УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Михаило Бабић

Интеграција TR-369 клијента на Андроид СТВ уређају

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2024

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:			
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Завршни (Bachelor) рад	
Аутор, **АУ**:		Михаило Бабић	
Ментор, **МН**:		Проф. др Илија Башичевић	
Наслов рада, **НР**:		Интеграција TR-369 клијента на Андроид СТБ уређају	
Језик публикације, **ЈП**:		Српски / ћирилица	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2024	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		7/36/6/1/18/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кqучне речи, **ПО**:		TR-369, MQTT, СТБ, Андроид	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		Потребно је прекомпајлирати и интегрисати решење отвореног кода Broadband Foruma OB USP Agent-a на Android TV оперативном систему, као и имплементирати Јава сервис који то решење покреће. Применити минимални модел података потребан за идентификацију уређаја и валидирати успешну комуникацију користећи obuspa-test-controller софтвер.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	Проф. др Мирослав Поповић	
	Члан:	Проф. др Небојша Пјевалица	Потпис ментора
	Члан, ментор:	Проф. др Илија Башичевић	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		Mihailo Babić	
Mentor, **MN**:		Ilija Bašičević, PhD.	
Title, **TI**:		TR-369 Client bring up on Android STB	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2024	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/36/6/0/18/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		TR-369, MQTT, STB, Android	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		The goal of this work is to cross-compile and integrate the open-source Broadband Forum OB USP Agent solution on the Android TV operating system, as well as to implement a Java service that runs this solution. Apply the minimum data model required for device identification and validate successful communication using the obuspa-test-controller software.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Miroslav Popović, PhD.	
	Member:	Nebojša Pjevalica, PhD.	Menthor's sign
	Member, Mentor:	Ilija Bašičević, PhD.	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

ЗАДАТАК ЗА ЗАВРШНИ РАД

(Податке уноси предметни наставник - ментор)

Студијски програм:	Рачунарство и аутоматика		
Студент:	Михаило Бабић	Број индекса:	RA 52/2020
Степен и врста студија:	ОАС		
Област:	Електротехника и рачунарство		
Ментор:	Проф. др Илија Башичевић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;

НАСЛОВ ЗАВРШНОГ РАДА:

Интеграција TR-369 клијента на Андроид СТБ уређају

ТЕКСТ ЗАДАТКА:

У оквиру овог дипломског рада потребно је урадити следеће:

1. Прекомпајлирати решење отвореног кода Broadband Foruma OB-USP-Agent за Android TV оперативни систем.
2. Интегрисати OB-USP-Agent у Android TV систем.
3. Применити минимални модел података потребан за идентификацију уређаја.
4. Валидирати успешну комуникацију коришћењем obuspa-test-controller решења отвореног кода.
5. Написати текст дипломског рада

Руководилац студијског програма:	Ментор рада:
Проф. др Милан Рапаић	Проф. др Илија Башичевић

Примерак за: o - Студента; o - Ментора

Садржај

Списак слика

СПИСАК ТАВЕЛА

СКРАЋЕНИЦЕ

TR-369 – **T**echnical **R**eport **369**, протокол за надзор и управљање

USP – **U**ser **S**ervice **P**latform,

MTP – **M**essage **T**ransfer **P**rotocol, протокол за слање порука

MQTT – **M**essage **Q**ueuing **T**elemetry **T**ransport, протокол за комуникацију

TCP – **T**ransmission **C**ontrol **P**rotocol

TLS – **T**ransport **L**ayer **S**ecurity

URL – **U**niform **R**esource **L**ocator

CRUD – **C**reate, **R**ead, **U**pdate, **D**elete, основне операције над подацима

IP – **I**nternet **P**rotocol, интернет протокол

E2EE – **E**nd-to-**E**nd Encryption, енкрипција са краја на крај

JNI – **J**ava **N**ative **I**nterface, спрега између Јава програмског језика и C кода

JSON – **J**ava**S**cript **O**bject **N**otation, формат за складиштење и слање података

STB – **S**et-**T**op **B**ox, Дигитални ТВ пријемник

1. Увод

Са брзим развојем интернет технологија, расте потреба за праћењем перформанси крајњих уређаја у мрежама, као и за даљинском подршком корисницима и прикупљањем дијагностичких података са уређаја. Овај раст потребе произилази из све већег броја уређаја који су повезани на мреже, укључујући и СТБ уређаје, који су кључни у пружању мултимедијалних услуга корисницима. Уз то, са порастом сложености мрежних система, постаје неопходно пратити перформансе и функционалност ових уређаја како би се осигурала стабилна и ефикасна мрежна инфраструктура. У том контексту, имплементација TR-369 протокола на Андроид СТБ уређају постаје значајна, омогућавајући напредно управљање и праћење уређаја путем мреже, као и ефикасно прикупљање дијагностичких података ради побољшања перформанси и корисничког искуства.

У овом раду реализовани су кориснички агент и контролер засновани на TR-369 комуникационом протоколу, као и скуп проширења који прилагођавају рад корисничког агента на циљаној СТБ платформи.

2. Теоријске основе

2.1 TR-369 комуникациони протокол

TR-369, назван USP (eng. User Services Platform), је стандардизован протокол за управљање удаљеним уређајима који нам нуди функције надгледања и управљања уређаја повезаних преко интернета. Спецификација циља на велики број уређаја укључујући приступне пролазе (eng. Gateway), телефоне, паметне уређаје. Сврха протокола је да понуди доследан и скалабилан начин за операторе, произвођаче и кориснике да ефикасно управљају и прате све већи број уређаја у својим мрежама. TR-369 је креиран као замена за постојећи TR-069 протокол, додајући подршку за нове врсте апликација уз одржавање компатибилности и адресирајући недостатке TR-069 протокола. Као резултат тога TR-369 пружа флексибилан и проширив оквир који се може ефективно користити за управљање све већем броју уређаја садашњице. Користи се за управљање животног циклуса повезаних уређаја, пружањем услуга, аутентификацију уређаја, конфигурацију уређаја, удаљену подршку корисницима, ефикасно прикупљање података и њиховој обради и анализи, одржавање мреже, осигуравање квалитета искуства и квалитета услуге, управљање паметним кућама и уређајима.

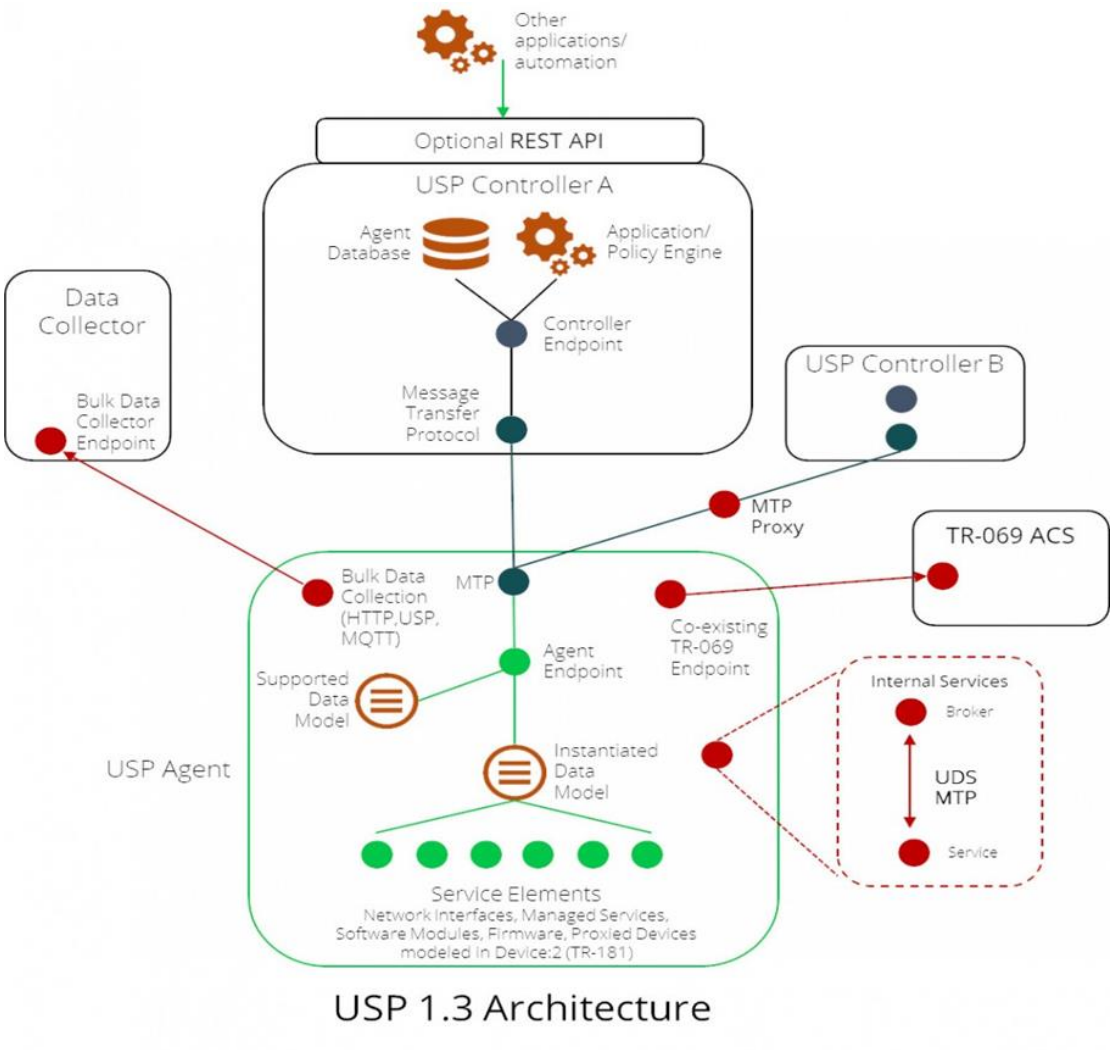
2.1.1 Архитектура

Платформа корисничких сервиса (USP) састоји се од колекције крајњих тачака (агената и контролера) које омогућавају апликацијама управљање елементима сервиса. Ови елементи сервиса се састоје од скупа објеката и параметара који моделирају одређени сервис, као што су мрежни посредници, модули програмске подршке,

уграђене програмске подршке уређаја, удаљени елементи који се посредују кроз друге посреднике, виртуелни елементи или други управљани сервиси.

