УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Вељко Михаиловић

Реализација окружења за динамичко превођење и извршавање TR-069 клијентске библиотеке за дигиталне телевизијске пријемнике са Linux ОС

МАСТЕР РАД

Нови Сад, 2014.

	УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА МАСТЕР РАД	Датум:

(Податке уноси предметни наставник - ментор)

СТУДИЈСКИ ПРОГРАМ:	Рачунарство и аутоматика
РУКОВОДИЛАЦ СТУДИЈСКОГ ПРОГРАМА:	Никола Јорговановић

Студент:	Вељко Михаиловић	Број индекса:	E2 41/2013
Област:	Пројектовање наменских рачунарских структура		
Ментор:	доц. др Милан Бјелица		

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

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

НАСЛОВ МАСТЕР РАДА:

Реализација окружења за динамичко превођење и извршавање TR-069 клијентске библиотеке за дигиталне телевизијске пријемнике са Linux ОС

ТЕКСТ ЗАДАТКА:

У оквиру задатка потребно је реализовати окружење које омогућава генерисање TP-069 клијентске библиотеке са апликативном програмском спрегом захваљујући којој је време интеграције библиотеке у циљни уређај смањено. Посебни аспекти које треба обухватити у оквиру рада су:
- преглед постојећих решења и анализа њихових проблема;
- предлог аутоматског генерисања и коришћења апликативне програмске спреге у фази превођења и покретања клијента;
- реализација окружења за динамичко превођење, као и одговарајућа адаптација кода реалне клијентске библиотеке;
- евалуација оптимизације интеграционог кода и времена развоја коришћењем развијеног решења, упоредном методом у односу на решење које не поседује реализована унапређења.

Руководилац студијског програма:	Ментор рада:

Примерак за: o - Студента; o - Ментора

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**:		Дипломски – мастер рад	
Аутор, **АУ**:		Вељко Михаиловић	
Ментор, **МН**:		Доц. Др Милан Бјелица	
Наслов рада, **НР**:		Реализација окружења за динамичко превођење и извршавање TR-069 клијентске библиотеке за дигиталне телевизијске пријемнике са Linux ОС	
Језик публикације, **ЈП**:		Српски / латиница	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2014	
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)		6/73/18/7/22/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кqучне речи, **ПО**:		TR-069, систем за превођење, прилагодљив , генератор кода, CWMP, дијагностика	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		У последњих неколико година сведоци смо све веће употребе TR-069 протокола који је дефинисао Broadband Forum. Многи уређаји већ имају интегрисану TR-069 клијентску библиотеку. Интеграција те библиотеке може бити тешка и може одузети пуно времена, због постојања великог броја параметара и објеката где сваки од њих захтева постојање одређеног броја функција да би могло да се манипулише њиховим вредностима. Овај рад предлаже решење при ком се те потребне функције генеришу приликом превођења TR-069 клијентске библиотеке.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	проф. др Никола Теслић	
	Члан:	доц. др Иштван Пап	
	Члан:	проф. др Бранислав Тодоровић	Потпис ментора
	Члан, ментор:	доц. др Милан Бјелица	

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**:		Master Thesis	
Author, **AU**:		Veljko Mihailović	
Mentor, **MN**:		Milan Bjelica, PhD	
Title, **TI**:		Adaptive build system for TR-069 consumer device library for Set-Top Boxes based on Linux OS	
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**:		2014	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		6/73/18/7/22/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S**/**KW**:		TR-069, build system, adaptive, code generator, CWMP, diagnostics	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		In the last few years, we have witnessed the expansion of Broadband Forum's CWMP (TR-069) protocol usage. Many devices today have integrated the TR-069 client library. Integration of that library can be difficult and time-consuming because of the many parameters and objects, where each of them requires an API function, to report its value. What we propose in this paper is the solution which includes API generation at the compile-time of the TR-069 client library.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Nikola Teslic, PhD	
	Member:	Branislav Todorovic. Phd	
	Member:	Istvan Papp, Phd	Menthor's sign
	Member, Mentor:	Milan Bjelica, PhD	

Zahvalnost

Posebno se zahvaljujem mentoru doc. dr Milanu Bjelici i doc. dr Ištvanu Papu na stručnoj pomoći, savetima i utrošenom vremenu.

Zahvaljujem se rukovodstvu instituta RT-RK na ukazanoj prilici da se bolje upoznam sa načinom rada u inženjerskom okruženju i budem uključen u proces razvoja novih programskih rešenja.

Posebnu zahvalnost ukazujem svojoj Milici, bez čijeg strpljenja i podrške bi izrada ovog rada mnogo duže trajala i sam rad bio manjeg kvaliteta.

Zahvaljujem se svojim roditeljima, Vlatku i Radi, sestri Bojani i baki Ruži na iskazanom strpljenju i poverenju.

Posebno se zahvaljujem svom pokojnom dedi Slobodanu, čiji su me saveti i poverenje učinili da budem kakav jesam i da stalno iz sebe izvlačim maksimum.

Na kraju zahvaljujem se svima koji su na bilo koji način doprineli izradi ovog završnog rada i mom školovanju.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

API - *Application Programming Interface*, Aplikativna programska sprega

DTT - *Digital Terrestrial Television*, Digitalna zemaljska televizija

IPTV - *Internet Protocol TV* IP televizija

EPG - *Electronic Programme Guide* Elektronski programski vodič

PIP - *Picture In Picture* Slika u slici

VOD - *Video On Demand* Video na zahtev

STB - *Set-Top Box* Digitalni televizijski prijemnik

OS - *Operating System* Operativni sistem

QoS - *Quality of Service* Kvalitet usluge (signala)

MAPI - *Data Model API* API modela podataka

CPE - *Customer Premises Equipment,* Krajnji korisnički uređaj

CWMP - *CPE Wide Area Network Management Protocol,* Mrežni protokol za upravljanje CPE

ACS - *Auto Configuration Server,* Auto konfiguracioni server

SOAP - *Simple Object Access Protocol,* Protokl za pristup jednostavnim objektima

HTTP - *HyperText Transfer Protocol,* Protokol za prenos hiperteksta

RPC - *Remote Procedure Call,* Poziv udaljene metode

DEVAL - *Device Abstraction Layer,* Sloj za apstrakciju uređaja

XML - *Extensble Markup Language,* Proširivi jezik za markiranje

1. Uvod

Digitalna televizija danas je uzela primat u većini zemalja u svetu pa i u Srbiji. Iako su razvijenije i tehnološki naprednije zemlje u potpunosti prešle na digitalnu televiziju, i ostale zemlje su u procesu zamene analognog signala digitalnim. U Srbiji je nedavno u rad puštena Digitalna Zemaljska Televizija (eng. DTT, *Digital Terrestrial Television*) koja pokriva oko 80% teritorije naše zemlje. Takođe, pružaoci usluga (provajderi) digitalne televizije, koji pružaju usluge kablovske, satelitske ili IP televizije (eng. IPTV- *Internet Protocol TV*), nastoje da svojim korisnicima pruže sve moderne usluge digitalne televizije poput: Elektronskog Programskog Vodiča (eng. EPG – *Electronic Program Guide*), Slika u slici (eng. PIP- *Picture In- Picture*), Web portal provajdera, Mozaik (prikaz vise od dva kanala odjednom u realnom vremenu), Video na zahtev (eng. VOD- *Video On Demand*), zatim aplikacije HbbTV, You Tube i druge. Usled sve veće zastupljenosti digitalne televizije, ubrzan je i razvoj uređaja koji mogu da prihvate, obrade i prikažu digitalni signal kao što su televizori, digitalni televizijski prijemnici (eng. STB - *Set-Top Box*), a takođe i ostali multimedijalni uređaji - pametni telefoni, tablet uređaji. Zajednička osobina ovih uređaja je da svaki ima pristup internet mreži, što stvara još više mogućnosti da se korisnicima pruži što raznovrsniji sadržaj i omogući što bolje iskustvo.

Bitna karakteristika digitalne televizije je kvalitet signala (eng. QoS – *Quality of Service*). Digitalna televizija je omogućila prenošenje stabilnog signala visoke rezolucije. Iako bolji, i digitalni signal je podložan raznim tipovima šuma. Kod IPTV-ja, zahtevi za kvalitetom signala su još izraženiji zbog mehanizma prenošenja signala. Kvalitet signala kod IP televizije zavisi od propusnosti mreže koju korisnik ima, mrežne opreme i smetnji u prenosu. Usled jako gustog saobraćaja na mreži, dolazi do gubitka na kvalitetu signala, gube se paketi, što kvari korisnički doživljaj. Menjanje kanala u uslovima gustog saobraćaja, biva produženo i do nekoliko sekundi. Takođe, mogući su kvarovi na digitalnim prijemnicima, kako na programskoj podršci tako i na fizičkim delovima uređaja. Stoga, provajderima je potrebna povratna informacija o dešavanjima

na uređajima kako bi mogli pravovremeno da reaguju na promene. Informacija o kvalitetu koju pruža korisnik prilikom anketiranja od strane provajdera često nije pouzdana s obzirom na subjektivnost; različiti korisnici dostavljaju različite povratne informacije o kvalitetu. Potreban je objektivniji pristup, gde provajderi dobijaju informacije o kvalitetu signala sa krajnjeg uređaja. Povratnu informaciju ka provajderu omogućavaju klijenti programske podrške na digitalnim prijemnicima koji su u mogućnosti da preuzmu podatke: o signalu koji digitalni televizijski prijemnik dobija, o statusu svake od komponenti digitalnog prijemnika i da te podatke prosledi poslužiocu (eng. *Server*), kao i da omogući oporavak od greške u toku rada ili ažuriranje programske podrške digitalnog prijemnika [4]. Ukoliko na uređaju postoji aktivna veza sa internetom, povratne informacije se šalju periodično; u suprotnom, podaci se čuvaju u memoriji uređaja sve dok se ne ostvari veza.

