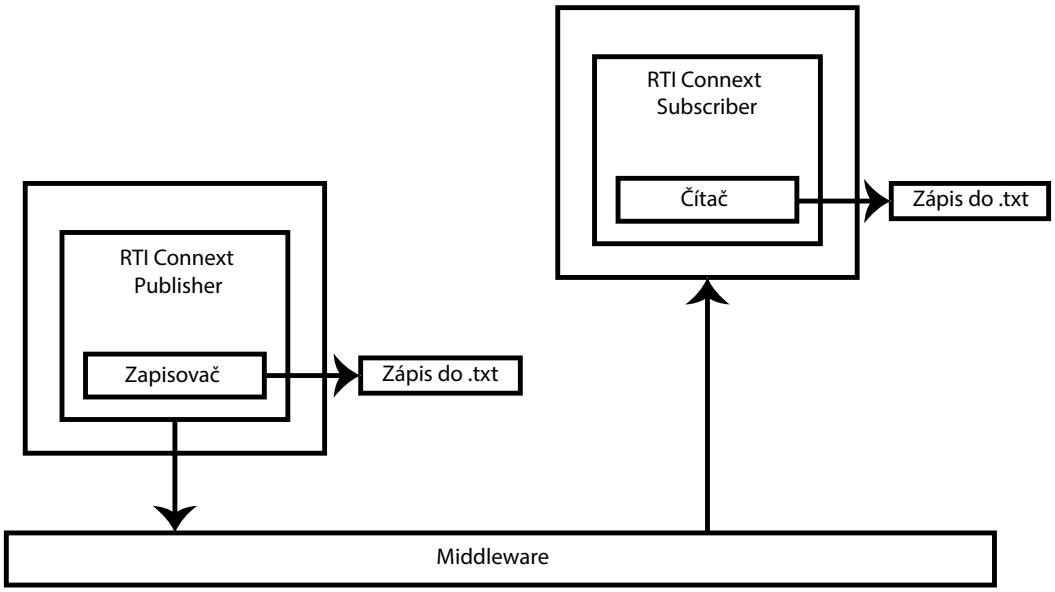
Obr. 2.6: Ukážka zabezpečenej komunikácie

2.4 Meranie odozvy

Hlavnou praktickou časťou bolo meranie odozvy medzi aplikáciou publishera a aplikáciou subscribera. Po dôkladnom analyzovaní bol zvolený pre meranie operačný systém Linux s platformou Ubuntu 18.04 64-bit, bežiaci vo virtuálnom nástroji VMware Workstation Pro. Pre beh operačného systému vo virtuálnom prostredí bolo nastavených 3 GB RAM a dve jadra procesora. Práca prebehla na notebooku Dell Inspiron 7000, ktorý má 8 GB RAM a štvorjadrový procesor Intel Core i7 s frekvenciou 2,6 GHz.

2.4.1 Scenár merania

Samotné meranie prebehlo pri meraní 100 správ a pri meraní 5000 správ. Odosielacia perióda bola nastavená na 10 ms a na 100 ms, to činí periódu 10 správ za sekundu a 100 správ za sekundu. Počet správ a perióda sa nastavovala priamo v kódoch publishera a subscribera. Najprv sa vždy spustil subscriber, perióda príjmania nebola dôležitá a nechala sa na nule, takže vždy keď prišla odozva od publishera tak po prijatí sa vytvorilo časové razítko, ktoré sa rovno uložilo do textového súbora. Potom sa spustil publisher kde sa nastavil počet správ priamo v konzole. Odosielacia perióda (send period) sa nastavovala priamo v kóde samotného publishera.



Obr. 2.7: Diagram merania

Čo sa týka úprav kódov merania, bolo potrebné modifikovať súbory publishera a subscribera. Bolo pridaných viac knižníc pre vytvorenie a ukladanie súborov, vytvorenie a uloženie časového razítka v nanosekundách. V samotných ukážkach sú zobrazené zmeny, v elektronickej prílohe už budú kompletné kódy publishera a subscribera. Tieto zmeny boli modifikované v nezabezpečenej a taktiež aj v zabezpečenej komunikácií. Vid. prílohu A.