USP је састављен од неколико компоненти:

- Механизам за откривање тачака и успостављање поверења између истих
- Метода за шифровање порука за транспорт
- Систем за поверљивост са краја на крај, интегритет и проверу идентитета
- Транспорт порука преко једног или више MTP са одговарајућом MTP сигурношћу
- Скуп стандардизованих порука заснованих на CRUD моделу, заједно са механизмом операцијама дефинисаним објектима, и механизмом нотификација CRUD-ON
- Ауторизација и контрола приступа
- Метода за моделирање услуга коришћењем скупа објеката, параметара, операција и догађаја подржаних и инстанцираних од стране модела података



Слика 2.1 Архитектура TR-369 [1]

USP крајња тачка може да се понаша као Агент или Контролер. Контролери само шаљу поруке Агентима, а Агенти шаљу поруке Контролерима. USP крајња тачка комуницира са осталим крајним тачкама преко једног или више протокола за трансфер порука (MTP). Ова комуникација је осигурана од стране MTP или уграђених USP механизама или оба истовремено.

USP Агент је део програмске подршке који пружа имплементацију USP протокола за везу са USP контролером са којим размењује податке. Садржи модел података заснован на TR-181и2 спецификацији, подржан модел података, и повезује га са конфигурацијама уређаја, услугама, и информацијама о статусу уређаја коришћењем интерних протокола, дозвољавајући USP контролеру да интерагује са системом. USP Агент има три мода рада: брокер, сервис и контролер.

USP Сервис је одговоран за имплементацију дела укупне функционалности уређаја. Сервис излаже скуп сервисних елемената који се односе на функционалност за чију имплементацију је одговоран. Сервис може имати потребу за интеракцијом са елементима сервиса који су изван њеног функционалног домена, било да су то елементи сервиса које је изложио брокер или неки други сервис. Сервис садржи локалног USP агента и може имати USP контролер уграђен у њега. USP агент служи да изложи део модела података који контролише сервис USP брокеру. (послужилац модела података). USP контролер служи за преузимање/конфигурисање делова модела података који нису директно изложени USP сервису (потрошач модела података).

USP Брокер је одговоран за излагање скупа сервисних елемената уређаја спољним USP контролерима. Ово укључује све елементе модела података које открива USP Агент који се налазе у брокеру, као и све елементе модела података које су открили USP сервиси који су се повезали са брокером. Брокер служи као посредник за сервисе које желе да ступе у интеракцију са елементима модела података које одржавају други делови уређаја. Брокер има уграђен и USP агент и контролер. USP Агент служи и као агент који излаже окружење управљања уређајем спољном свету и агент било ком контролеру који се налазе унутар уређаја.

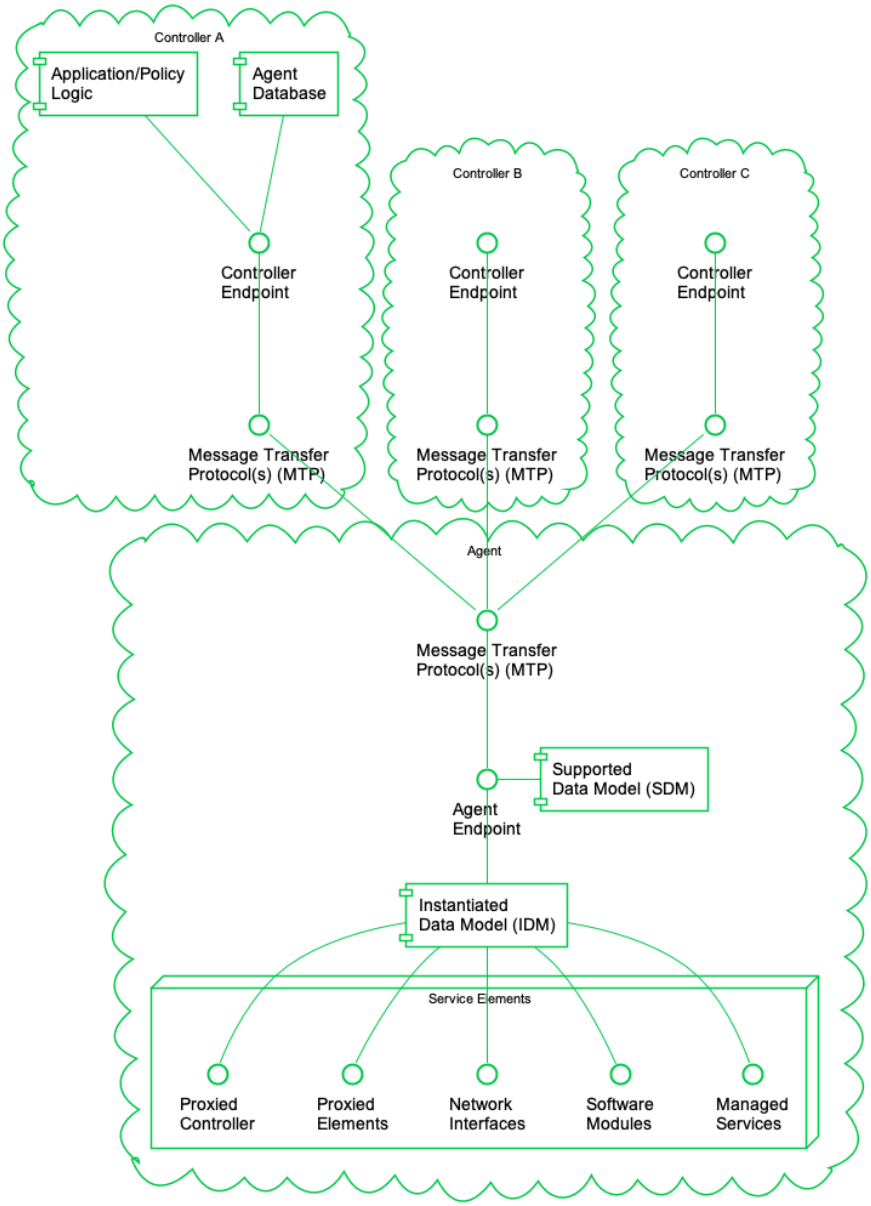
USP брокер има 3 главне одговорности:

- Прати сервисне елементе (делове модела података) које USP сервиси желе да изложе другим ентитетима.
- Преусмерава USP комуникацију унутар уређаја на основу сервисних елемената које су изложили USP сервиси.

- Обезбеди уједињени приказ сервисних елемената уређаја USP контролорима који су ван уређаја.

USP Контролер (**интерни**) служи као контролор за сву комуникацију са USP сервисима, као и преузимање/конфигурисање делова модела података који нису директно изложени USP сервису. (потрошач модела података).

USP Контролер манипулише, кроз USP Агенте, скуп сервисних елемената који су представљени у моделу података Агента. Може да одржава базу података USP Агената, њихових могућности и њихових стања. USP контролер обично служи као посредник до корисничке апликације.



Слика 2.2 Контролер-Агент шема [2]

USP је дизајниран тако да омогући контролеру да управља сервисним елементима агента, користећи стандардизоване описе сервисних елемената. Сервисни елемент је генерални израз за објекате, команде, догађаје и параметре који заједно чине скуп функционалности који су под контролом агента или контролера. Агентски сервисни елементи су представљени у моделу података, где инстанцирани модел података представља тренутно стање уређаја, а подржани модел представља све сервисне елементе које агент подржава.

USP модели података су подељени у два типа: Root и Service. Основни модел података, Device, користи се за описивање главних функција уређаја који су свесни мреже, укључујући посреднике, програмску подршку, дијагностику, компоненте заједничке за USP и друге сервисе, и основне информације агента неопходне за рад USP-а. Сервисни модели података описују модуларну функционалност која омогућава проширење основног модела података на уређају (у оквиру Device.Services.) за пружање одређених услуга, као што су гласовна услуга, сет-топ-бокс услуга, мрежно складиште, паметни кућни уређаји и слично. Објекти, параметри, команде и догађаји које одређени агент подржава из своје имплементације модела података дефинишу шта се извештава контролерима преко GetSupportedDM поруке.

2.1.2 Откривање и оглашавање

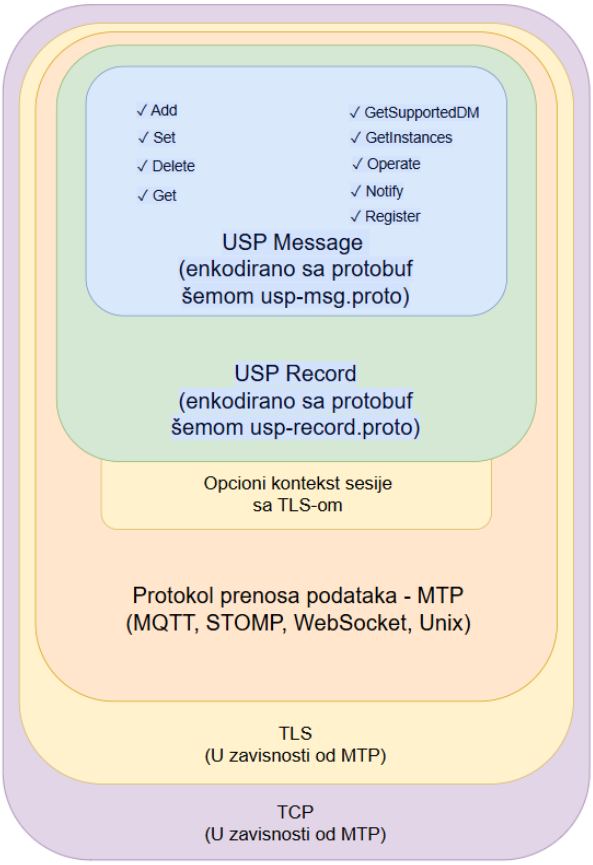
Откривање је процес којим USP крајње тачке сазнају детаље MTP везе друге крајње тачке, било за слање USP порука у контексту постојеће везе (где је USP идентификатор крајње тачке контролора) или за успостављање новог односа. Оглашавање је процес којим USP крајње тачке објављују своје присуство (или се објављује присуство USP крајње тачке) другим USP крајњим тачкама.

Агент који има USP везу са контролером мора да зна идентификатор крајње тачке контролора. Агент који остварује USP везу са контролером треба да добије информације које му омогућавају да одреди најмање један MTP, IP адресу, пролаз и путању ресурса (ако то захтева MTP) контролора. Ово може бити URL са свим овим компонентама.

2.1.3 Протоколи преноса порука (MTP)

USP поруке се шаљу између крајњих тачака преко једног или више протокола за пренос порука, где се протокол за пренос порука (MTP) односи на протокол апликативног слоја.