Jedan od klijenata koji omogućava navedene odlike je definisan prema standardizovanom TR-069 protokolu (eng. *Technical Report 069*) [1] koji garantuje siguran prenos. On definiše postojanje TR-069 poslužioca koji prikuplja podatke i TR-069 klijenta koji se nalazi na krajnjem korisničkom uređaju. Za svaki od uređaja postoji posebno definisan model podataka (eng. *Data Model*) koji jednoznačno, putem parametara, definiše uređaj na kom se TR-069 klijent nalazi. Za digitalne prijemnike, taj model podataka nosi naziv TR-135 (eng. *Technical Report 135*) [2]. Takođe, standard dozvoljava pravljenje sopstvenog modela podataka uz praćenje standardom precizno definisanih pravila [3].

Problem se javlja prilikom ugrađivanja TR-069 klijenta u korisnički uređaj. Svaki uređaj je specifičan i ima svoj model podataka, što znači da je za svaki uređaj potrebno uložiti napor da bi se klijentu dostavile fizičke vrednosti parametara na uređaju. Digitalni televizijski prijemnici se razlikuju prema tipu signala koji prihvataju, a samim tim, skup parametara koji ih opisuje su različiti. Dodatni problem je što u praksi svaki proizvođač definiše sopstvene parametre koji nisu deo standarda.. Na primer, TR-135 modelom podataka je definisano preko 300 parametara. Za svaki od ovih parametara potrebno je podržati dobavljanje i prosleđivanje korišćenjem Aplikativne programske sprege (eng. **API**- *Application Programming Interface*). Vreme potrebno za pisanje prilagodnih funkcija za preuzimanje ili postavljanje vrednosti parametara direktno je srazmerno veličini modela podataka. Veliki broj ovih funkcija sadrže sličan programski kod, međutim, aktivnost dodavanja ovih funkcija je manuelan i dugotrajan posao podložan greškama. Takođe, ukoliko dođe do izmene modela podataka u toku integracije, a to je vrlo čest slučaj, potrebno je uložiti dodatan napor da bi se kod ažurirao. Održavanje jednom napisanog koda je veoma teško.

U ovom radu, opisan je predlog rešenja navedenih problema korišćenjem Generatora API funkcija koji je ugrađen u sistem prevođenja TR-069 klijenta. Generator uzima model podataka i

obradom njegovog sadržaja uz pomoć šablona, na izlazu proizvodi API prilagođen modelu podataka (Data Model API - MAPI). Na ovaj način znatno se smanjuje posao onoga koji ugrađuje TR-069 klijenta u uređaj omogućavajući mu da se koncentriše na korišćenje ovih funkcija, umesto na njihovo pisanje. Praktični deo rada obuhvata pravljenje Generatora API funkcija i njegovo ugrađivanje u sistem prevođenja TR-069 klijenta.

Rad je organizovan u nekoliko celina:

- **Teorijske osnove** – opis komunikacionog protokola, opis modela podataka i pravila, opis platforme za testiranje, postojeća rešenja;
- **Koncept rešenja** – opsi funkcionisanja predloženog sistema za dinamičko prevođenje i izvšavanje TR-069 klijentske biblioteke;
- **Programsko rešenje** – opis klasa, metoda i interfejsa koji čine Generator API-ja, opis izmena u sistemu prevođenja TR-069 klijentske biblioteke;
- **Ispitivanje i rezultati** – opis ispitnih slučajeva i rezultata predloženog rešenja;
- **Zaključak** – ispunjenost predloženog rešenja, prednosti, budući rad.

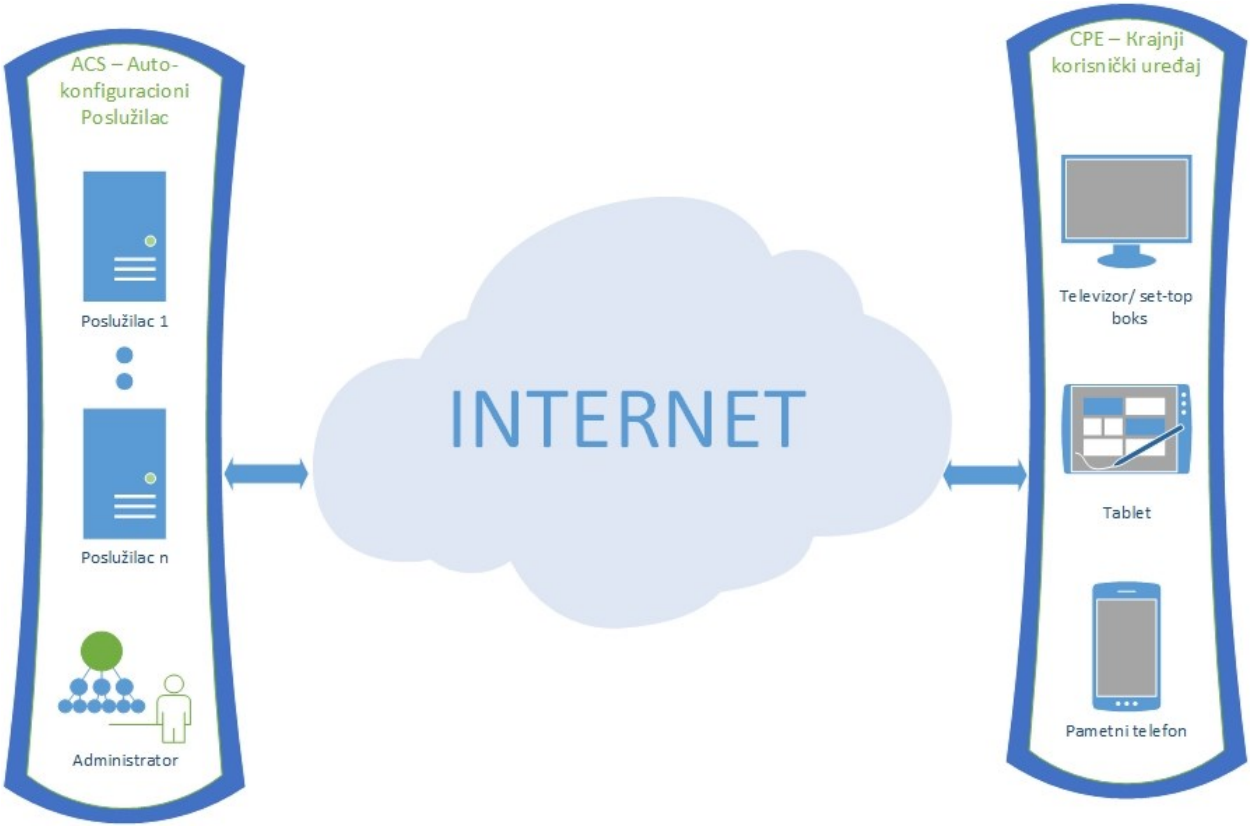
2. Teorijske osnove

U ovom poglavlju date su teorijske osnove na kojima počiva predloženo rešenje. Opisana je struktura TR-069 klijentske biblioteke nad kojom je realizovano predloženo rešenje, kao i bitni koncepti potrebni za realizaciju rešenja. Objašnjeni su odnosi između uređaja i klijentske biblioteke i gde se MAPI nalazi u strukturi TR-069 klijentske biblioteke.