2.4.2 Vytvorenie skriptu pre spracovanie nameraných údajov

Pre jednoduchšie spracovanie výsledkov bol vytvorený skript B, keďže prebehlo viacero meraní a je veľmi náročné spracovávať výsledky ručne, keďže veľkosť správ bol až do 1 MB a počet správ v meraniach bol podľa naplánovaného scenára veľmi vysoký. V súboroch už boli časové razítka v nanosekundách, ktoré boli získané z modifikovaných C++ súborov implementácie RTI spomínaných vyššie.

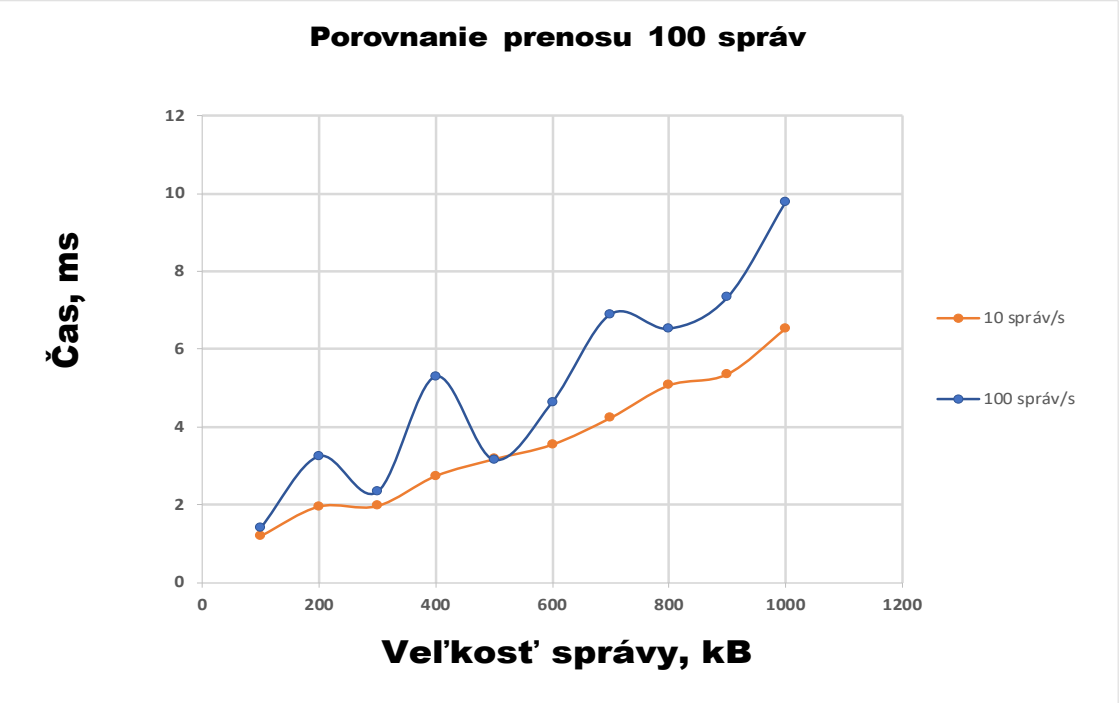
V skripte sa počíta rozdiel hodnôt subscribera a publishera. Nakoniec sú všetky hodnoty priemerované a v konečnom výpise je len jedna priemerná hodnota meškania správy. V ďalšej časti práce budú demonštrované výsledky rôznych scenárov merania, či už s rôznou veľkosťou správy, rôznou posielanou periódou a rôznym množstvom odoslaných správ. Je potrebné zabezpečiť rovnaký počet hodnôt v oboch súboroch, pretože pri veľmi malej perióde sa v meraní správy na začiatku komunikácie strácali. To zapríčiňuje nespoľahlivý protokol RTPS, ktorý je z rady protokolu UDP. V elektronickej prílohe budú kompletné tabuľky aj s počtom neprijatých správ oboch komunikácií.

2.4.3 Výsledky priemernej latencie správ nezabezpečenej komunikácie

Táto podkapitola je zameraná na jasné zobrazenie priemerného meškania správ scenárov nezabezpečenej komunikácie. Meraním bolo dokázané, že implementácia od RTI nedokáže väčšie správy spracovávať tak ako správy s menšou veľkosťou. Okrem veľkosti správ má veľký vplyv na prenos aj posielacia *(send)* perióda, ktorú vie užívateľ nastaviť v publisherovi, ale vždy je potrebná kompilácia, keďže sa jedná o zmeny kódoch a nie už v spustených aplikáciach.