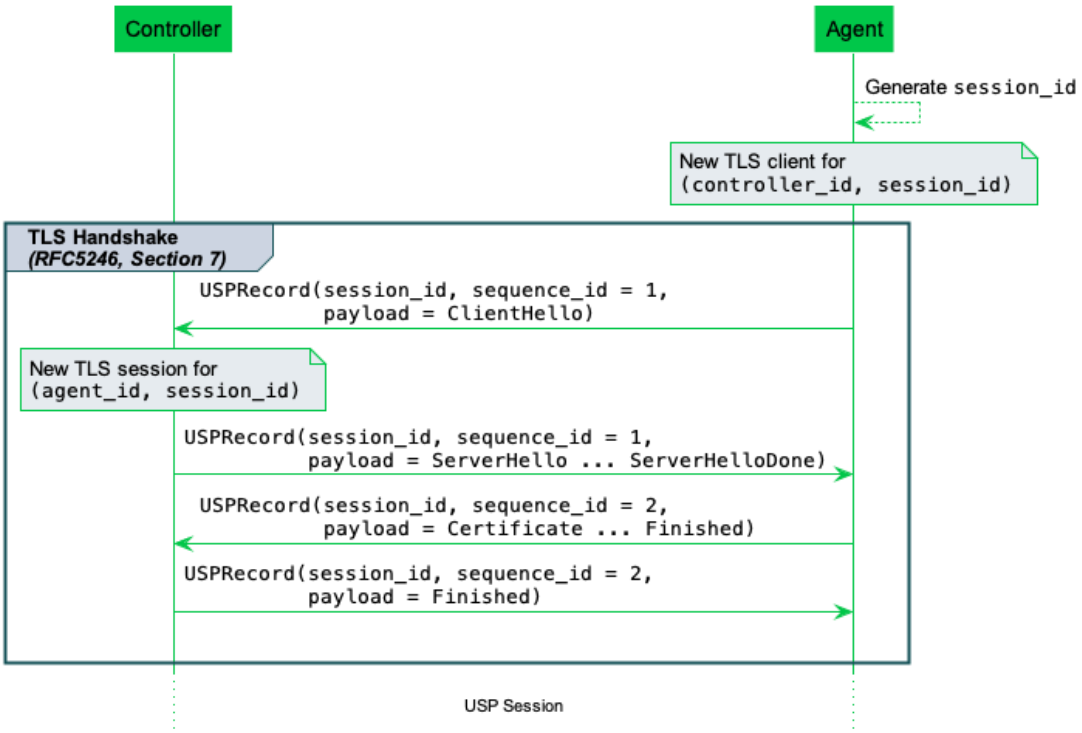
Агенти и контролори могу подржавати више од једног MTP-а. Када агент подржава више MTP-ова, агент може бити конфигурисан са параметрима за достизање одређеног контролера преко више од једног MTP-а. Када агент треба да пошаље обавештење таквом контролору, агент може бити дизајниран (или конфигурисан) да изабере одређени MTP, да покуша да пошаље обавештење контролору на свим MTP-овима истовремено или да покуша MTP-ове узастопно. USP је дизајниран да омогући крајњим тачкама да препознају када добију дупликат поруке и да одбаце све дупликате. Крајње тачке ће увек слати одговоре на исти MTP где је порука примљена. Протокол за пренос порука мора да користи сигуран транспорт када USP поруке прелазе границе локалне мреже. На пример, можда неће бити потребно користити MTP ниво заштите када се налазите у локалној мрежи (LAN) крајњег корисника. Међутим, неопходно је обезбедити транспорт до и са интернета, и очекује се да MTP-ови који раде преко TCP-а имплементирају барем TLS 1.2. Специфични захтеви за имплементацију су дефинисани за сваки MTP који USP подржава.



Слика 2.3 USP проткол стек

2.1.3.1 TLS

Када се TLS користи као механизам заштите корисног дела поруке за USP поруку, TLS захтева употребу контекста сесије за преговарање о својој TLS сесији. USP крајња тачка која је покренула контекст сесије ће деловати у улози TLS клијента приликом успостављања безбедносног слоја. Безбедносни слој је конструисан коришћењем стандардног TLS руковања, инкапсулираног унутар једног или више од датаграма корисног дела поруке USP записа. Према TLS протоколу, успостављање нове TLS сесије захтева два повратна путовања.



Слика 2.4 Остваривање TLS везе [3]

2.1.3.2 USP Поруке

USP садржи поруке за креирање, читање, ажурирање и брисање објеката, извођење операција дефинисаних објектима и омогућавање агентима да обавештавају контролоре о догађајима. Ово се често назива CRUD са додатком O (ради) и N (обавести) или CRUD-ON.

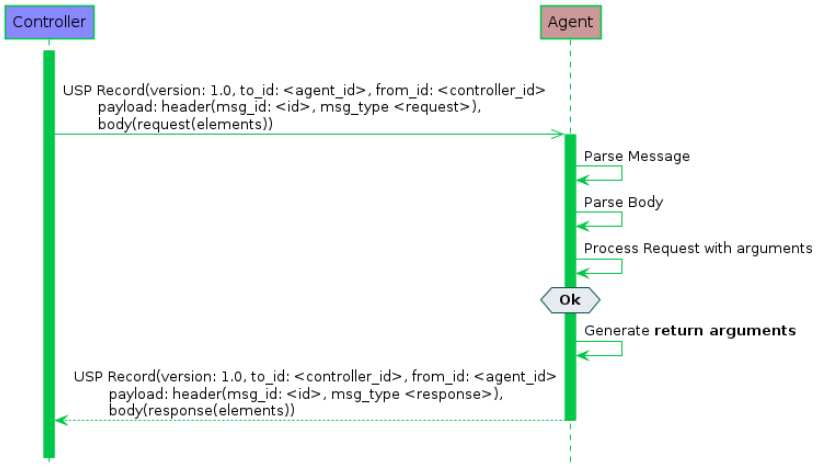
USP порука се односи на комуникациону јединицу која се размењује између USP контролера и USP агента у оквиру USP оквира. Свака USP порука садржи заглавље и тело. Заглавље садржи основне информације о одредишту и координацији, и одвојено је да би омогућило рад сигурносних механизама и механизама за откривање.

Тело садржи саму поруку и њене аргументе. Постоје три типа USP порука: захтев, одговор и грешка.

Захтев је порука послата са изворне USP крајње тачке на циљну USP крајњу тачку која укључује поља која треба обрадити и враћа одговор или грешку. Осим ако није другачије назначено, сви захтеви имају придружени одговор. Иако се већина захтева шаље од контролора агенту, поруке обавештења и регистрације прате исти формат као и захтев, али се шаљу од агента контролору.

2.1.3.3 USP запис

Све USP поруке су инкапсулиране USP записом. USP запис је дефинисан као носилац података протокола за пренос порука, који обухвата низ датаграма који чине USP поруку, пружа додатне метаподатке потребне за заштиту интегритета, заштиту носиоца података и испоруку фрагментираних USP порука. Додатна поља метаподатака се користе за идентификацију контекста E2E сесије, одређивање стања функције сегментације и поновног састављања, потврде примљених датаграма, захтевање поновног преноса и одређивање типа кодирања и безбедносног механизма који се користи за кодирање USP поруке. USP запис се дефинише у PROTOBUF формату.



Слика 2.5 Успешна захтев/одговор секвенца [4]

2.1.4 UNIX утичница

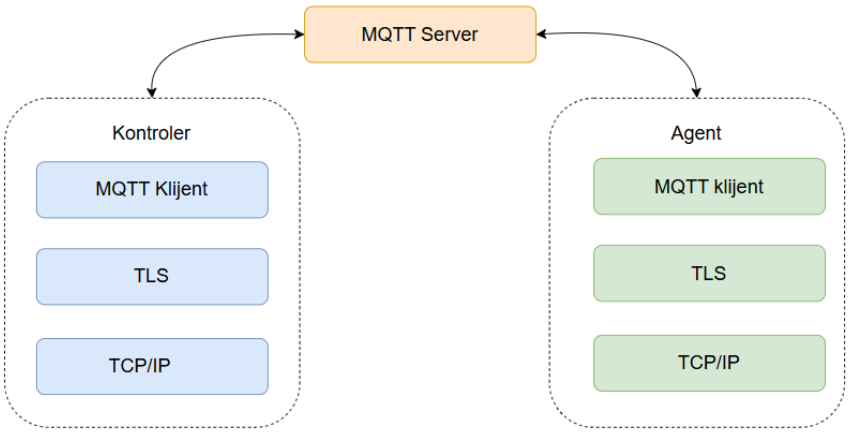
Био је потребан начин да се контролне команде шаљу покренутом USP Агенту за шта користимо UNIX утичницу. Ово је интерни протокол преноса порука за комуникацију на истом уређају. У овом случају се користи за слање команди између USP агента покренутог као сервис и USP агента који се покреће када постоји потреба да се ажурирају подаци.

2.2 MQTT

MQTT је протокол за комуникацију који се користи у апликацијама за ефикасну размену порука између уређаја и сервера. Протокол је дизајниран да буде једноставан за имплементацију, и користи механизам претплате и објаве. Као би се поруке усмериле намењеним корисницима дефинишу се теме. Уређај који жели да пошаље поруку објављује поруку на одређену тему, а уређај који жели да прима поруке претплаћује се на тему. Посредник (брокер) распоређује поруке између уређаја, у складу са темама на које су повезани уређаји претплаћени и на које објављују. PINGREQ и PINGRESP пакети се могу користити за одржавање везе ако се тајмер ближи истеку и нема потребе за другом врстом поруке. PINGREQ такође може да користи клијент у било ком тренутку да провери статус везе.