2.1 Komunikacioni protokol TR-069

TR-069 je napravljen od strane Širokopojasnog foruma (eng. *Broadband forum*) [1] i nazvan je CPE WAN upravljački protokol (eng. CWMP, *Customer Premises Equipment Wide Area Network Management Protocol*). Ovaj protokol se nalazi na ISO OSI aplikativnom nivou, i njegova funkcionalnost je da obavlja upravljanje i nadzor nad uređajima koji se nalaze kod krajnjih korisnika. CWMP definiše dva tipa uređaja: krajnji korisnički uređaj (eng.CPE, *Customer Premises Equipment*) i automatsko-konfiguracioni poslužilac (eng. ACS, *Auto-Configuration Server*). CPE može biti bilo koji korisnički uređaj kao što su: televizor, tablet, set-top-boks, pametni telefon, usmerivač (eng. *Router*). Komunikacija je dvosmerna, zasnovana na SOAP/HTTP(S) protokolima. Ceo sistem je osmišljen da bi se razni, gore pomenuti uređaji, mogli konfigurisati za pristup internetu, kao i za rad za koji su namenjeni, a isto tako i da bi se omogućilo lakše praćenje otkaza u radu uređaja. Prvi put, ovaj protokol je objavljen u maju 2004. godine. Do danas, objavljeno je 5 revizija (eng. *amendment*). Trenutna revizija protokola je 1.4. Opšti izgled celog sistema prikazan je na Slici 2.1.

Prednost TR-069 protokola nad nekim od ranijih rešenja je što se ne zahteva stalna veza između CPE i ACS. Strana koja započinje komunikacionu sesiju je CPE. Pakovanje tih poruka se odvija na SOAP nivou i obrađuje se na svim nivoima protokol steka. U Tabeli 2.1 prikazani su zastupljeni protokoli po nivoima, koji su definisani standardom.



Slika 2.1 Opšti prikaz funkcionisanja sistema zasnovanog na TR-069 protokolu

Upravljačke poruke korišćene u TR-069 protokolu su definisane kao RPC (eng. *Remote Procedure Call*) metode, čime je omogućena jednostavna proširivost standarda. Ukoliko je potrebno dodavanje nove upravljačke poruke, potrebno je definisati novu RPC metodu, način rukovanja i njenu sintaksu na SOAP nivou, bez potrebe za menjanjem nižih nivoa protokol steka.

Nivo protokola	Protokol
6	CPE/ACS aplikativni protokol
5	RPC
4	SOAP
3	HTTP(S)
2	SSL/TLS
1	TCP/IP

Tabela 2.1 Prikaz korišćenih protokola po nivoima protokol steka

U Tabeli 2.2 su prikazane neke od obaveznih/opcionih RPC metoda koje bi svaki sistem koji želi da implementira TR-069 protokol, trebalo da podrži.

RPC metode	Revizija 1	Revizija 2	Revizija 3	Revizija 4	Revizija 5
GetRPCMethods	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
SetParameterValues	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
GetParameterValues	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
GetParameterNames	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
SetParameterAttributes	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
GetParameterAttributes	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
AddObject	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
DeleteObject	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
Download	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
Reboot	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
Inform	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
TransferComplete	Obavezna	Obavezna	Obavezna	Obavezna	Obavezna
Upload	Opciona	Opciona	Opciona	Opciona	Opciona
FactoryReset	Opciona	Opciona	Opciona	Opciona	Opciona
GetQueuedTransfers	Opciona	Opciona	Zastarela	Zastarela	Zastarela
ScheduleInform	Opciona	Opciona	Opciona	Opciona	Opciona
SetVouchers	Opciona	Opciona	Zastarela	Zastarela	Zastarela
GetOptions	Opciona	Opciona	Zastarela	Zastarela	Zastarela
GetRPCMethods(ACS)	Opciona	Opciona	Opciona	Opciona	Opciona
RequestDownload	Opciona	Opciona	Opciona	Opciona	Opciona
Kicked	Opciona	Opciona	Zastarela	Zastarela	Zastarela
GetAllQueuedTransfers		Opciona	Opciona	Opciona	Opciona
AutonomousTransferComplete		Opciona	Opciona	Opciona	Opciona
DUStateChangeComplete			Opciona	Opciona	Opciona
AutonomousDUStateChangeComplete			Opciona	Opciona	Opciona
CancelTransfer			Opciona	Opciona	Opciona
ScheduleDownload			Opciona	Opciona	Opciona
ChangeDUState			Opciona	Opciona	Opciona

Tabela 2.2 Prikaz RPC metoda prema profilima TR-069 protokola [1]

Poželjno je da i CPE-ovi i ACS pored najnovije revizije standarda podrže i stare revizije, zbog kompatibilnosti sa uređajima starije generacije, koji podržavaju na primer RPC metode iz revizije 1, TR-069 protokola.

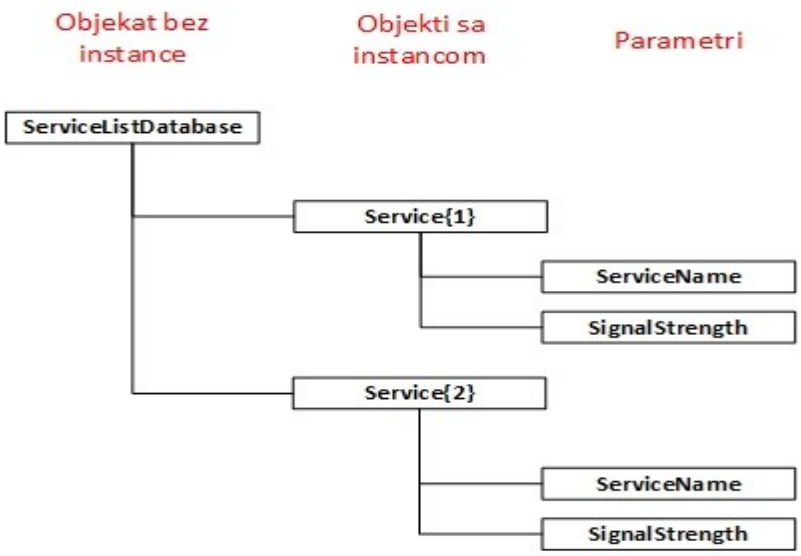
2.1.1 Modeli podataka

Karakteristika ovog protokola je da poseduje mehanizam, koji je podržan protokolom, a koji omogućava prilagođavanje protokola u zavisnosti od prirode i tipa uređaja. TR-069 koristi koncept modela podataka (eng. *Data Model*), koji predstavljaju apstrakciju fizičkog uređaja. Model podataka standardizuje relevantne parametre i objekte kojima poslužilac i uređaj u koji je integrisan TR-069 klijent mogu pristupiti. Standardizovani modeli podataka postoje za razne mrežne uređaje, baze podataka, mobilne ili set-top-boks uređaje i usko su vezani za protokol. Kako bi protokol ostao fleksibilan, omogućena je personalizacija modela podataka putem profila (eng. *Profiles*) [3] uz mogućnost definisanja sopstvenog (specifičnog za proizvođača) profila. U mnogim slučajevima je ovo korisna opcija, jer svaki uređaj pored standardnih ima i neke posebne osobine. Sem standardizovanja tipa informacija, model podataka definiše tip svakog parametra, zajedno sa dozvolama čitanja i pisanja. Model podataka koji je prilagođen set-top-boks uređajima naziva se TR-135 [2].TR-106 [3] je osnovni model za TR-069 protokol, bez koga se ne može ostvariti veza i potreban je svakom uređaju. TR-106 omogućuje praćenje verzija programske podrške koja se nalazi na uređaju i kontrolu nad vremenskim razmakom između dve komunikacione sesije. Takođe TR-106 definiše pravila prema kojima se modeluju uređaji, tj. pravila koja treba da se prate prilikom pravljenja novog modela podataka. Definisanjem sopstvenog profila omogućuje se korišćenje podskupa parametara i objekata.

Modeli podataka se sastoje iz velikog broja parametara i objekata, gde svaki od parametara odgovara podatku koji se može čitati i/ili pisati. Tipovi parametara koji su definisani TR-106 modelom podataka su:

- Niz karaktera (eng. String)
- Celobrojni (eng. Integer)
- Celobrojni prošireni (eng. Long)
- Neoznačeni celobrojni (eng. Unsigned Integer)
- Neoznačeni prošireni celobrojni (eng. Unsigned Long)
- *Boolean*
- Vremenski tip (eng. dateTime)
- Binarni tip (eng. Base64)
- Heksadecimalni tip (eng. hexBinary)

Postoje dve vrste objekata: *objekti bez instanci* i *objekti sa jednom ili više instanci*. Objekti bez instanci služe za klasifikaciju više srodnih parametara ili obe vrste objekata, dok objekti sa više instanci dodatno omogućavaju kreiranje više kopija objekata iste strukture (naprimer: Service{1}, Service{2}). Izgled parametara i objekata prikazan je na Slici .2.2



Slika 2.2 Izgled parametara i objekata u modelu podataka

Da bi se TR-069 klijent integrisao u uređaj, potrebno je napraviti API, koji podržava sve parametre i objekte sa više instanci. Maksimalni skup funkcija za parametre su funkcije: *Postavi*, *Dobavi*, *Prijavi* i *Odjavi*, dok je maksimalni skup funkcija za objekte: *Dodaj*, *Ukloni*, *Prijavi* i *Odjavi*.

Tip funkcije *Postavi* vrši postavljanje odgovarajućeg parametra u stablu podataka, koje odgovara modelu podataka za taj uređaj, na prosleđenu vrednost.