Prenos 100 správ

Zatiaľ čo pri perióde 10 správ za sekundu sa hodnota úspešnosti prijatia a odoslania správ pohybovala na úrovni 90 %, pri perióde 100 správ za sekundu bola táto úspešnosť rapídne menšia, približne na úrovni 50 %. Oba krivky v obr. 2.8 vychádzajú z hodnôt pri odosielaní 100 správ.



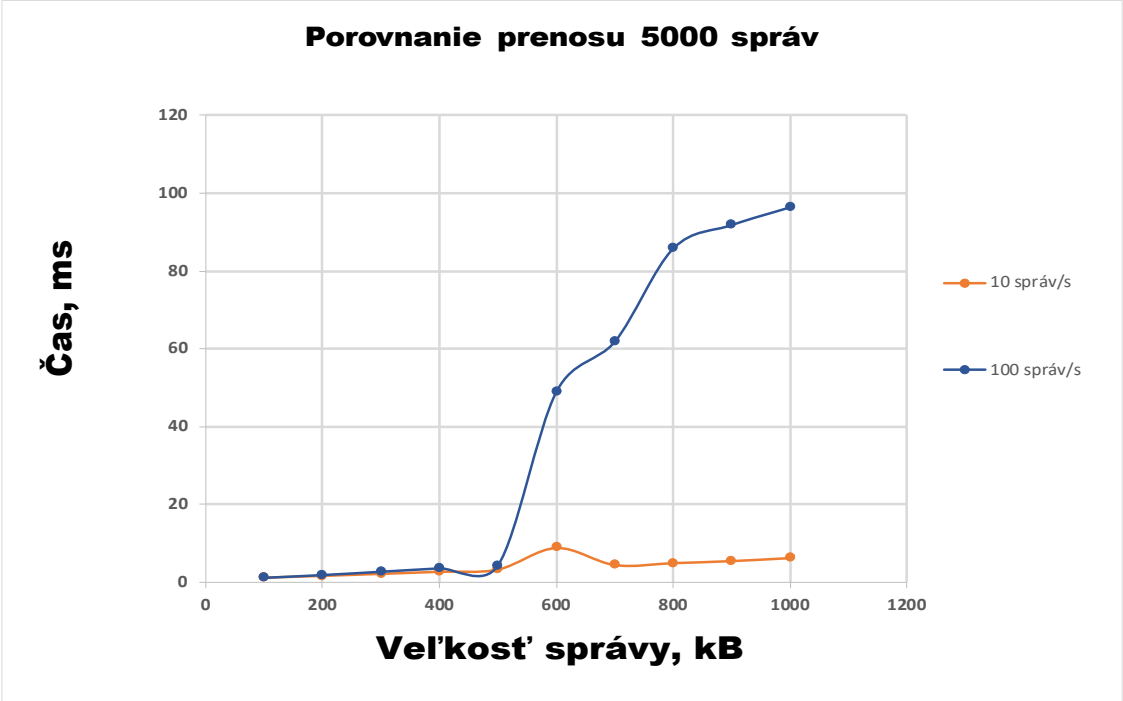
Obr. 2.8: Latencia 100 správ pri porovnaní nezabezpečenej komunikácie

Prenos 5000 správ

V grafe 2.9 nezabezpečenej komunikácie bol skúmaný vplyv počtu správ na priemernú latenciu, kde bolo spravené meranie s počtom správ 5000. Pri prenose s periódou 10 správ/s meškanie správ neprekročí 10 ms, čo je porovnateľné s výsledkom pri posielaní 100 správ, avšak pri zvýšení prenosovej periódy na 100 správ/s sa pri prenose správy nad 500 kB prudko zvýši latencia až do 100 ms. Dôvody tohto rapídneho zvýšenia môžu byť rôzne, jedna z príčin je v hardvéri PC, na ktorom bolo meranie realizované. Samozrejme ďalšie dôvody môžu byť v nastavenej operačnej pamäti RAM na 3 GB vo virtuálnom prostredí, ktorá nemusí stačiť.