2.2.1 MQTT као MTP

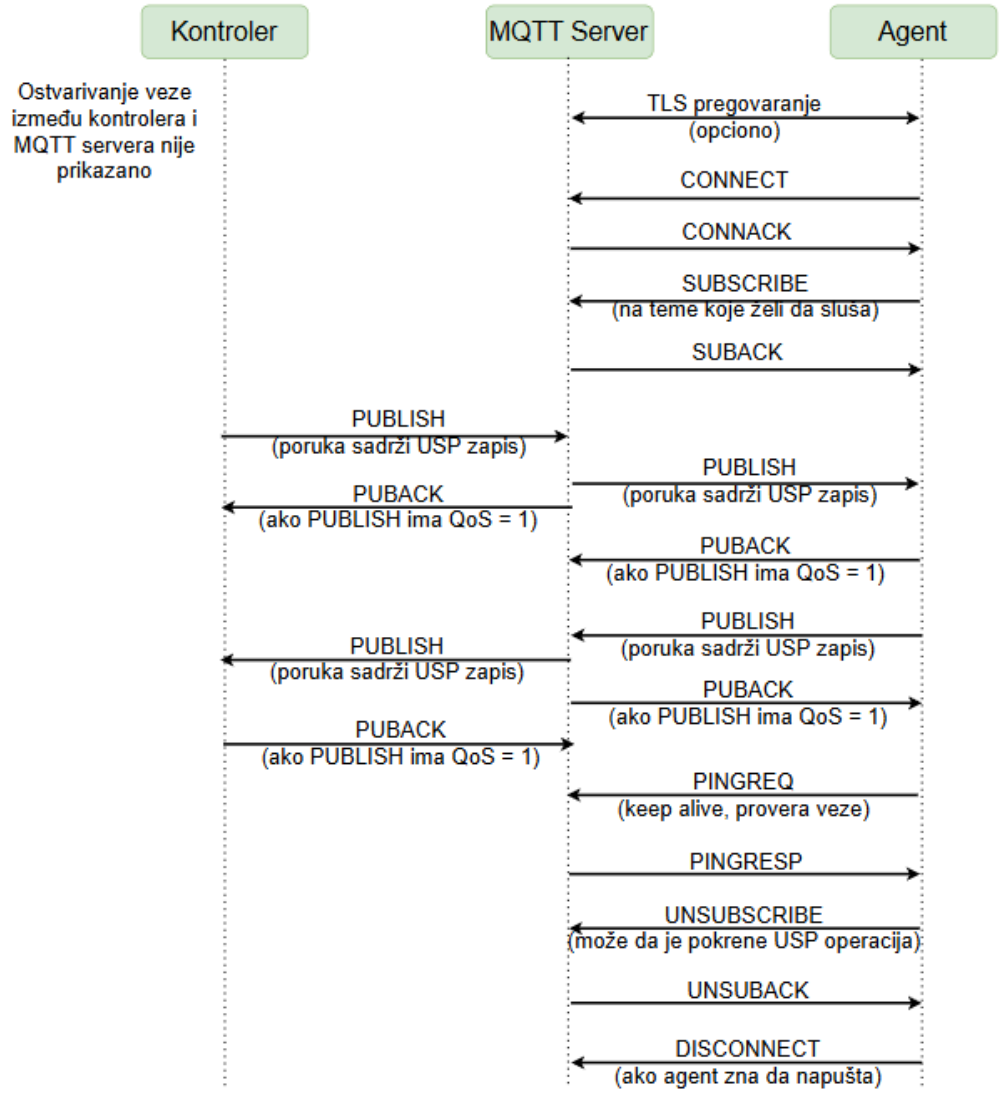
MQTT MTP преноси USP записе између USP крајњих тачака користећи MQTT протокол. Поруке које се преносе између MQTT клијената користе модел интеракције магистрале порука где је MQTT сервер посредник за размену порука који усмерава и испоручује поруке на основу имена теме укљученог у заглавље променљиве MQTT објављивачког пакета.



Слика 2.6 USP преко MQTT проткола

Основни кораци за било коју USP крајњу тачку која користи MQTT као MTP су:

- преговарање TLS-а
- Повезивање на MQTT сервер
- Претплата на тему
- Објављивање USP поруке
- Опционо слање PINGREQ поруке за одржавање везе



Слика 2.7 Кораци успостављања MQTT везе

2.3 Андроид

Андроид платформа је данас најраспрострањенија платформа за мобилне уређаје, базирана на Linux језгру. Прилагођена је за широку употребу на различитим мобилним уређајима, укључујући телефоне, таблете, као и лаптоп рачунаре. Поред тога, нашла је примену и у другим специјализованим уређајима као што су паметни сатови, паметни телевизори и пријемници дигиталног ТВ сигнала.

2.3.1 Сервис

Андроид сервиси су кључни елементи Андроид платформе који омогућавају апликацијама да раде у позадини, чак и када корисник није директно интерактиван са апликацијом. Сервиси омогућавају извршавање дуготрајних операција, као што су обрада података, репродукција медијских садржаја или синхронизација података са удаљеним серверима, без потребе да корисник одржава активни интерфејс апликације. Они се покрећу као независни процеси и могу се користити за разне задатке унутар апликације, чиме доприносе бољем корисничком искуству и ефикаснијем раду система у целини. Важно је напоменути да се Андроид сервиси могу конфигурисати да раде у различитим режимима у зависности од потреба апликације и уређаја на којем се користи, чиме пружају флексибилност и оптимизацију перформанси.

2.3.1.1 Апликативни сервиси

Ови сервиси се инсталирају накнадно са апликацијама и обично служе специфичним потребама апликације. Локални сервиси су подврста апликативних сервиса и извршавају се у истом меморијском простору као и апликација која их је покренула. За комуникацију са локалним сервисима користе се два основна начина:

- **методе startService() и stopService()**: Метода startService() креира нови сервис и позива његове методе onCreate() и onStartCommand().Ова метода се користи када апликација треба да покрене сервис који ради у позадини и обавља одређене задатке.
- **метода bindService()**: Ова метода се користи за везивање апликације за сервис, омогућавајући апликацији да комуницира са сервисом и извршава функције које сервис пружа.

Реализација локалног сервиса обухвата неколико корака:

Креирање класе сервиса: Дефинише се класа која наслеђује Service класу и имплементира потребне методе сервиса.

- **Регистрација у манифесту**: Сервис се мора регистровати у AndroidManifest.xml фајлу како би систем био свестан његовог постојања и како би га апликација могла користити.
- **Имплементација корисничког кода**: Потребно је написати одговарајући код унутар апликације који користи функционалности које сервис пружа, као што су обрада података у позадини или управљање дуготрајним операцијама.

Коришћењем Андроид сервиса, апликације могу ефикасно обављати задатке у позадини без директног ангажовања корисника, чиме се побољшава укупно корисничко искуство и оптимизују перформансе апликације.

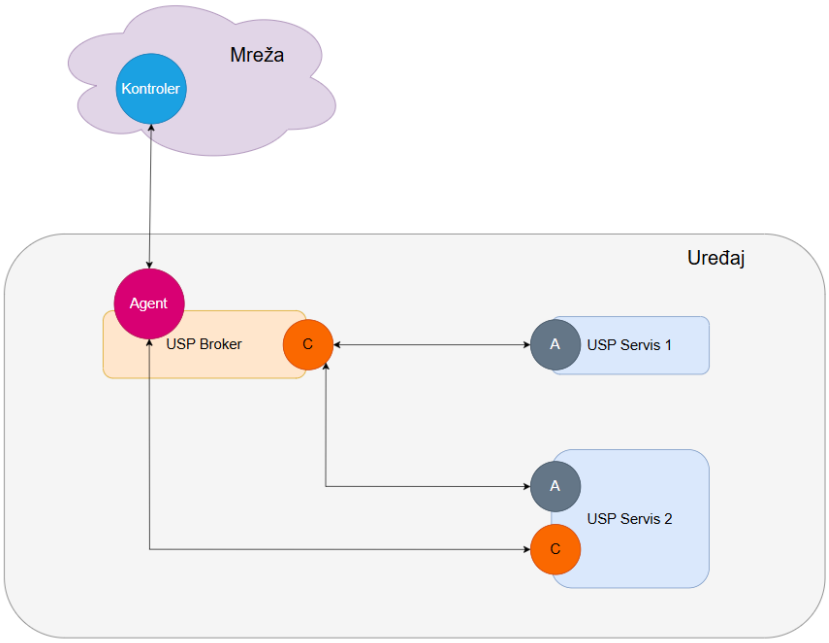
2.3.2 JNI

JNI омогућава Јава програмима да комуницирају са другим програмима и библиотекама написаним у језицима попут C/C++, као и да буду позивани од стране тих програма. Ова платформа се користи када је потребно да се Јава апликацији омогући приступ специфичним оперативним системским функцијама или када је потребно интегрисати постојећи код написан у другом језику. Главна предност коришћења JNI је у побољшању перформанси, јер се C/C++ код обично брже извршава од Јава кода. Међутим, употреба JNI-а може довести до губитка Јава карактеристичне преносивости, јер код који се користи преко JNI-а мора бити унапред преведен за сваки циљни оперативни систем. Такође, употреба JNI-а може увести сигурносне ризике, јер је код написан у језицима као што су C и C++ мање сигуран од Јава кода који пружа додатне нивое заштите као што је управљање меморијом аутоматским сакупљањем смећа (eng. Garbage Collection). Неправилности у нативном коду могу директно утицати на стабилност и безбедност целокупне Јава апликације. Препорука је да се JNI користи само када је то заиста неопходно, да се нативни код изолује од остатка апликације колико год је могуће, и да се примени опрез приликом развоја и испитивања апликација које користе ЈНИ. Важно је напоменути да се нативне методе означавају са кључном речи **native** и имплементирају као функције у динамичким библиотекама. За генерисање прототипа ових функција користи се алат SWIG, који је често коришћен за интеграцију C/C++ функционалности са Јава кодом, ослањајући се на декларације нативних метода у Јава коду. **SWIG** омогућава аутоматско генерисање интерфејса између Јава и других програмских језика попут C/C++, што олакшава позивање функција имплементираних у другим језицима из Јава апликација.

3. Концепт решења

3.1 Компоненте система

Рад се састоји из три целине. Прва целина је USP Агент реализован као библиотека, која имплементира функционалности TR-369 клијента и користи се као део система који обухвата прикупљање и складиштење информација о СТБ уређају, као и њихово прослеђивање USP контролеру. Друга целина је Јава сервис који је задужен за учитавање USP Агента реализованог као библиотеку на покретању система СТБ уређаја, као и за ажурирање података USP Агента. Последња целина је сам USP контролер који комуницира са USP Агентима СТБ уређаја који су повезани у систем и приказује нам њихове податке.



Слика 3.1 Пример USP уређаја

3.2 USP Агент

USP Агентска библиотека је модификација USP Агента из репозиторијума отвореног кода креираног од стране Броадбанд Форум-а, фокусирана на креирању имплементације USP спецификације из угла Агента. USP Агент је задужен за откривање скупа сервисних елемената тј. модела података сачињеног од објеката и параметара који описују неку специфичну функционалност за потребе USP контролера. Иако је USP способан да буде коришћен у широком спектру окружења, кућна мрежа је најчешћа, и том окружењу USP агент се налази као део CPE опреме, као што су рутер, приступна тачка, ИоТ пролаз или у овом случају СТБ уређај.