Tip funkcije *Dobavi* uzima vrednost iz stabla i dobavlja je pozivaocu.

Tip funkcije *Dodaj* za objekte vrši dodavanje određenog objekta u stablo i vraća instancu objekta.

Tip funkcije *Ukloni* uklanja određenu instancu objekta iz stabla.

Tip funkcije *Prijavi* omogućuje programskoj podršci, u koju je ugrađen TR-069 kljent, da izvrši komande koje zada ACS putem povratne funkcije (eng. *callback*). Konkretno, kod digitalnih prijemnika, ACS može da zatraži promenu kanala na uređaju. U stablu se promeni vrednost parametra za trenutnu kanal na vrednost dobijenu od ACS i TR-069 klijent obavesti programsku podršku.

Tip funkcije *Odjavi* uklanja prijavljenu povratnu funkciju iz TR-069 klijenta.

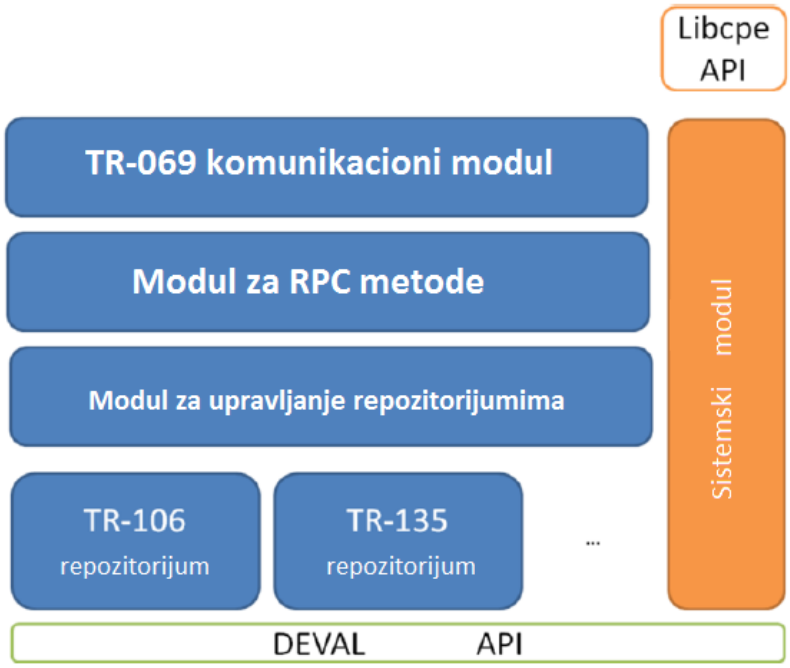
Prijavi i *Odjavi* imaju istu ulogu i za parametre i za objekte.

Ukoliko ACS ne može da promeni vrednost parametra, doda ili obriše objekat sa više instanci onda funkcije *Prijavi* i *Odjavi* nisu potrebne.

2.2 Struktura TR-069 klijenta

TR-069 klijent koji je korišćen u razvoju dinamičkog sistema za prevođenje TR-069 klijentske biblioteke razvijan je na institutu za Računarsku tehniku i računarske komunikacije.

Njegova struktura i moduli od kojih je sačinjen prikazani su na Slici 2.3. Podržano je pokretanje više instanci biblioteke na jednom uređaju, što omogućava rad sa više korisnika istovremeno (eng. *Multiclient*).



Slika 2.3 Moduli u TR-069 klijentskoj biblioteci

2.2.1 Sistemski modul

Sistemski modul pruža prilagodni sloj koji pušta u rad, a i zaustavlja TR-069 klijentske biblioteke. Isto tako, u okviru njega se obavlja zauzimanje resursa, učitavaju modeli podataka i uspostavlja se veza (konekcija) sa poslužiocem. On predstavlja osnovni modul u CPE klijentu i omogućava dostupnost funkcionalnosti biblioteke drugim aplikacijama.

2.2.2 TR-069 komunikacioni modul

TR-069 komunikacioni modul je zadužen za komunikaciju između poslužioca i uređaja. U sklopu modula su funkcionalnosti:

- Čekanje i obrada događaja koji će pokrenuti TR-069 sesiju [1];
- Izvođenje TR-069 sesije;
- Izvođenje prenosa datoteka;
- Omogućavanje sprege za rukovanje programskom podrškom krajnjeg uređaja;
(eng. Firmware upgrade) [4];
- Parsiranje HTTP zaglavlja i proveru ispravnosti;
- Pakovanje, validacija i parsiranje SOAP struktura;
- Pozivanje odgovarajuće RPC metode;

- Slanje i prijem HTTP poruka.

Prilikom prijema svakog HTTP paketa vrši se parsiranje HTTP zaglavlja, a potom provera ispravnosti SOAP paketa. Ukoliko je primljen, SOAP zahtev određuje koji tip RPC metode se poziva i vrši se parsiranje SOAP paketa i poziv RPC metode sa dobijenim podacima. Ukoliko dobije zahtev za vrednosti nekog od parametara ili objekata, pristupa Modulu za rukovanje repozitorijumima preko RPC metoda i vraća ih ACS poslužiocu.

2.2.3 Modul za upravljanje repozitorijumima

Modul za rukovanje repozitorijumima (eng. *Data Repository Engine*) obezbeđuje CRUD (eng. *Create, Read, Update, Delete*) funkcionalnosti nad repozitorijumima definisanim na osnovu TR-069 podržanih modela podataka. Uređenje struktura u ovom modulu omogućuje lako proširivanje novim modelima podataka. Svaki model podataka se prilikom inicijalizacije biblioteke instancira kao stablo i poveže u listu repozitorijuma, tako da se dodavanje novih repozitorijuma može izvršiti bez izmena u kodu biblioteke. Da bi se pristupilo parametru ili objektu u repozitorijumu, potrebno je obezbediti putanju do parametra/objekta definisanog u modelu podataka i instancu napravljenog repozitorijuma. Ovaj modul obezbeđuje osnovu na koju se naslanjaju API funkcije *Postavi*, *Dobavi*, *Dodaj*, *Ukloni*, *Prijavi* i *s*.

2.2.4 Ostali moduli

Moduli TR-106 repozitorijum i TR-135 repozitorijum predstavljaju prilagodni API konkretnih krajnjih uređaja prema biblioteci. Zasnovani su na odgovarajućim modelima podataka. Ovi moduli predstavljaju funkcije koje su potrebne za integraciju TR-069 klijenta u uređaj. U njima treba da se nalaze minimalni skupovi funkcija za svaki parametar i za svaki objekat sa više instanci.

Libcpe API i DEVAL(eng *Device Abstraction Layer*) API su „.h“ fajlovi koji se isporučuju korisnicima biblioteke i sadrže kompletan skup funkcija kojima se može iskoristiti potpuna funkcionalnost TR-069 klijentske biblioteke. DEVAL API je unija funkcija za sve repozitorijume koji postoje na konkretnom uređaju, odnosno za sve modele koji su potrebni za modelovanje uređaja u koji se TR-069 klijentska biblioteka ugrađuje. Libcpe API je skup funkcija za pokretanje i zaustavljanje rada klijentske biblioteke. DEVAL API predstavlja MAPI.

2.3 Sistemi za prevođenje

Sistemi prevođenja (eng. *Build System*) igraju bitnu ulogu u bilo kojem programskom projektu. Jako je bitno imati podešeno i dobro okruženje za sistem za prevođenje koda, jer to u velikoj meri olakšava održavanje programskog rešenja. Na samom početku potrebno je izabrati

odgovarajuće rešenje za sistem prevođenja. Za Linux OS postoje razna rešenja od kojih su samo neka:

- GNU Make [16];
- CMAKE [17];
- QMAKE [18].

2.3.1 GNU Make

GNU Make je najstariji sistem za prevođenje za Linux OS. Sastoji se iz grupe alata koji se zovu *Autotools* i služe za prevođenje izvršnog koda na ciljnu platformu. Sastoji se iz tri alata:

- Autoconf – alat za podešavanje okruženja za odgovarajuću platformu i generisanje *Makefile* datoteka;
- Automake – alat koji pomaže pri pravljenju prenosivih *Makefile* datoteka;
- Gnu libtool – biblioteka koja definiše pravljenje statičkih i dinamičkih biblioteka na Unix sistemima.

Makefile su datoteke za prevođenje izvornog koda, koje sadrže opis i instrukcije koje program *make* treba da izvrši. U njima se nalazi tačan opis prevođenja izvornog koda projekta, prevodioci koji se koriste itd. *Make* prevede *Makefile* datoteku/e i krene da izvršava instrukcije. Ceo sistem je zasnovan na četiri koraka: 1) definisanje cilja (eng. *target*), 2) određivanje zavisnosti cilja (eng. *dependencies*) 3) naredbe koje se izvršavaju (eng. *commands*) i 4) varijable okruženja (eng. *environment variables*).