2.4.4 Zabezpečenie RTI

Anonymná správa prechádzajúca počítačovými sieťami je nevyhnutná pre mnohé zabezpečenie komunikačných protokolov. Anonymné kanály boli predpokladom mnohých protokolov, ktoré zahŕňajú anonymné odovzdávanie správ medzi rovesníkmi. IP charakter dnešných sieti porušuje túto anonymitu pridaním adresy IP odosielateľa a príjemcu v každom pakete. S týmto vedomím sa každý môže pozrieť na pakety a mať vedomosti o tom, kto správu odosiela a pre koho sú správy určené. Okrem toho prijímač vždy vie, kto správu odoslal a môže odkaz odoslať späť odosielateľovi [16].



Obr. 2.9: Latencia 5000 správ pri porovnaní nezabezpečenej komunikácie

Generovanie autentifikačných kľúčov

Autentifikácia je proces uistenia sa, že účastník domény *(domain participant)* je tým, kým tvrdí, že je. Autentifikácia sa vykonáva prostredníctvom série správ o výzvach a odpovediach medzi účastníkmi. Tieto správy vykonávajú vzájomnú autentifikáciu, takže konečným výsledkom je, že tento účastník autentifikuje vzdialeného účastníka a naopak. Ak tento účastník zlyhá, vzdialený účastník ho ignoruje. Inak tento účastník iniciuje objavovanie koncového bodu so vzdialeným účastníkom a komunikácia sa obnoví ako normálne [17].

Povinná časť autentifikácie je reťazec, ktorý určuje úplnú cestu a názov súboru, ktorý obsahuje certifikáty *Identity Certificate Authority*. Súbor by mal byť vo formáte PEM. Táto certifikačná autorita identity sa používa na podpisovanie súborov certifikátov autentifikácie. OpenSSL by mal generovať tento súbor pomocou príkazov, ako sú nasledujúce:

RSA:

```
% openssl genrsa -out cakey.pem 2048
% openssl req -new -key cakey.pem -out ca.csr -config openssl.cnf
% openssl x509 -req -days 3650 -in ca.csr -signkey cakey.pem -out cacert.pem
% echo 01 > ca.srl
```

DSA:

% openssl dsaparam 2048 > dsaparam

% openssl gendsa -out cakeydsa.pem dsaparam

% openssl req -new -key cakeydsa.pem -out dsaca.csr -config opensslDSA.cnf

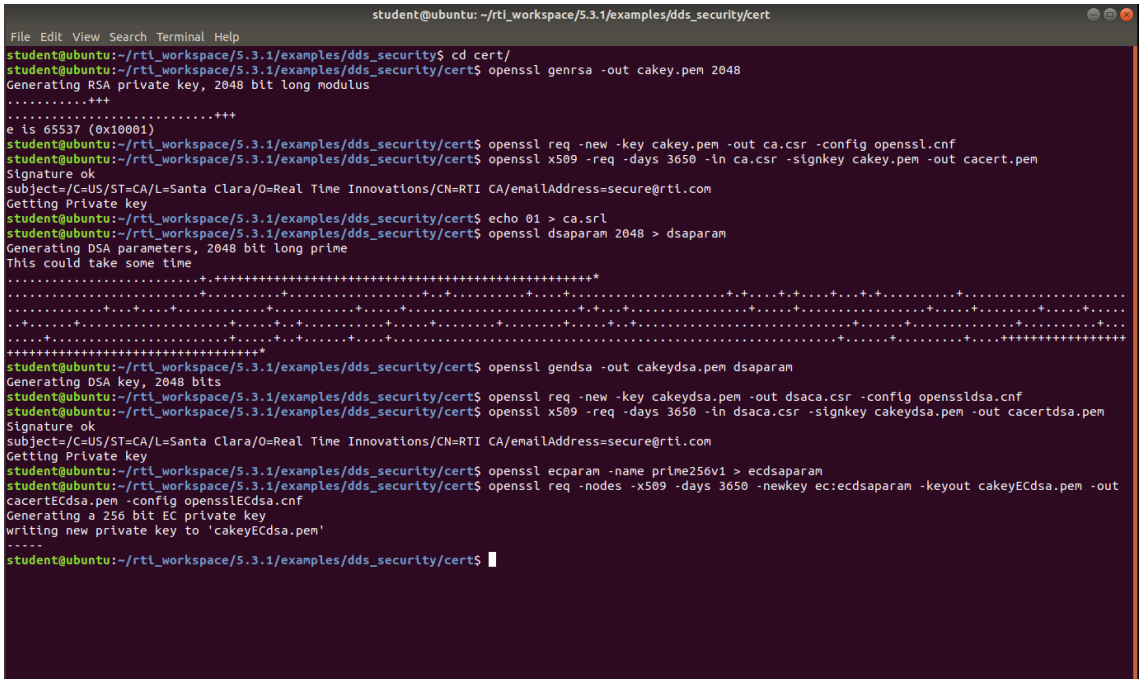
% openssl x509 -req -days 3650 -in dsaca.csr -signkey cakeydsa.pem -out cacertdsa.pem