3.2.1 Кораци

Да бисмо могли користити USP Агент на СТБ платформи, прво је потребно да преведемо изворни код свих библиотека од којих USP Агент зависи, као и саму агентску апликацију за специфичну платформу (eng. Cross-compile). Након тога, проширујемо модел података са нашим објектима и параметрима. То значи да прилагођавамо или додајемо нове елементе у структуру података који користи апликација, како би одговарала специфичним захтевима или потребама наше имплементације. Следећи корак је конфигурација клијента. То укључује подешавање MTP-а који клијент користи, начине повезивања и комуникације, интерне контролере, и остале конфигурације које омогућавају функционалности прилагођене нашим потребама. Испитујемо функционалност апликације на СТБ платорми као засебне апликације и затим је преводимо као библиотеку коју ће Јава сервис користити преко JNI-а.

3.3 Јава сервис за покретање нативног C кода

Јава сервис који интегрише и покреће нативни C код користи SWIG за генерисање Јава и C кода омотача, омогућавајући Јава апликацији директно позивање функција имплементираних у C-у.

SWIG се користи за аутоматско генерисање Јава омотача који омогућавају комуникацију између Јава и нативног C кода. У SWIG интерфејс фајлу дефинишемо потребне функције и њихове параметре којима желимо да приступимо из Јава апликације. Јава сервис је структуриран да покреће три инстанце USP Агентске апликације:

Брокер инстанца има улогу да прати и посредује комуникацији између сервиса унутар уређаја. Такође има улогу да спољним USP контролерима пружи сједињен модел података који сачињен спајањем модела података свих сервиса на уређају. Комуникација између брокера и сервиса на уређају се врши преко UNIX сокета, док са спољним USP контролерима комуницира преко MQTT протокола, одговарајући на CRUD-ON захтеве контролера, команде, или слање аутоматских ажурирања USP контролерима који су претплаћени на његове теме.

Сервисна инстанца је одговорна за имплементацију и одржавање модела података и његових проширења, остварује комуникацију искључиво са брокером који модел података даље открива спољним контролерима. Комуникација са брокером се дешава преко UNIX утичница, на којем брокер шаље поруке сервису**.**

Трећа инстанца је задужена за ажурирање података унутар модела података, која на временски интервал проверава да ли су настале промене у неким параметрима, и ако промена вредности постоји, нове вредности прослеђује сервису који ажурира вредности унутар сервиса који преко UNIX утичнице обавештава брокер, који онда обавештава USP контролер који је претплаћен на тему о променама вредности параметара**.**

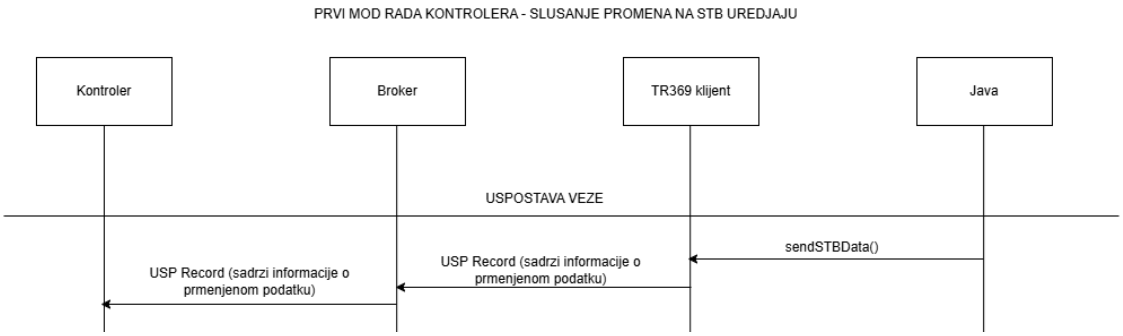
3.4 USP Контролер

USP контролер је проширење USP Агентске апликације са додатим контролерским функционалностима. Контролер се налази на удаљеном уређају и служи као централна тачка за прикупљање информација о свим СТБ уређајима који су део система..

3.4.1 Режим рада

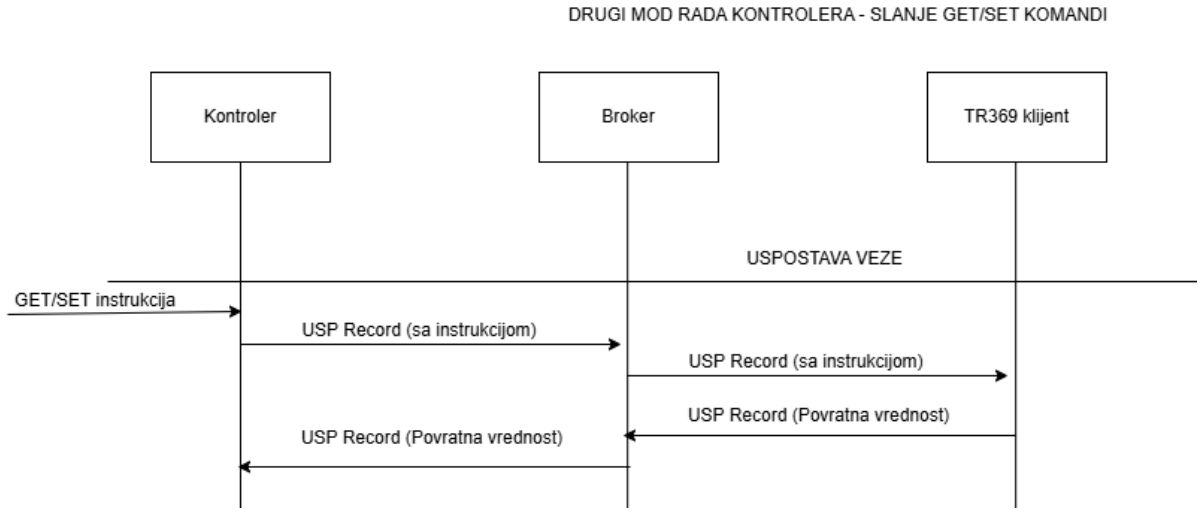
Контролер може да ради у једном од два режима рада које налаже протокол:

Континуирани режим рада: У овом режиму, сервер је непрекидно активан. Он остаје укључен и активно ослушкује обавештења na MQTT темама на које је претплаћен. Ово омогућава серверу да одмах реагује на промене на СТБ уређајима, пружајући обавештења о промени вредности или статусу уређаја.



Слика 3.2 Први мод рада контролера

Режим рада на захтев: Овај режим се активира само када се захтевају информације о одређеном СТБ уређају. Контролер се покреће само у тренутку када је потребно прикупити податке или извршити одређену акцију на захтев, након чега прекида свој рад до следећег захтева. Ово смањује непотребну потрошњу ресурса и енергије када сервер није активно потребан.



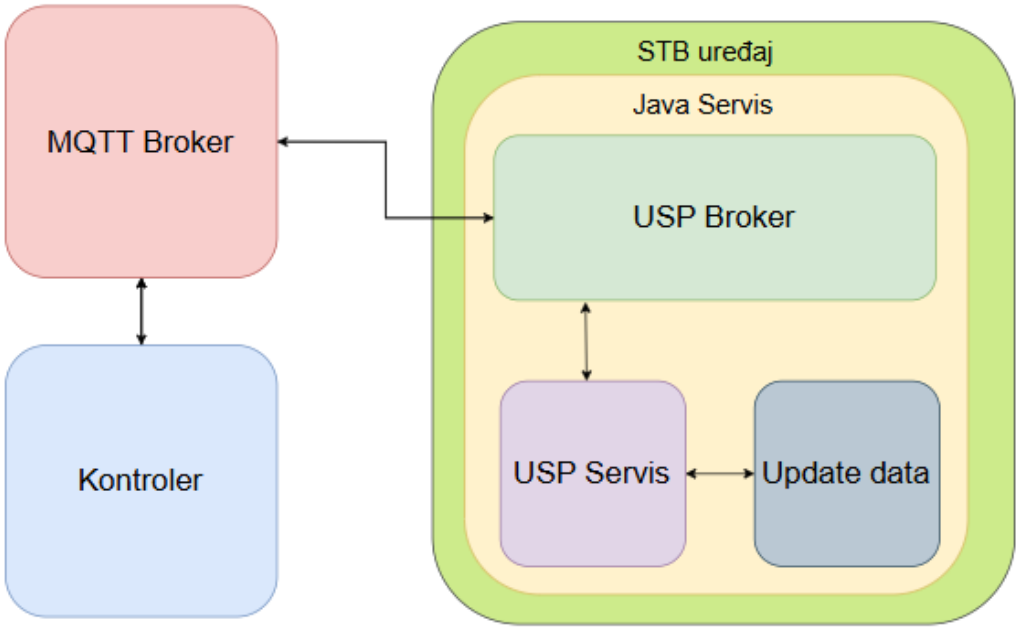
Слика 3.3 Други мод рада контролера

4. Програмско решење

У овом поглављу је описано програмско решење које имплементира функционалности клијента и контролера заснованог на TR-369 стандарду. Решења су реализована у програмским језицима C и Јава. Такође је важно напоменути зависност од библиотека: libc, libcrypto, libm, libmosquitto, libsqlite3, libssl, libpthread, libz.

TR-369 клијент и контролер се заснивају на раду више нити:

- Нит модела података – задужена за све USP API позиве над моделом података
- MTP нити – нити задужена за комуникациони протокол, код контролера само MQTT, код брокера MQTT и UDS, и код сервиса само UDS нит



Слика 4.1 Скица реалног система

4.1 USP Агент

USP Агентско решење се састоји од две целине, C кода и Јава сервиса. C код је потребно превести за циљану платформу као библиотеку, као и све библиотеке од којих агент зависи. C код има три режима рада: брокер, сервис и инстанца за ажурирање података.

4.1.1 USP Брокер

Као улазна функција у брокер развијена је функција:

```
int MainBroker (int argc, char *argv[])
```

Почетна функција када се покреће брокер из Јава сервиса, као параметре прима дужину низа параметара и сам низ параметара. Први корак је разрешавање улазних параметара и додељивање вредности променљивама вредностима из низа улазних параметара. Покреће основне подсистеме за праћења тока програма, обраду грешке, и узајамно искључење.