Definisanje cilja podrazumeva šta sve *make* treba da izvrši da bi napravio cilj. Na primer, cilj može da predstavlja biblioteku koja se dobija prevođenjem izvršnog koda.

Određivanje zavisnosti cilja podrazumeva da se za cilj navedu datoteke ili drugi ciljevi od kojih zavisi cilj. U slučaju biblioteke to je izvorni kod.

Nakon zavisnosti, za cilj je potrebno navesti komande koje treba da se izvrše, da bi cilj bio napravljen.

Varijable okruženja omogućuju Makefile datoteci da bude konfigurabilna i prenosiva sa platforme na platformu. Cilj, zavisnosti i komande je moguće definisati preko varijabli.

Pisanje *Makefile* datoteka, može biti naporno i komplikovano, što često dovodi do grešaka koje onemogućavaju uspešan prevod izvornog koda.

2.3.2 CMAKE

CMAKE je alat koji automatski pravi datoteke za prevođenje za bilo koji operativni sistem (*Unix, Mac OS, Windows*). Njegov zadatak je da napravi datoteke za prevođenje (*Makefile* na Linux OS). U datotekama koje se zovu *CMakeLists* isto kao i u Makefile datotekama, definišu se

ciljevi, zavisnosti, varijable i naredbe. Ono što CMAKE nudi je jednostavnije definisanje ovih osobina putem direktiva definisanih u CMAKE. Lakše se otkrivaju greške u definisanju sistema prevođenja za projekat, nego što je to slučaj sa *Makefile* – ovima. Manje je komplikovan, jednostavno se manipuliše direktorijumima u kojima se nalazi izvorni kod i jednostavan je za održavanje.

Za svaki direktorijum koji sadrži deo izvornog koda projekta, potrebno je dodati *CMakeLists* datoteku, da bi se kod uspešno povezaou i podržale prave zavisnosti.

Prevođenje izvornog koda, korišćenjem CMAKE alata se izvodi u dva koraka.

- Korišćenjem alata *cmake* učita se korenska *CMakeLists* datoteka i sve datoteke u ostalim direktorijumima, i od njih se automatski naprave *Makefile* datoteke;
- Alat *make* se izvrši nad napravljenim *Makefile* datotekama postupkom opisanim u poglavlju 2.3.1.

Kao sistem za prevođenje biblioteka opisana u 2.2 koristi upravo CMAKE zbog svoje jednostavnosti i prenosivosti.

2.3.3 QMAKE

QMAKE je alat istovetan CMAKE alatu. Kao i CMAKE, QMAKE pravi datoteteke za prevođenje shodno operativnom sistemu na kojem se koristi. Nastao je zajedno sa razvojem Qt (eng. *cute*) biblioteka. Široko je rasprostranjen u zajednici, koristi se na primer u WebKit projektu (QTWebKit). Principi na kojima počiava QMAKE su identični onima koje koristi CMAKE tako da je proces prevođenja izvorni datoteka istovetan CMAKE postupku.

2.4 Dokumentacija

Svaki API uz sebe ima i prigodnu dokumentaciju koja omogućuje korisniku API-ja:

- Jednostavno korišćenje API funkcija;
- Jednostavnu pretragu za željenom APU funkcijom;
- Opis funkcionalnosti svake funkcije u API-ju.

Postoje alati (kao što je *Doxygen*) koji generišu dokumentacije za API pomoću komentara koji se nalaze uz API funkciju u datoteci. Ti komentari su formatirani na poseban način, tako da *Doxygen* prolaskom kroz API funkcije u datoteci, može da izdvoji komentare za dokumentaciju. *Doxygen* definiše direktive koje mu pomažu u generisanju dokumentacije iz komentara. Neke od njih su:

- *@fn* – direktiva iza koje se nalazi ime API funkcije;
- *@param* – direktiva iza koje se navodi ime argumenta funkcije i njegova svrha;

- *@bref* – direktiva iza koje se nalazi kratak opis funkcionalnosti API funkcije;
- *@detail* – direktiva iza koje se nalazi duži opis funkcionalnosti API funkcije.

Za više detalja o direktivama i načinu korišćenja pogledati [14]. Primer komentara API funkcije formatiranog za dokumentaciju, prikazan je na Slici 2.4.

```
/**
    @fn ime API funkcije sa argumentima
    @brief kratak opis šta zapravo funkcija radi, šta predstavlja
    ulazna/izlazna vrednost
    @param ime opis argumenta funkcije, može ih biti više ili nijedan.

*/
```

Slika 2.4 Primer komentara API funkcije formatiraong prema *Doxygen* pravilima

2.5 Opis problema

Postojeći problemi sa današnjim rešenjima za TR-069 klijentsku biblioteku vezani su za ugrađivanje ove biblioteke u razne uređaje, prvenstveno zbog potrebnog vremena da se obezbedi odgovarajući API, kao i grešaka koje nastaju tokom pisanja koda. U Poglavlju 2.1.1, opisani su modeli podataka i njihov sadržaj. Modeli podataka su u direktnoj vezi sa brojem API funkcija koje je potrebno podržati, da bi se mogla izvršiti integracija TR-069 klijentske biblioteke u ciljni uređaj. Pošto za svaki parametar postoji maksimalni skup od četiri funkcije, a u modelu podataka za digitalne televizijske prijemnike TR-135 definisano je preko 300 parametara, dolazi se do zaključka da je potrebno podržati 1200 funkcija u slučaju da su za svaki parametar potrebne četiri funkcije. Ovaj podatak važi ako u modelu ne postoje objekti sa više instanci, ali ih u TR-135 modelu ima. Koliko god da je telo funkcija malo, pisanje ovolikog broja funkcija traje jako dugo i podložno je greškama. Isto tako, dodatno vreme se troši prilikom iščitavanja iz modela podataka kog tipa je parametar za koji se funkcija piše, zatim putanja do parametra u stablu.

2.6 Postojeća rešenja

Tokom istraživanja za potrebe ovog rada, nisu pronađena slična rešenja ili istraživanja na temu koja je pokrivena ovim radom. Neka od postojećih rešenja TR-069 klijenata su:

- TR-069 klijent razvijen od strane kompanije AVSystems[10];
- FreeCWMP [9] rešenje koje je napravljeno od strane zajednice otvorenog koda.

U opisu svog rešenja, AVSystems navodi da je integracija njihovog klijenta olakšana i jednostavna. Nakon uvida u njihovu strukturu, ustanovljeno je da ne poseduju mehanizam za automatsko generisanje MAPI-ja.

Održavaoci FreeCWMP projekta ne omogućuju MAPI, već naglašavaju da je njihovo rešenje lako proširivo za bilo koji CWMP model podataka. Nakon uvida u strukturu i organizaciju njihovog rešenja, ustanovljeno je da ne poseduju rešenje za brzu i jednostavnu podršku MAPI-ja.

Na osnovu istraženih postojećih rešenja, izvodi se zaključak da je rad na Generatoru API-ja opravdan.

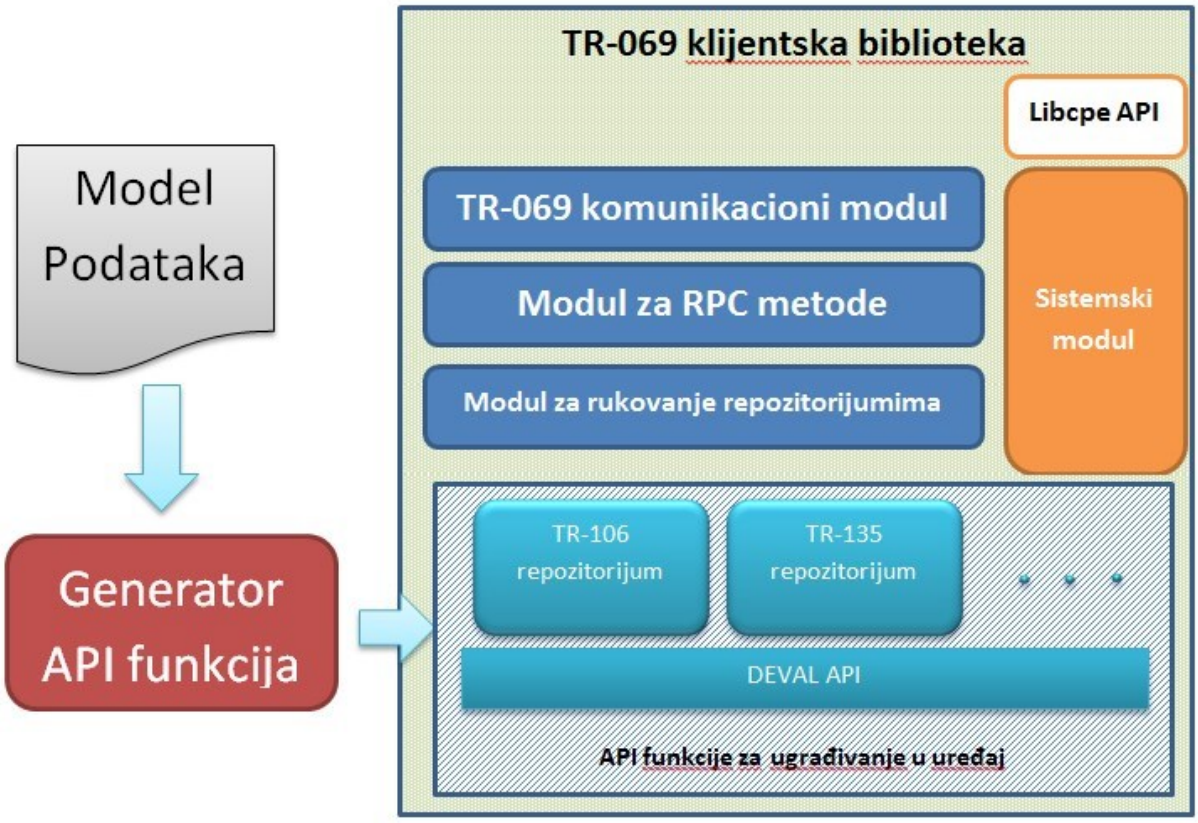
U radu su korišćeni važni koncepti iz prakse: adaptivni sistemi prevođenja (eng. *Build System*), automatski generisani kod i korišćeni su postojeći sistemi za prevođenje u Linux OS.