ECDSA:

% openssl ecparam -name prime256v1 > ecdsaparam

% openssl req -nodes -x509 -days 3650 -newkey ec:ecdsaparam -keyout cakeyECdsa.pem -out cacertECdsa.pem -config opensslECDSA.cnf

Pri spúšťaní vyššie uvedených príkazov môže dojsť k varovaniu ohľadom cesty ku konfiguračnému súboru *openssl.cnf*. Je potrebné nastaviť systémovú premennú *OPENSSL_CONF* k tomuto súboru.



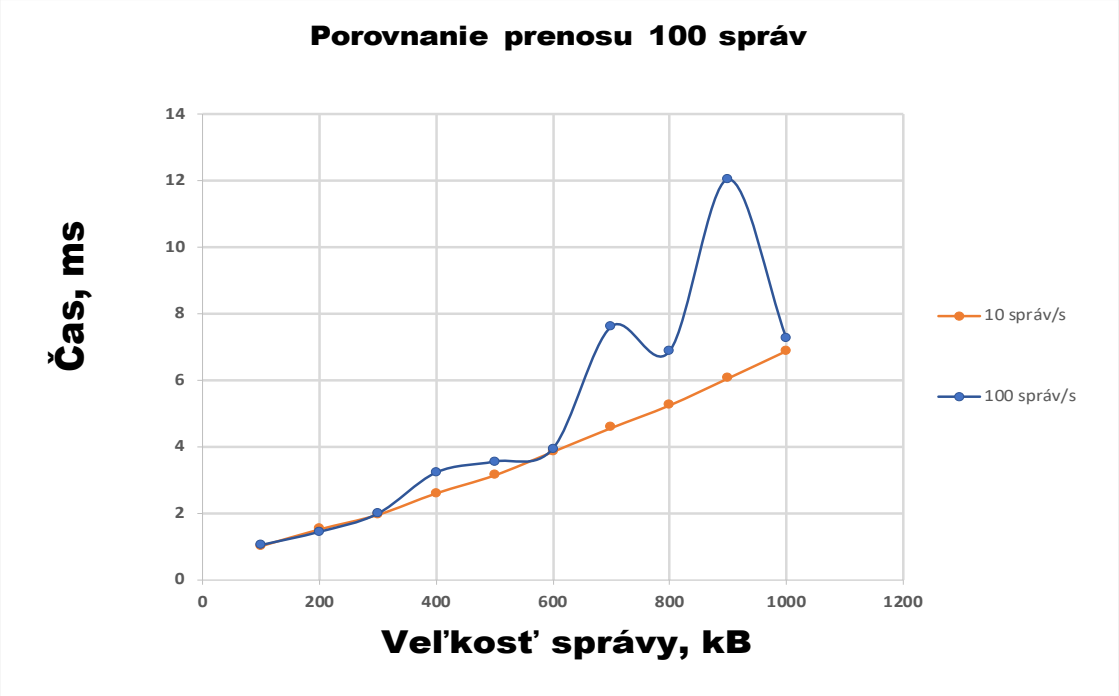
Obr. 2.10: Ukážka úspešného generovania autentifikačných kľúčov

2.4.5 Výsledky priemernej latencie správ zabezpečenej komunikácie

Aby bolo skompletizované meranie, bolo potrebné taktiež zopakovať ten istý proces pri zabezpečenej komunikácií, ktorá obsahuje zabezpečovacie prvky z upravenej openssl knižnice spoločnosti RTI. Meranie prebehlo za rovnakých podmienok ako meranie nezabezpečenej komunikácie, v porovnaní grafov sa latencia pohybuje na podobnej úrovni ako už v spomínanej nezabezpečenej komunikácií.