Након тога покреће функције преузете из решења отвореног кода:

```
int MAIN_Start(char *db_file, bool enable_mem_info);
```

Параметри функције су путања до базе података и информација да ли треба да се прикупљају подаци о употреби меморије за праћење грешака. Иницијализује тајмере који се користе за периодично проверавање промена вредности, иницијализује базу података позивом функције:

```
int DATABASE_Init(char *db_file)
```

За параметар прима путању до фајла базе података коју треба да иницијализује, ако база не постоји креира је на основу конфигурационог фајла који је прослеђен као улазни параметар програма. Враћа код грешке ако постоји, ако не - враћа 0.

```
int DM_EXEC_Init(void)
```

Нема улазне параметре, и функција јој је да иницијализује утичнице за главни ред порука и иницијализује mutex. Враћа код грешке ако постоји, ако не - враћа 0.

```
int MTP_EXEC_Init(void)
```

Иницијализује сокете за MTP који се користе за ред порука бућења у овом случају за MQTT и UDS протоколе. Враћа код грешке ако постоји, ако не - враћа 0.

```
int DATA_MODEL_Init(void)
```

Иницијализује модел података и региструје све чворове у шеми модела података. Враћа код грешке ако постоји, ако не - враћа 0.

```
int DATA_MODEL_Start(void)
```

Покретање свих инстанци модела података, додавање свих неопходних објеката и параметара, као и објекте и параметре дефинисане од стране корисника у базу. Враћа код грешке ако постоји, ако не - враћа 0.

```
int DEVICE_CONTROLLER_StartAllMtpClients(void);
```

Покреће све MTP клијенте у стању где им је дозвољено повезивање, у овом случају MQTT клијент, региструје теме на које је претплаћен и на које објављује, региструје функције повратне спреге, нотификације. Ако им није дозвољено повезивање покреће тајмер који периодично проверава да ли је повезивање омогућено. Најбитније функције повратне спреге функције:

```
void SetupCallbacks(mqtt_client_t *client)
```

Улазни параметар је показивач на инстанцу MQTT клијента. Региструје све callback функције за MQTT клијента.

```
void ConnectCallback(struct mosquitto *mosq, void *userdata, int result);
```

Параметри су контекст libmosquitto библиотеке, показивач на инстанцу MQTT клијента у бази модела података, код стања из CONNACK поруке, и особине примљене у CONNACK поруци

```
void DisconnectCallback(struct mosquitto *mosq, void *userdata, int rc);
```

Позива се од стране библиотеке када се утичница одвеже. Параметри су контекст из libmosqutto библиотеке, показивач на instancu MQTT клијента у бази модела података, као и разлог искључивања.

```
void SubscribeCallback(struct mosquitto *mosq, void *userdata, int mid, int
qos_count, const int* granted_qos)
```

Позива се од стране библиотеке када је примљен SUBACK пакет. Параметри су контекст из libmosquitto библитеке, показивач на instancu MQTT клијента у бази модела података, идентификациони број теме на коју се претплаћујемо, остали параметри нису коришћени.

```
void UnsubscribeCallback(struct mosquitto *mosq, void *userdata, int mid );
```

Позива се од стране библиотеке када је примљен UNSUBACK пакет. Параметри су контекст из libmosquitto библитеке, показивач на инстанцу MQTT клијента у бази модела података, идентификациони број теме са којом прекидамо претплату.

```
void PublishCallback(struct mosquitto* mosq, void *userdata, int mid)
```

Позива се од стране библиотеке када је објава послата (према условима QoS). Параметри су контекст из libmosquitto библитеке, показивач на инстанцу MQTT клијента у бази модела података, идентификациони број поруке која је послата.

```
void MessageCallback(struct mosquitto *mosq, void *userdata, const struct
      mosquitto_message *message, const mosquitto_property *props);
```

Позива се од стране библиотеке када је примљена порука. Параметри су контекст из libmosquitto библитеке, показивач на instancu MQTT клијента у бази модела података, показивач на објекат поруке библиотеке и показивач на MQTT особине које се налазе унутар поруке. Задатак јој је да прими MQTT поруку и од ње формира USP запис који се прослеђује и даље процесуира у нити модела података.

```
int OS_UTILS_CreateThread(const char* name, void *(* start_routine)(void *), void
      *args)
```

Функција омотач која покреће POSIX нит, први параметар је име нити, у овом случају "MTP_MQTT" или "MTP_UDS", други параметар је показивач на главну функцију нити у овом случају то су:

```
void *MTP_EXEC_MqttMain(void *args)
```

Главна петља MTP нити за MQTT протокол у којој креира утичнице, на којим чека пријем нове поруке, коју даље прослеђује на обраду.

```
void *MTP_EXEC_UdsMain(void *args)
```

Главна петља MTP нити за UDS протокол у којој креира утичнице, на којим чека пријем нове поруке, коју даље прослеђује на обраду.

```
void *DM_EXEC_Main(void *args)
```

Главна петља модела података покреће сервер спреге командне линије, где се повезује на дефинисану UNIX утичницу и отпочиње слушање. Након тога покреће бесконачну петљу без излазних услова где чека на активност на утичницама контролера како би даље обрадили податке и команде, прослеђивајући их до сервиса коме су намењене, покреће све тајмере који су иницијализовани и спремни за рад, и затим промене о којима су тајмери послали обавештења обрађује.

4.1.2 USP Сервис

USP сервис има редукован сет функционалности јер му је омогућено само комуницирање са USP брокером преко UDS протокола, јер само USP брокер комуницира са спољним контролерима preko MQTT протокола. Као улазна функција сервиса развијена је функција:

```
int MainService(int argc, char *argv[]);
```

Почетна функција када се покреће USP сервис из Јава сервиса, као параметре прима дужину низа параметара и сам низ параметара. Први корак је разрешавање улазних параметара и додељивање вредности променљивама вредностима из низа улазних параметара. Покреће основне подсистеме за праћења тока програма, обраду грешке, и узајамно искључење (eng. mutex)

Након тога покреће функције преузете из решења отвореног кода:

```
int MAIN_Start(char *db_file, bool enable_mem_info);
```

Параметри функције су путања до базе података и информација да ли треба да прикупљају подаци о меморији за праћење грешака. Иницијализује тајмере који се користе за периодично проверавање промена вредности, иницијализује базу података позивом фунцкије.

```
int DATABASE_Init(char *db_file);
```

За параметар прима путању до фајла базе података коју треба да иницијализује, ако база не постоји креира је на основу конфигурационог фајла који је прослеђен као улазни параметар програма. Враћа код грешке ако постоји, ако не - враћа 0.

```
int DM_EXEC_Init(void)
```

Нема улазне параметре, и функција јој је да иницијализује утичнице за главни ред порука и иницијализује mutex. Враћа код грешке ако постоји, ако не - враћа 0.

```
int DATA_MODEL_Init(void)
```

Иницијализује модел података и региструје све чворове у шеми модела података. Враћа код грешке ако постоји, ако не - враћа 0.

```
int DATA_MODEL_Start(void)
```

Покретање свих инстанци модела података, додавање свих неопходних објеката и параметара, као и објекте и параметре дефинисане од стране корисника у базу. Враћа код грешке ако постоји, ако не - враћа 0.

```
void *DM_EXEC_Main(void *args)
```

Главна петља модела података покреће сервер спреге командне линије, где се повезује на дефинисану UNIX утичницу и отпочиње слушање. Након тога покреће бесконачну петљу без излазних услова где чека на активност на утичницама контролера како би даље обрадили податке и команде, како би изменили модел података или обавестили брокер, покреће све тајмере који су иницијализовани и спремни за рад, и затим промене о којима су тајмери послали обавештења обрађује.

Корисник додаје нове параметере функцијама преузетим из решења отвореног кода:

```
int USP_REGISTER_Object(char*, dm_validate_add_cb_t, dm_add_cb_t,
     dm_notify_add_cb_t, dm_validate_del_cb_t, dm_del_cb_t, dm_notify_del_cb_t);
```

Региструје инстанцу објекта модела који корисник може да унесе у базу преко контролера. Параметри функције су путања објекта модела, функција повратне спреге која проверава да ли контролер може да дода инстанцу објекта, функција повратне спреге која се користи ако је објекат додат у коду од стране програмера који проширује модел и поставља почетне вредности, функција повратне спреге која обавештава када је инстанца креирана, функција повратне спреге која проверава да ли контролер може да обрише инстанцу објекта и функција повратне спреге која обавештава да је инстанце објекта обрисана. Враћа код грешке ако постоји, ако не - враћа 0.

```
int USP_REGISTER_DBParam_ReadWrite(char *, char *, dm_validate_value_cb_t,
     dm_notify_set_cb_t, unsigned);
```

Региструје параметер модела који корисник уноси у базу. Параметри функције су путања до параметра модела, почетна вредност, функција повратне спреге која проверава исправност унешених података, функција повратне спреге која шаље обавештење када је вредност промењена, и тип параметра. Враћа код грешке ако постоји, ако не - враћа 0.

За проширење модела података потребно је модификовати функцију:

```
int VENDOR_Init(void)
```

Иницијализује све додатне параметре и функције са којима желимо да проширимо модел података. Враћа код грешке ако постоји, ако не - враћа 0.

За потребе рада развијене су функције:

```
int DEVICE_STB_Init()
```

Региструје објекат модела података са којим проширујемо модел и његове параметре са функцијама поврате спреге које реагују на промене и догађаје везане за уведене параметре. Враћа код грешке ако постоји, ако не - враћа 0.

```
int NotifySTBDataChange(dm_req_t *re, char *value)
```

Функција повратне спреге која се активира када дође до промене вредности парамета којима смо проширили модел и исписује настале промене. Враћа код грешке ако постоји, ако не - враћа 0.

4.1.3 Ажурирање података

```
int SendStbData(int argc, char *argv[]);
```

Главна функција инстанце USP Агента задужена за CRUD-ON операције над моделом података. Као параметре прима дужину низа параметара и сам низ параметара. Први корак је разрешавање улазних параметара и додељивање вредности променљивама вредностима из низа улазних параметара. Враћа код грешке ако постоји, ако не - враћа 0.

Након тога покреће функцију преузету из решења отвореног кода:

```
int CLI_CLIENT_ExecCommand(int argc, char *argv[], char *db_file);
```

Параметри функције су број аргумената команде, показивач на први аргумент низа команде и база модела података сервиса у којем се део модела који модификујемо налази. Извршава команду, слањем задате команде серверу спреге командне линије активног USP Агента. Враћа код грешке ако постоји, ако не - враћа 0.

4.2 Имплементација Јава сервиса са JNI за USP Агента

Имплементација Јава сервиса за USP агент представља кључни део архитектуре система за управљање уређајима у мрежи. Овај сервис омогућава интеграцију Јава апликације са нативним C кодом путем JNI, што омогућава ефикасан рад USP Брокера и USP Сервиса на СТБ уређају. Јава сервис је имплементиран као сервис који се покреће као позадински процес. Главни задаци сервиса укључују покретање и управљање USP Брокером и USP Сервисом, као и периодично слање података контролеру.