U svom radu B. Adams [6] opisuje vezu između izvornog koda (eng. *Source Code*) i sistema prevođenja, ističući da je potrebno ažurirati i održavati sistem za prevođenje zajedno sa izvornim kodom. Rast izvornog koda, podstiče rast i razvoj sistema za prevođenje. Prema njemu, ova dva sistema zajedno evoluiraju.

M. De Jonge ukazuje da dobar, robusan i dobro modularizovan sistem za prevođenje omogućuje ljudima koji razvijaju programsku podršku da iskoriste već napisan izvorni kod [7]. Još jedan, jako bitan koncept je bio potreban za izradu ovog rešenja, Automatsko generisanje izvornog koda. A. Acuri i X. Yao su u svom radu [8] opisali način za evoluciju automatski generisanog koda zajedno sa programskom podrškom čiji je deo. Ovaj koncept datira iz 70-ih i primenjen je prilikom rešavanja predočenih problema.

3. Koncept rešenja

Predloženo rešenje za dinamičko prevođenje TR-069 klijentske biblioteke podrazumeva dodavanje dodatnog spoljašnjeg modula u hijerarhiju TR-069 klijentske biblioteke. Dodatni modul je moguće dodati na bilo koju realizaciju TR-069 klijentske biblioteke, a na Slici 3.1 prikaz TR-069 klijentske biblioteke opisane u Poglavlju 2.2 sa Generatorom API-ja.

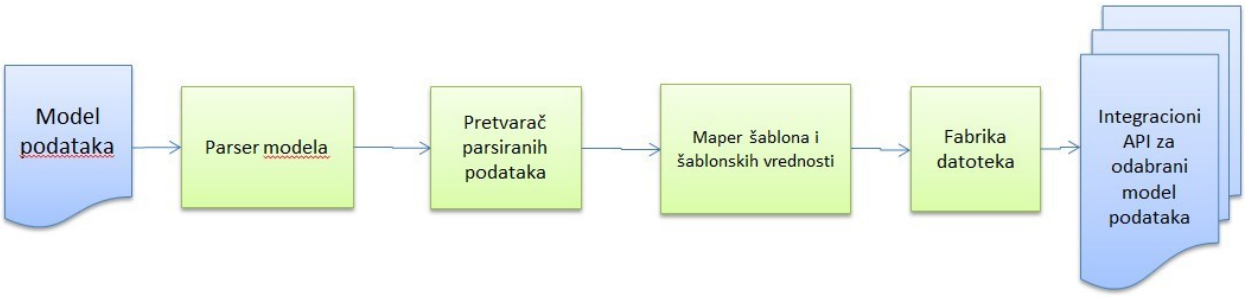


Slika 3.1 Struktura TR-069 klijentske biblioteke opisane u Poglavlju 2.2, sa Generatorom Api funkcija

Predloženo rešenje je univerzalno za bilo koju implementaciju TR-069 klijentske biblioteke. Generator nije deo biblioteke, ali izlaz iz njega jeste i koristi pri njenom prevođenju.

3.1 Generator API funkcija

Generator API funkcija (Generator API-ja) predstavlja alat koji dopunjuje TR-069 klijentsku biblioteku. Na ulazu, Generator API-ja očekuje model podataka koji odgovara uređaju za koji je potreban MAPI. Na izlazu daje izvorni kod koji treba da se integriše u sistem prevođenja TR-069 klijentske biblioteke, kao što je prikazano na Slici 3.1. Struktura ovog alata je takođe modularna i njegova struktura je prikazana na Slici 3.2.



Slika 3.2 Struktura Generatora API-ja

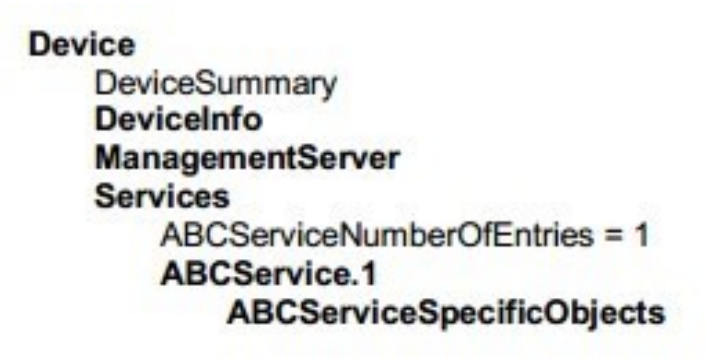
Na slici je prikazan i tok obrade koji prolazi prosleđeni Model podataka do MAPI-ja.

3.1.1 Parser modela

Kao ulaz prihvatljiv je bilo koji Model podataka definisan od strane „Broadband foruma“ ili model napravljen prateći pravila koja su definisana sa TR-106. Zbog toga Parser modela je napravljen tako da zadovolji sledeća pravila (za detalje pogledati [3]):

- Strukturni zahtevi za hijerarhiju podataka;
- Zahtevi za verzionisanje modela podataka;
- Zahtevi za definisanje profila;
- Skup zajedničkih objekata;
- Bazni profil za bilo koji uređaj koji podržava skup zajedničkih objekata.

Strukturni zahtevi se odnose na definisanje oznaka za objekte, definisanje objekata bez instance i objekata sa više instanci, dodavanje komponenti u modelu, koji objekti su obavezni a koji nisu. Na Slici 3.3 je prikazan primer strukturne hijerarhije. Iz ovoga se jasno vidi da je svaki objekat/parametar jednoznačno određen putanjom.



Slika 3.3 Primer strukturne hijerarhije u modelima podataka [3]

Verzionisanje modela podataka je način da se model proširi, da se dodaju novi objekti i parametri, ali i da se omogući podrška za starije revizije (na primer TR-106-1-0 i TR-106-1-1). Prvi broj označava veću reviziju, a drugi manju. Ukoliko je model podataka koji je prosleđen Parseru modela, zavisan od nekog drugog modela ili starije revizije, taj model ili revizija se uključuju i parser ih spaja u jedan po potrebi.