Prenos 100 správ

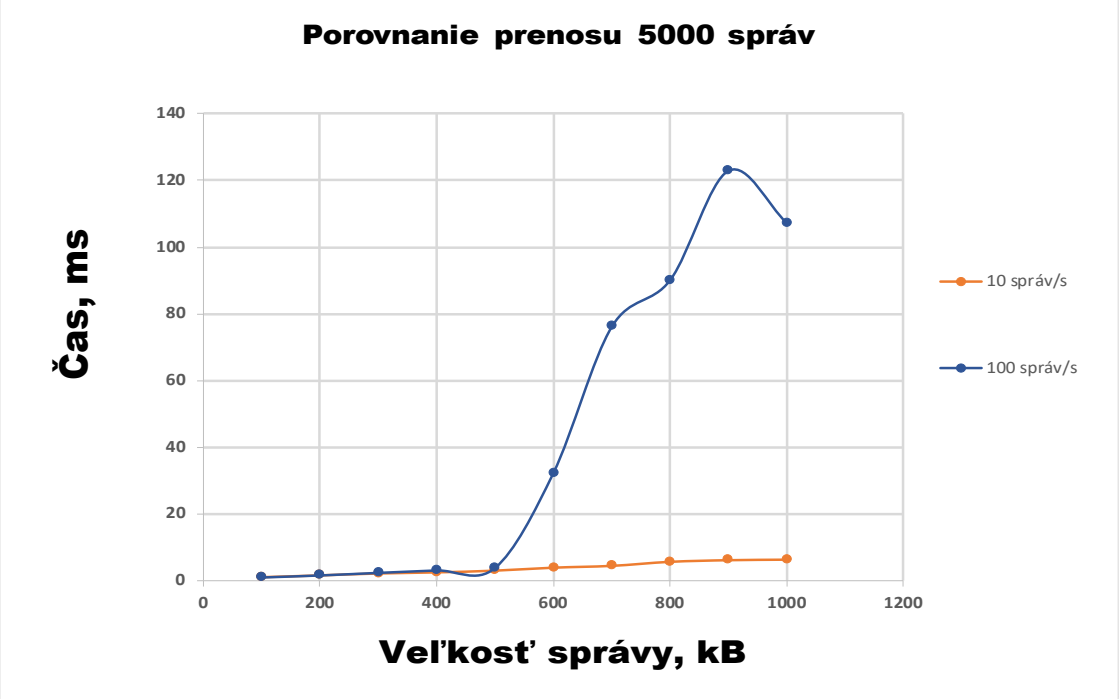
Pri perióde 10 správ za sekundu je pri sledovaní 100 správ krivka stabilná a rastie lineárne, avšak pri perióde 100 správ za sekundu to už stabilné nie je a to hlavne z dôvodu veľkého percenta neprijatých správ na úrovni 70 %. Dôvody väčšieho počtu neprijatých správ môžu byť rôzne, určite veľkú rolu hraje hardvér notebooku na ktorom meranie prebehlo, taktiež rôzne zabezpečovacie prvky z modifikovanej openssl knižnice, keďže zabezpečená komunikácia je predsa len o čosi náročnejšia ako nezabezpečená.



Obr. 2.11: Latencia 100 správ pri porovnaní zabezpečenej komunikácie

Prenos 5000 správ

V grafe s počtom správ 5000, sa opäť prejavila nestabilita pri meraní väčšej veľkosti správy pri perióde 100 správ za sekundu. Latencia vyskočila až na úroveň 122 ms. Pri posielaní 10 správ za sekundu sa krivka ešte viac zlinearizovala, keďže percento neprijatých správ sa ešte znížilo. Z tohto pohľadu je jasné, že používanie periódy 100 správ za sekundu nie je veľmi spoľahlivé riešenie a veľmi veľké percento správ nie je potvrdených subscriberom.