При покретању сервиса прво се учитавају библиотеке од којих USP Брокер и USP Сервис зависе, затим и саме JNI библиотеке, које су генерисане уз помоћ SWIG интерфејс фајлова у којима су дефинисане које функције нативног C кода се откривају Јава сервису. Да би им приступ био омогућен учитавају се функцијом **System.LoadLibrary()**. Након тога се учитавају почетне вредности свих фајлова и параметара потребих за иницијализацију инстанци USP Агента као што су базе модела података, фајлови спреге командне линије, конфигурациони фајлови брокера и сервиса, сертификати, и мрежне информације о уређају. За добављање тих информација развијамо функције:

```
void initData(void);
```

Функција која се покреће на почетку извршавања програма која креира фајлове и додељује вредности променљивама као што су конфигурације и сертификати који су неопходни за покретање инстанци USP Агента.

```
String findNetworkInterface(void);
```

Проналази први активни мрежни интерфејс који није loopback и враћа његово име. Ова информација је кључна за даље операције повезане са мрежом тј. комуникацију са серверима или другим уређајима.

```
void copyAssetToFile(String assetName, File destinationFile);
```

Помоћна функција која ефикасно преноси датотеке iz assets фолдера у системски фајл, уз додатак специфичне обраде за одређене датотеке. Ова функционалност је корисна за динамичко прилагођавање конфигурација на основу тренутних података, као што су путања до UNIX утичнице за интерну комуникацију између брокера и сервиса, и MAC адреса, добијена из нативне функције:

```
char* GetMacAddress(char* network);
```

```
JSONObject updateData();
```

Ефикасно прикупља и структурира кључне информације о стању уређаја у форми JSON објекта. Овај JSON објекат се користи за ажурирање унутар модела података, чиме се омогућава динамичко праћење и управљање уређајем.

4.2.1 Конфигурација брокера и сервиса

Понашање брокера и сервиса одређују њихови конфигурациони фајлови, што чини програмску подршку изузетно једноставном за модификовање у складу са променљивим захтевима. Потребни параметри су идентификациони број крајње тачке која се користи као идентификатор у комуникацији са другим уређајима или унутар уређаја са брокером/сервисима. Клијент дефинише информације о брокеру протокола, типу протокола, и информације које су потребе за повезивање на исти, као и идентификациони параметар који је јединствен за сваки СТБ уређај. Интерни контролер који је задужен за комуникацију, у њему се дефинише протокол, у овом случају MQTT, теме на које објављује, и референца на клијента које садржи информације о брокеру протокола. Локални агент нам конфигурише на које теме се који клијент претплаћује, као и тип побуде који покреће слање порука, нпр. промена вредности унутар модела података. Информације о UNIX утичници као што су његова локација, режим рада, да ли је инстанца сервер или клијент. Конфигурација може да има произвољан број клијената, интерних контролера, локалних агената, самим тим омогућава да USP Агент користи више протокола истовремено и са сваким има одвојене теме за комуникацију.

```
Device.LocalAgent.EndpointID "usp::client"

Device.LocalAgent.Controller.1.BootParameter.1.Enable true
Device.LocalAgent.Controller.1.BootParameter.1.ParameterName "Device.LocalAgent.EndpointID"
Device.LocalAgent.Controller.1.Enable true
Device.LocalAgent.Controller.1.Alias "controller"
Device.LocalAgent.Controller.1.EndpointID "usp::controller"
Device.LocalAgent.Controller.1.MTP.1.Enable true
Device.LocalAgent.Controller.1.MTP.1.Protocol "MQTT"
Device.LocalAgent.Controller.1.MTP.1.MQTT.Reference "Device.MQTT.Client.1"
Device.LocalAgent.Controller.1.MTP.1.MQTT.Topic "usp/fromClient"

Device.LocalAgent.Subscription.1.Enable true
Device.LocalAgent.Subscription.1.ID {placeholder}
Device.LocalAgent.Subscription.1.Recipient Device.LocalAgent.Controller.1
Device.LocalAgent.Subscription.1.NotifRetry false
Device.LocalAgent.Subscription.1.NotifType ValueChange
Device.LocalAgent.Subscription.1.ReferenceList Device.STBData.
Device.LocalAgent.Subscription.1.Persistent true

Device.LocalAgent.MTP.1.Alias "client"
Device.LocalAgent.MTP.1.Enable true
Device.LocalAgent.MTP.1.Protocol "MQTT"
Device.LocalAgent.MTP.1.MQTT.ResponseTopicConfigured "usp/fromController/{placeholder}"
Device.LocalAgent.MTP.1.MQTT.Reference "Device.MQTT.Client.1"

Device.UnixDomainSockets.UnixDomainSocket.1.Alias "broker"
Device.UnixDomainSockets.UnixDomainSocket.1.Mode "Listen"
Device.UnixDomainSockets.UnixDomainSocket.1.Path "{unix_socket}"

Device.MQTT.Client.1.BrokerAddress "192.168.235.63"
Device.MQTT.Client.1.ProtocolVersion "5.0"
Device.MQTT.Client.1.BrokerPort "8883"
Device.MQTT.Client.1.TransportProtocol "TCP/IP"
Device.MQTT.Client.1.Username ""
Device.MQTT.Client.1.Password ""
Device.MQTT.Client.1.Alias "client"
Device.MQTT.Client.1.Enable true
Device.MQTT.Client.1.ClientID "D0:5A:00:00:5F:90"
Device.MQTT.Client.1.KeepAliveTime "600"

Device.MQTT.Client.1.ConnectRetryTime "5"
Device.MQTT.Client.1.ConnectRetryIntervalMultiplier   "2000"
Device.MQTT.Client.1.ConnectRetryMaxInterval "60"

Internal.Reboot.Cause "LocalFactoryReset"
```

Слика 4.2 Конфигурација брокера

```
Device.LocalAgent.EndpointID "usp::service_test"
Device.UnixDomainSockets.UnixDomainSocket.1.Alias "broker"
Device.UnixDomainSockets.UnixDomainSocket.1.Mode "Connect"
Device.UnixDomainSockets.UnixDomainSocket.1.Path "{unix_socket}"

Internal.Reboot.Cause "LocalFactoryReset"
```

Слика 4.3 Конфигурација сервиса

4.2.2 Нити

Сервис користи три нити које раде паралелно, како не би блокирали извршавање главне нити.

Нит брокера покреће USP Брокер који управља комуникацијом са централним контролером и сервисима. Параметри који се прослеђују су дужина низа параметара и сам низ који садржи путању до базе модела података, конфигурациони фајл, мрежна спрега, фајл спреге командне линије, опциони параметри су сертификати, омогућавање протобуф исписа као и испис грешака и тока извршења програма. Извршавање траје док не настане прекид грешком, који се евидентира и се поново покушава покретање функције.

Нит сервиса покреће USP Сервис који комуницира са USP брокером и одржава базу модела података. Параметри који се прослеђују су дужина низа параметара и сам низ који садржи путању до базе модела података, конфигурациони фајл, мрежну спрега, фајл спреге командне линије, опциони параметри протобуф испис USP порука и записа, као и испис грешака и тока извршења програма. Извршавање траје док не настане прекид грешком, који се евидентира и поново се покушава покретање функције.

Нит за ажурирање података има улогу ажурирања параметара унутар модела података. Периодично проверава вредности и ако постоји промена обавештава сервис о томе, који податке шаље даље брокеру. Ако је успешно послат и извршена промена, враћа 0, и брокер те информације прослеђује удаљеном контролеру, у супротном евидентира код грешке.

4.3 USP Контролер

USP Контролер је задужен за комуникацију са више USP агената на удаљеним уређајима и прикупљање података о тим уређајима. Контролер омогућава централизовано управљање и надзор, користећи стандардне протоколе за размену порука као што је MQTT. У наставку су описане битне функције и начин њихове имплементације у оквиру USP Контролера.

У првом режиму рада контролер се понаша идентично као брокер, где је једина разлика што не ажурира модел података приликом промене вредности параметара већ промене бележи и исписује у конзолу.

У другом режиму се разликује јер постоји додатни улазни параметар који чини захтев за добијање, промену, додавање или брисање параметара модела. Наведене функције су дефинисане у решења отвореног кода:

```
int CTRL_FILE_PARSER_Start(char *controller_file, char *db_file)
```

Функција која парсира захтев за слање поруке USP агенту, као параметре прима фајл са захтевом и локацију базе података. Када препозна команду позива одговарајућу функцију за формиранје USP поруке. Враћа код грешке ако постоји, ако не - враћа 0.

```
int MSG_HANDLER_QueueMessage(char *endpoint_id, Usp__Msg *usp, mtp_reply_to_t *mrt)
```

Функција која серијализује USP поруку у бафер, улаз су јој контролер са којег шаљемо поруку, протобуф структура са USP поруком, и детаљи где одговор треба да буде послат. Враћа код грешке ако постоји, ако не - враћа 0.

```
int MSG_HANDLER_QueueUspRecord(Usp__Header__MsgType usp_msg_type, char
*endpoint_id, unsigned char *pbuf, int pbuf_len, char *usp_msg_id, mtp_reply_to_t
*mrt, time_t expiry_time)
```

Функција серијализује протобуф USP запис структуру у бафер, са енкапсулираном USP поруком, и ставља је у ред чекања за слање. Параметри су тип поруке, контролер са којег шаљемо поруку, показивач на бафер са поруком, дужина бафера, детаљи о одговору и време истека после којег се порука брише ако није послата. Враћа код грешке ако постоји, ако не - враћа 0.