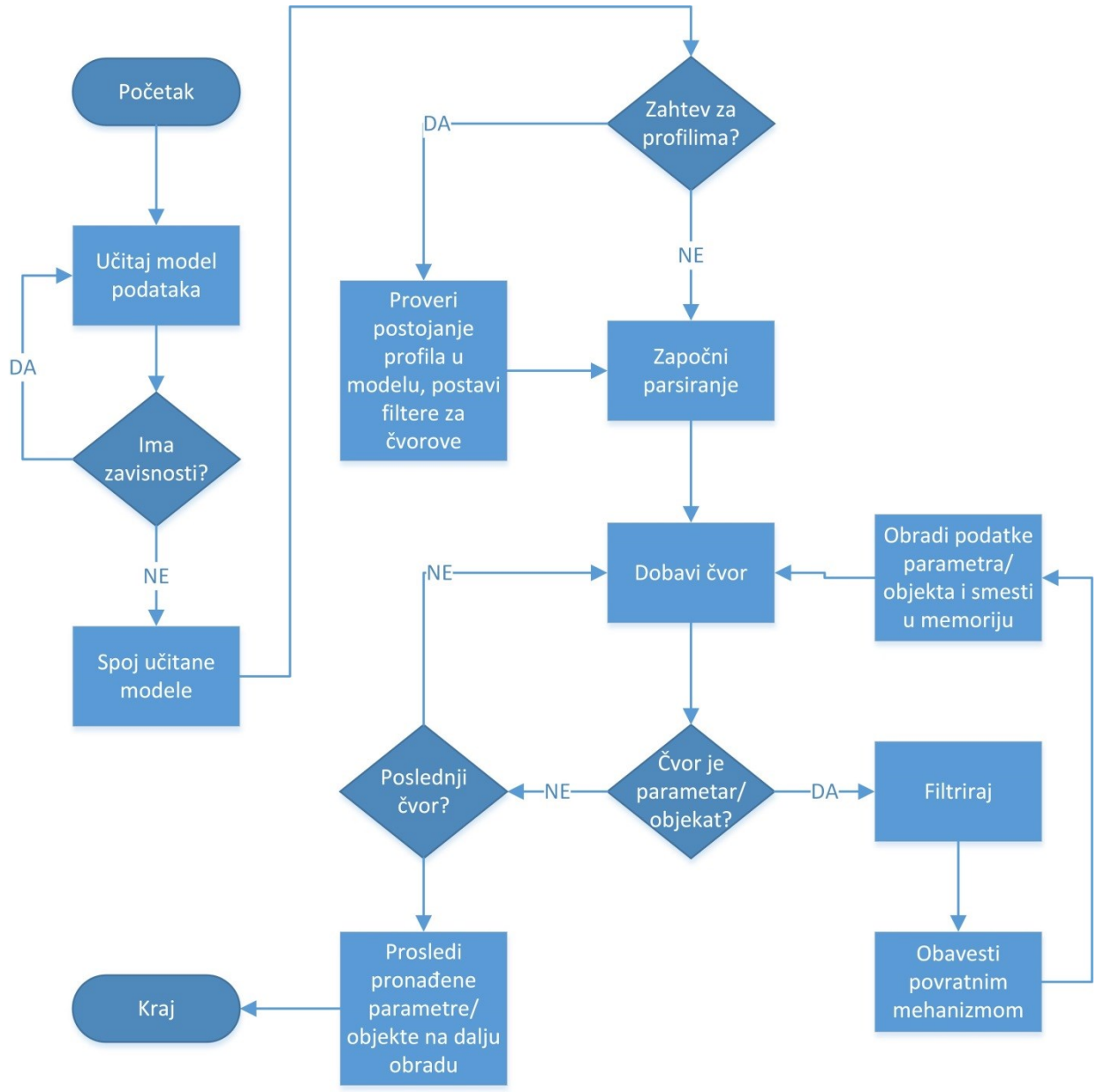
Definisanje profila podrazumeva određivanje podskupa objekata i parametara iz modela podataka. Ukoliko Parser modela dobije listu profila, on pokušava da pretraži model za tim profilima i isparsira objekte i parametre koji potpadaju pod njih. Ukoliko ne pronađe nijedan od zadatih profila, Parser modela obradi sve objekte i parametre iz modela podataka.

Bazni profil obuhvata osnovne objekte i podskup njihovih objekata/parametara. Ti objekti su, na primer, Podaci o uređaju (eng. *DeviceInfo*), Karakteristike uređaja (eng. *Capabilities*).

Koristeći neki od već postojećih parsera za „.xml“ datoteke (eng XML- *Extensive Markup Language*), vrši se obrada svakog čvora u modelu podataka. Čvor može biti *parametar*, *objekat bez instance* ili *objekat sa više instanci*. Podaci relevantni za Generator API-ja kao što su putanja do parametra/objekta, tip parametra, ograničenja parametra/objekta, opis parametra/objekta se čuvaju u kontejnerima i prosleđuju dalje na obradu. Pronađeni parametri/objekti sa podacima se prosleđuju na dalju obradu putem povratnog mehanizma (eng. *callback*). Svaki obrađeni parametar/objekat se smešta u memoriju i proces se nastavlja dok se parsiranje modela podataka ne završi.

Takođe, na osnovu prikupljenih podataka, ovaj modul odlučuje za svaki parametar/objekat da li je potrebno podržati funkcije *Prijavi* i *Odjavi*. Ove funkcije nisu potrebne ukoliko ACS poslužilac nema prava za pisanje.

Kada završi svoju obradu, Parser modela prosledi sakupljene podatke sledećem modulu u hijerarhiji Generatora API-ja. Algoritam procesa parsiranja čvorova u modelu podataka prikazan je na Slici 3.4.



Slika 3.4 Algoritam Parser modula

3.1.2 Pretvarač parsiranih podataka

Zadatak Pretvarača je da prihvati podatke iz Parsera i prilagodi ih vrednostima koje se mogu smestiti u šablone. Karakteristika ovog modula je što nije vezan za konkretan programski jezik za koji se MAPI pravi. Zbog toga se ovo rešenje za generisanje API-ja može primeniti na bilo koju implementaciju TR-069 klijentske biblioteke.

Za svaki parametar i objekat, ovaj modul proizvodi podatke za šablone. Svaka od funkcija za MAPI je specifična i zahteva sebi svojstvene podatke: ime, povratna vrednost, ulazna vrednost itd.

Prilikom pravljenja imena za funkciju, bilo za parametre ili objekte sa više instanci, potrebno je obezbediti da ime bude jedinstveno u celom MAPI-ju. Svaki tip funkcija mora sadržati ime modela podataka i mora sadržati tip funkcije.

Tip parametra određuje koji tip argumenta funkcija treba da primi kao ulazni (*Postavi*) ili izlazni (*Dobavi*). Funkcije mogu imati i dodatni broj argumenata, što direktno zavisi od broja objekata sa više instanci koji se nalaze na putanji parametra u stablu. Pošto objekti nemaju tip, funkcije *Dodaj* /*Ukloni* kao izlazni/ulazni parametar primaju samo broj instance. Pored ovog argumenta mogu imati i druge koji, kao i kod parametara, direktno zavise od broja objekata sa više instanci koje imaju na putanji.

Skup potrebnih vrednosti podataka za šablone se određuje na osnovu konkretne TR-069 klijentske biblioteke i primera MAPI-ja uz tu biblioteku, tako da je ovaj deo prilagodljiv tim potrebama. Primer MAPI-ja služi tome da bi se mogao izdvojiti nepromenljivi i promenljivi deo API funkcija. Na primer, sve što je u API-ju direktno vezano za parametar, potrebno je napraviti šablonsku vrednost, a nepromenljivi deo se koristi za definisanje šablona (eng. *template*).

3.1.3 Maper šablona i šablonskih vrednosti

Ovaj modul sadrži šablone, i šablonske vrednosti. Prilagodljiv je programskom jeziku dok je šablone jednostavno menjati. Ideja je da se sleđećem modulu ponudi potreban i dovoljan skup funkcionalnosti, kojima taj modul može da proizvodi datoteke sa izvornim kodom API funkcija. Prilagodljivost omogućava da sledeći modul ne mora da poznaje specifičnosti MAPI datoteka koje pravi, kao što je programski jezik za koji se MAPI pravi, način povezivanja itd.