Obr. 2.12: Latencia 5000 správ pri porovnaní zabezpečenej komunikácie

3 Záver

V bakalárskej práci boli porovnané DDS implementácie, či už z hľadiska dátovo alebo objektovo zameraných modelov, taktiež sa skúmalo či je lepšie použiť opensource alebo už pokročilejšiu implementáciu, ktorá nie je voľne dostupná. Boli preskúmané prostredia Enterprise Architect, OpenDDS, Opensplice a RTI Connext.

V práci sa pracovalo v prostredí RTI Connext, kde bola poskytnutá dočasná akademická licencia firmou RTI. Následne bola demonštrovaná zabezpečená a nezabezpečená aplikácia vo virtuálnom prostredí VMware Workstation Pro prostredníctvom operačného systému Linux Ubuntu 18.04. Z priebehu aplikácií boli odchytené výsledky Publishera a Subscribera v programe Wireshark, prebehlo porovnanie a dokázanie, že zabezpečená komunikácia je bezpečnejšia no vyžaduje použitie špeciálnych kryptografických knižníc, v našom prípade išlo o knižnicu openssl modifikovanú firmou RTI.

Celá komunikácia prebieha pomocou protokolu RTPS, ktorý definuje formát správ a interpretuje používaný scenár pri komunikácií medzi aplikáciami, čo je vlastne pointa tohto protokolu. RTPS využíva možnosti multicastu prepravného mechanizmu, kde jedna správa od odosielateľa môže dosiahnuť viac prijímačov.

V ďalšom priebehu práce boli modifikované kódy publisher a subscriber v nezabezpečenej a taktiež aj v zabezpečenej komunikácií podľa potreby merania, hlavne čo sa týka časového razítka prevedeného do nanosekúnd (v grafoch prevedené do milisekúnd) a ukladanie samotných časov do súboru. Pre realizáciu posielania väčších správ (až do 1MB) bolo nutné generovať nové súbory s upravenou premennou v základnom .idl súbore implementácie RTI. V meraní bola skúmaná latencia pri rôznych typoch merania s rôznymi veľkosťami správ. Pre spracovanie výsledkov bol vytvorený skript v jazyku python. Pri zabezpečenej komunikácií bolo potrebné generovať autentifikačných kľúčov čo sa taktiež podarilo zrealizovať. Posielanie správ s periódou 10 správ za sekundu sa ukázalo ako oveľa presnejšie a stabilnejšie ako posielanie správ pri vyššej perióde, konkrétne 100 správ za sekundu. Pri posielaní správ s väčšou veľkosťou pri perióde 100 správ za sekundu bola latencia vyššia pravdepodobne ovplyvňovaná hardvérom počítača a nastavením virtuálneho prostredia.

Literatúra

[1] K. Krinkin, A. Filatov, A. Filatov, O. Kurishev, and A. Lyanguzov, "Data Distribution Services Performance Evaluation Framework," in *2018 22nd Conference of Open Innovations Association (FRUCT)*, May 2018, pp. 94–100.

[2] J. M. Schlesselman, G. Pardo-Castellote, and B. Farabaugh, "OMG data-distribution service (DDS): architectural update," in *IEEE MILCOM 2004. Military Communications Conference, 2004.*, Oct 2004, pp. 961–967.

[3] G. Pardo-Castellote, "COTS Journal," in *DDS Spec Outfits PublishSubscribe Technology for the GIG*, 2005.

[4] L. Pokorný, "Podpůrné nástroje pro RTPS komunikaci," Diplomová práca, České vysoké učení technické v Praze, Fakulta elektrotechnická., Praha, 2005.

[5] M. Xiong, J. Parsons, J. Edmondson, H. Nguyen, and D. C. Schmidt, "Evaluating the performance of publish/subscribe platforms for information management in distributed real-time and embedded systems ," 2010.