Контролер чека одређени временски интервал на одговор, ако одговор не стигне контролер се гаси са кодом грешке.

```
void MessageCallback(struct mosquitto *mosq, void *userdata, const struct
    mosquitto_message *message, const mosquitto_property *props);
```

Позива се од када је примљена порука. Параметри су контекст из libmosquitto библитеке, показивач на instancu MQTT клијента у бази модела података, показивач на објекат поруке библиотеке и показивач на MQTT особине које се налазе унутар поруке. Задатак јој је да прими MQTT поруку и од ње формира USP запис који се прослеђује и даље обрађује у нити модела података.

```
void DM_EXEC_PostUspRecord(unsigned char *pbuf, int pbuf_len, ctrust_role_t role,
mtp_reply_to_t *mrt)
```

Шаље USP запис добијен као одговор у нит модела података за даље процесирање. Параметри су показивач на бафер са записом, дужина бафера, ранг дозволе за примање поруке, детаљи о одговору.

```
void PrintProtobufCMessageRecursive(ProtobufCMessage *msg, int indent)
```

Исписује поруку у протобуф формату, позива се рекурзивно. Параметри су протобуф порука и ниво индентације.

```
msg_id:"1" to_id:"usp::client" mqtt_topic:"usp/fromController/D0:5A:00:00:5F:90" mqtt_instance:"1"
msg_type:"Get" param_paths:"Device.STBData.IPAddress" param_paths:"Device.STBData.Serial"
msg_type:"GetSupportedDM" obj_paths:"Device." first_level_only:"false"
        return_commands:"true" return_events:"true" return_params:"true"
```

Слика 4.4 Пример захтева

```
Device.LocalAgent.EndpointID "usp::controller"

Device.MQTT.Client.1.Enable true
Device.MQTT.Client.1.ProtocolVersion "5.0"
Device.MQTT.Client.1.BrokerAddress "localhost"
Device.MQTT.Client.1.BrokerPort "8883"
Device.MQTT.Client.1.TransportProtocol "TCP/IP"
Device.MQTT.Client.1.ClientID "controller"
Device.MQTT.Client.1.Username ""
Device.MQTT.Client.1.Password ""
Device.MQTT.Client.1.KeepAliveTime "600"

Device.LocalAgent.MTP.1.Alias "controller"
Device.LocalAgent.MTP.1.Enable true
Device.LocalAgent.MTP.1.Protocol "MQTT"
Device.LocalAgent.MTP.1.MQTT.Reference "Device.MQTT.Client.1"
Device.LocalAgent.MTP.1.MQTT.ResponseTopicConfigured "usp/fromClient"

Device.LocalAgent.Controller.1.Enable true
Device.LocalAgent.Controller.1.Alias "client"
Device.LocalAgent.Controller.1.EndpointID "usp::client"
Device.LocalAgent.Controller.1.MTP.1.Enable true
Device.LocalAgent.Controller.1.MTP.1.Protocol "MQTT"
Device.LocalAgent.Controller.1.MTP.1.MQTT.Reference "Device.MQTT.Client.1"
Device.LocalAgent.Controller.1.MTP.1.MQTT.Topic "usp/fromController"

Internal.Reboot.Cause "LocalFactoryReset"
```

Слика 4.5 Конфигурација контролера

5. Резултати

Након модификације решења извршене су провере функционалности контролера скриптама за испитивање, као и покретањем контролера и брокер на удаљеним уређајима и агента на СТБ уређају.

5.1 Скрипте

Скрипте омогућавају детаљно испитивање различитих аспеката MQTT протокола, као што су аутентификација, шифровање, управљање везама, и руковање претплатама на теме. Скрипте су преузете из готовог решења отвореног кода, и важне јер помажу у идентификацији и отклањању потенцијалних проблема у ранијим фазама развоја, чиме се смањује ризик од проблема у продукционом окружењу. Једна од кључних компоненти у овом процесу је скрипта shared.sh, која служи као заједнички ресурс за остале скрипте за испитивање функционалности система. Скрипта **shared.sh** садржи основне функције и конфигурације које се често користе у различитим испитивањима, као што су постављање окружења, покретање и заустављање сервиса, као и функције за испис и проверу статуса. Скрипту је потребно прекконфигурисати како би одговарала систему који се испитује. Коришћењем **shared.sh** скрипте за испитивање су модуларније и лакше за одржавање, без потребе за понављањем истих поставки у свакој појединачној скрипти.

Назви скрипте	Опис	Резултат
test_dynamic_adding_ clients.sh	Ова скрипта испитује динамичко додавање клијената на MQTT брокер, проверавајући исправност регистрације и функционалност нових клијената у реалном времену.	Прошао
test_keep_alive.sh	Скрипта испитује "keep alive" функционалност у MQTT комуникацији, проверавајући одржавање везе и реакцију система на губитак сигнала.	Прошао
test_password.sh	Скрипта испитује аутентификацију клијената на MQTT брокеру, проверивањем исправности обраде исправних и неисправних лозинки.	Прошао
test_retry.sh	Скрипта испитује механизам поновног покушаја слања порука, проверавајући број покушаја и интервале пре пријаве неуспеха.	Прошао
test_set_parameters.sh	Скрипта испитује постављање параметара за MQTT комуникацију и примену конфигурацијских промена на активне конекције и поруке.	Прошао
test_ssl.sh	Скрипта испитује MQTT комуникацију преко SSL/TLS протокола, проверавајући заштиту података током преноса ради осигурања сигурности и интегритета.	Прошао
test_start_stop.sh	Скрипта испитује покретање и заустављање MQTT сервиса, проверавајући исправно руковање ресурсима и њихово враћање у првобитно стање.	Прошао
test_subscriptions.sh	Скрипта испитује функционалност претплата на теме у MQTT протоколу, провером исправности пријема порука и имплементације механизама управљања претплатама.	Прошао

Табела 5.1 Скрипте

Након извршавања скрипти за испитивање функционалност, се покрећу контролер и агент како би се проверила целокупна функционалност система. Овај корак омогућава симулацију стварне комуникације између уређаја преко MQTT протокола, чиме се потврђује исправност свих компоненти и њихова способност међусобне интеракције у реалним условима. На овај начин, осигуравамо да систем ради како је предвиђено и да су сви елементи исправно интегрисани. Проверава се први режим рада система где контролер само слуша за промене које су настале код удаљеног агента где сервис детектује промене у моделу, обавештава брокер агента који ажурира модел и шаље обавештење удаљеном контролеру да је дошло до промена у моделу података као и нове вредности података.

```
version: "1.3"
to_id: "usp::controller"
from_id: "usp::client"
payload_security: PLAINTEXT
no_session_context {
}

header {
  msg_id: "ValueChange-2024-09-02T17:30:14Z-1"
  msg_type: NOTIFY
}
body {
  request {
    notify {
      subscription_id: "D0:5A:00:00:5F:90"
      send_resp: false
      value_change {
        param_path: "Device.STBData.IPAddress"
        param_value: "192.168.235.11"
      }
    }
  }
}
```

Слика 5.1 Примљена порука о промењеним вредностима параметра модела

Други режим ради по другачијем принципу, удаљени контролер не слуша за промене, него сам шаље захтеве одабраном СТБ уређају и чека одговор. Почетак захтева мора да садржи идентификатор клијената коме се порука шаље као и тема на коју је тај клијент претплаћен.

```
msg_id:"1" to_id:"usp::client" mqtt_topic:"usp/fromController/ID mqtt_instance:"1"
```

Захтеви које могу да шаљу су за добављање параметара модела, мењање вредности параметара модела, додавање нових објеката у структуру модела као и њихово брисање, добављање инстанци објеката модела, добављање информација о подржаном моделу као и о протоколу или слање команди агенту.

Примери порука које контролер може да шаље:

```
msg_type:"Get" param_paths:"Device.STBData.Serial"
msg_type:"Set" allow_partial:"true" update_objs{obj_path:"Device.STBData."
     param:"Serial" value:"ChangedSerial" required:"true"}}
msg_type: "Add" create_objs{obj_path:"Device.LocalAgent.Subscription."
     allow_partial:"true" {param:"Enable" value:"true"}}
msg_type: "Delete" obj_paths:"Device.LocalAgent.Subscription.[ID==\"test123\"]."
msg_type:"GetSupportedDM" obj_paths:"Device." first_level_only:"false"
     return_commands:"true" return_events:"true" return_params:"true"
msg_type:"GetSupportedProtocol" controller_supported_protocol_versions:"1.0"
msg_type:"Operate" command:"Device.Reboot()" command_key:"boot123" send_resp:"true"
```

Слика 5.2 Захтев са промену вредности параметара и одговор агента

Слика 5.3 Захтев за подацима и одговор агента

6. Закључак

Имплементација TR-369 протокола на Андроид СТБ (Сет-Топ Box) уређајима омогућава напредно управљање и праћење ових уређаја унутар мреже, чиме се значајно унапређују перформансе и корисничко искуство у мултимедијалним окружењима. У оквиру овог рада развијен је свеобухватан систем који се састоји од USP агента, Јава сервиса за покретање нативног C кода и USP контролера. USP агент омогућава прикупљање дијагностичких података о СТБ уређајима путем мреже, док Јава сервис користи JNI за интеграцију са нативним C кодом, омогућавајући ефикасно извршавање функција на СТБ уређајима. USP контролер централизује прикупљање информација са свих повезаних СТБ уређаја и омогућава даљинско управљање и надгледање.

На основу резултата овог рада, следећи корак у даљем развоју система је додатно проширење модела података како би се подржале напредне функционалности уређаја, укључујући специфичне команде за управљање уређајима путем TR-369 протокола. Имплементација ових команди омогућила би прецизније и детаљније управљање СТБ уређајима, као и побољшано праћење и дијагностику, чиме би се унапредила контрола над уређајима у стварном времену.

Након проширења модела података и имплементације додатних команди, следећи корак би могао бити развој REST API сервиса који би омогућио интеграцију са постојећим системима за управљање мрежама и уређајима. Овај API сервис би комуницирао са базом података која би складиштила све релевантне информације о СТБ уређајима, укључујући њихове тренутне статусе, конфигурације и дијагностичке податке. Тиме би се омогућило једноставније и ефикасније управљање великим бројем СТБ уређаја, бржа подршка корисницима, као и оптимизација мрежне инфраструктуре.

Развијањем REST API сервиса, систем би постао скалабилан и флексибилан, спреман за будуће интеграције са напредним мрежним и мултимедијалним сервисима. То би омогућило лако прилагођавање и проширење функционалности у складу са развојем технологије и променама у корисничким захтевима, чиме би се значајно побољшало управљање СТБ уређајима у мрежама и обезбедило пружање висококвалитетних услуга крајњим корисницима.

7. Литература

[1] BroadBand Forum - TR-369, Септембар 2024, Доступно на: BBF – TR-369 – The User Services Platform (usp.technology)

[2] Github, OB-USP-Agent repozitorijum, Јун 2024, Доступно на: BroadbandForum/obuspa: OB-USP-AGENT is a system daemon providing a User Services Platform (USP) Agent. https://github.com/BroadbandForum/obuspa/wiki

[3] Github, OB-USP-Agent Test Controller repozitorijum, Јун 2024, Доступно на: BroadbandForum/obuspa-test-controller: A test USP Controller that is based on the OB-USP-Agent. (github.com)

[4] QACafe, Јул 2024, Доступно на: An overview of the User Services Platform (USP/TR-369) | qa | cafe (qacafe.com)

[5] Android Docs, Јун 2024, Доступно на: Android Mobile App Developer Tools – Android Developers

[6] SWIG, Јун 2024, Доступно на: Simplified Wrapper and Interface Generator (swig.org)