[6] "RTI Connext Core Libraries and Utilities User's Manual, Version 5.0," 2012. [Online]. Dostupné z: https://community.rti.com/static/documentation/connext-dds/5.2.3/doc/manuals/connext_dds/RTI_ConnextDDS_CoreLibraries_UsersManual.pdf

[7] "PrismTech OpenSplice RTNetworking," 2013. [Online]. Dostupné z: http://www.prismtech.com/opensplice/products/communication

[8] "Enterprise Architect," 2000-2019. [Online]. Dostupné z: https://sparxsystems.com/

[9] E. de Jong, "Connext DDS documentation," 2013. [Online]. Dostupné z: http://www.controlshell.com/products/dds/index.html

[10] "Real-Time Innovations, Inc. RTPS Wire Protocol, Version 1.0." 2002. [Online]. Dostupné z: http://orte.sourceforge.net/rtps1.2.pdf

[11] "Introduction to OpenDDS," 1993-2016. [Online]. Dostupné z: https://objectcomputing.com/resources/publications/mnb/introduction-to-opendds

[12] "DDS Overview," 2019. [Online]. Dostupné z: http://opendds.org/about/dds_overview

[13] D. C. Schmidt and H. van't Hag, "Addressing the challenges of mission-critical information management in next-generation net-centric pub/sub systems with OpenSplice DDS," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, April 2008, pp. 1–8.

[14] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Annual International Cryptology Conference*. Springer, 1999.

[15] "OpenSSL Software Foundation," 1999-2018. [Online]. Dostupné z: https://www.openssl.org/

[16] P. Ribarski and L. Antovski, "Mixnets: Implementation and performance evaluation of decryption and re-encryption types," in *Proceedings of the ITI 2012 34th International Conference on Information Technology Interfaces*, 2012.

[17] "RTI Security Plugins Getting Started Guide Version 5.3.0," 2017. [Online]. Dostupné z: https://community.rti.com/static/documentation/connext-dds/5.3.0/doc/manuals/connext_dds/dds_security/RTI_SecurityPlugins_GettingStarted.pdf

Zoznam symbolov, veličín a skratiek

API	Application Programming Interface
ATC	Air Traffic Control
ATM	Air Traffic Management
BSD	Berkeley Software Distribution
CORBA	Common Object Request Broker Architecture
DCPS	Data-centric publish-subscribe
DDS	Data Distribution Service
DES	Data Encryption Standard
EA	Enterprise Architect
HLA	High Level Architecture
IIoT	Industrial Internet of Things
IDL	Interface Description Language
IEEE	Institute of Electrical and Electronics Engineers
OMG	Object Management Group
OS	Operating System
PC	Personal Computer
QoS	Quality of Service
RTF	Rich Text Format
RTI	Run-time Infrastructure
RTPS	Real-Time Publish-Subscribe
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TSL	Transport Layer Security
UAV	Unmanned aerial vehicle
UDP	User Datagram Protocol
UML	Unified Modeling Language
VM	Virtual Machine
WAN	Wide Area Network
XMI	XML Metadata Interchange

Zoznam príloh

A Zmeny kódu publishera a subscribera

```
// Zmena posielacej periódy programu, prvé číslo je v sekundách,
// druhé číslo je v nanosekundách
struct DDS_Duration_t send_period = {0,10000000};

// Ukladanie súboru pomocou knižnice #include<fstream>
    using namespace std;
    ofstream vystup;
    vystup.open("textPublisher.txt");

// Zápis premennej timestamp do súboru a výpis do konzoly
if (vystup.is_open()) {
    vystup << timestamp;
    vystup.close();
}

// Prevod času na nanosekundy a zápis do premennej
// pomocou knižnice
#include<chrono>
    using namespace std::chrono;
    timestamp = std::chrono::duration_cast
    <std::chrono::nanoseconds>
    (std::chrono::system_clock::now()
    .time_since_epoch()).count();
```

B Skript pre spracovanie nameraných hodnôt

```
1 #!/usr/bin/python
2
3 # Otvorenie ssborov a pretypovanie na float
4 pub = [float(riadok.strip()) for riadok in open("
                                    PublisherSubscriber/textPublisher
                                    .txt")]
5 sub = [float(riadok.strip()) for riadok in open("
                                    PublisherSubscriber/
                                    textSubscriber.txt")]
6
7 # Vypocet rozdielu a pretypovanie na string pre pocitanie
                                    priemeru
8 rozdiely = [sub - pub for sub, pub in zip(sub, pub)]
9 print rozdiely
10
11 # Ziskanie priemeru zo zoznamu hodnot
12 def Average(rozdiely):
13     return sum(rozdiely) / len(rozdiely)
14
15 average = Average(rozdiely)
16
17 # Vypis priemernej hodnoty na 3 desatinne miesta
18 print("Priemerna hodnota = ", round(average, 3))
```