3.1.3.1 Šabloni

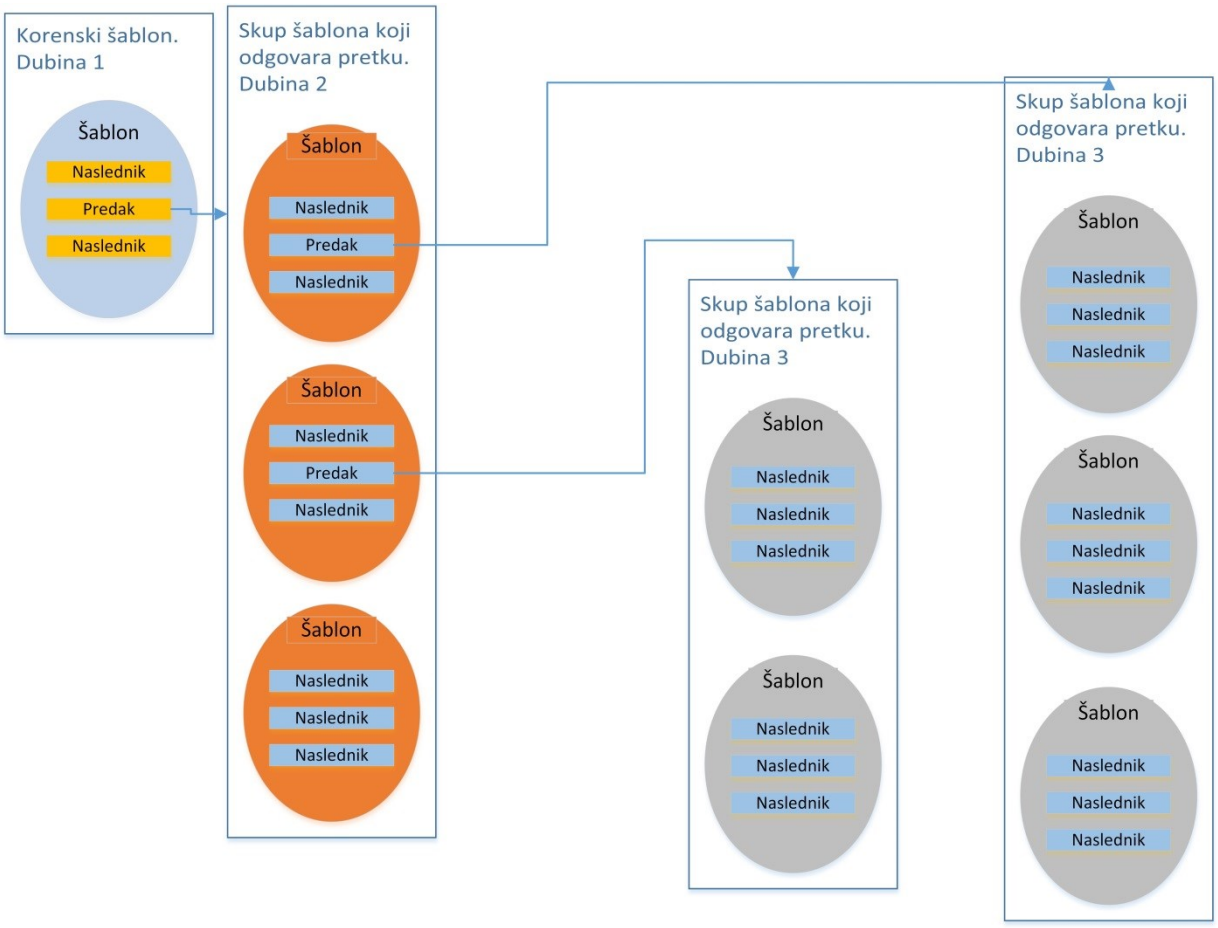
Šabloni su datoteke koje sadrže tekst sa jasno definisanim mestima u tekstu, gde opisna vrednost (eng labela-*Label*), treba da se zameni šablonskom vrednošću. Kada su u pitanju šabloni za API, oni treba da sadrže informacije o datoteci prema standardu za programski jezik. Svaki API mora da sadrži i opis svake funkcije, odnosno dokumentaciju. Opisi svake od funkcija treba da budu napisani prema jasno definisanim pravilima. Jedan od alata koji na osnovu opisa funkcija, pravi dokumentaciju je *Doxygen*. Primer zaglavlja datoteke, kao i primer opisa jedne od funkcija dat je na Slici 3.5 i Slici 2.4.

```
]/*****************************************************************************//**
  * Kratak opis sadržaja datoteke, pravima pristupa, korišćenja, presnimavanja
  *
  * @project ime projekta u okviru kojeg je datoteka
  * @file ime ove datoteke
  * @brief
  * Kratak opis svrhe API-ja iz ove datoteke
  *
  * @Author ime i prezime autora
  *
  * @notes
  * zabeleške
  *
  * @history istorija menjanja datoteke
  * @date datum nastanka datoteke
  *****************************************************************************/
```

Slika 3.5 Izgled zaglavlja datoteke prema Doxygen pravilima

Šabloni se nalaze izvan Generatora API-ja i mogu se menjati bilo kojim alatom za uređivanje teksta.

Da šabloni ne bi bili preveliki, potrebno je uzeti referentnu datoteku koja sadrži primere MAPI-ja i razložiti je na manje šablone i labele. Spajanjem ovih šablona na kraju, dobija se opet referentna datoteka. Ovo omogućava da umesto velikog broja labela za, na primer, ime funkcije, imamo jednu, koja pripada manjem šablonu koji se uklapa u veći šablon, takođe putem labele. Da bi ovo funkcionisalo, uvode se pojmovi labele pretka i labele naslednika (u daljem tekstu Predak i Naslednik). Predak je labela čija vrednost treba da se zameni popunjenim šablonom. Za svakog Pretka, vezan je tačno jedan šablon, dok za jedan šablon može da bude vezano više Predaka. Takođe, nekad je potrebno da jedan Predak treba zameniti unijom različito popunjenih šablona. Ovim se postiže da imamo jedan šablon za parametre/objekte koji se popuni nekoliko puta (za TR-135 preko 300), nakon čega se ovi šabloni upakuju u jednu vrednost i zamene vrednost Pretka. Naslednik je labela čija vrednost se menja šablonskom vrednošću koje obezbedi Pretvarač šablonskih vrednosti. Šabloni predstavljaju čvorove u stablu, Preci predstavljaju imena tih čvorova dok su Naslednici listovi stabla.Svaki skup šablona, se nalazi na određenoj dubini u stablu šablona. Organizacija ovog stabla je ista kao stablo modela podataka i prikazana je na Slici 3.6.



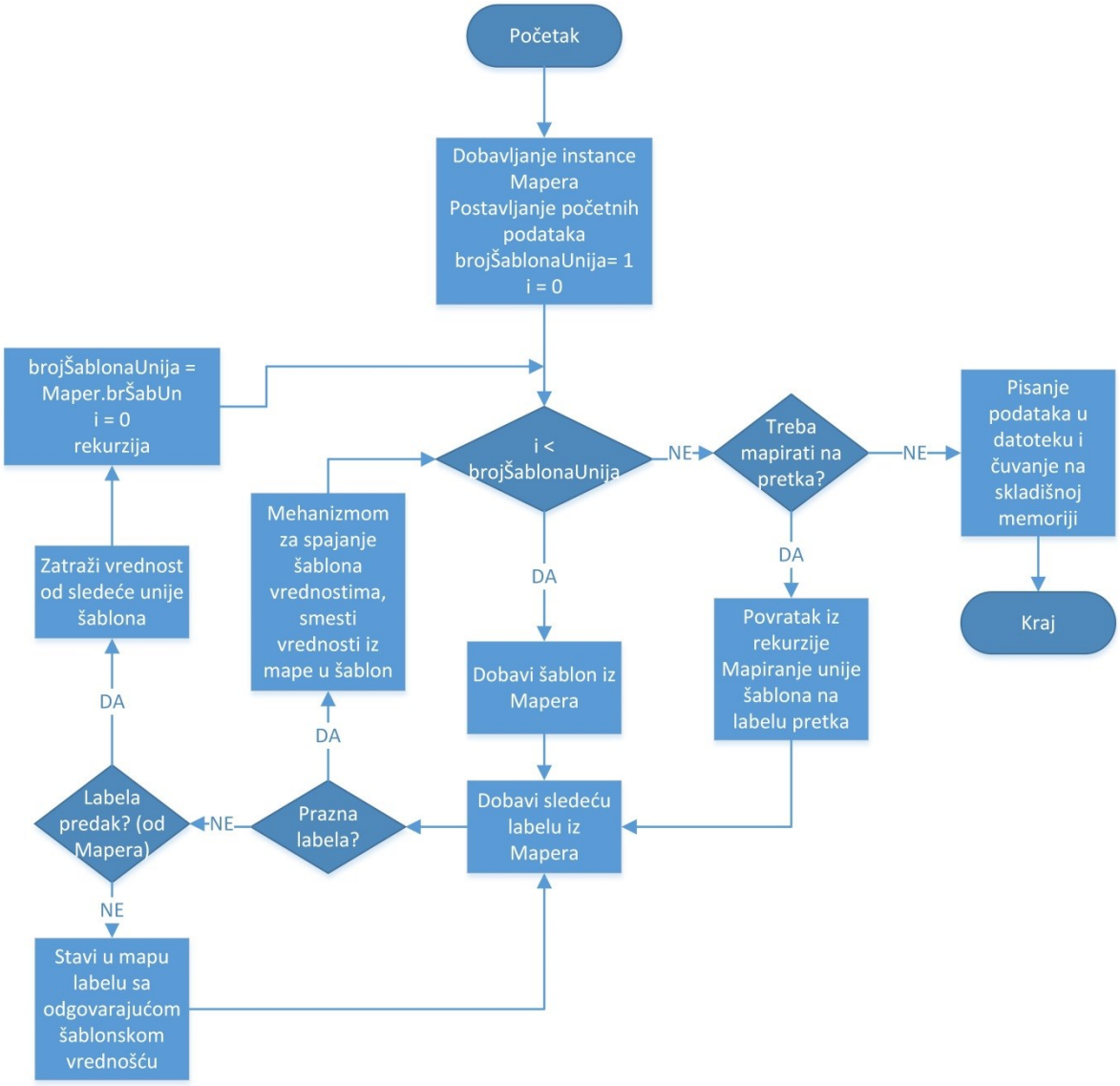
Slika 3.6 Šabloni, preci i naslednici i veze među njima

Maper sadrži tačne podatke o broju parametara i objekata za koji su potrebne API funkcije, poseduje šablone, šablonske vrednosti, zna koje labele su preci a koje naslednici i ove informacije i podatke daje Fabrici datoteka na korišćenje.

3.1.4 Fabrika datoteka

Fabrika datoteka je jednostavan modul čiji je zadatak spajanje šablona i šablonskih vrednosti u krajnju datoteku. Sastavni deo fabrike je mehanizam koji vrši umetanje šablonskih vrednosti u šablone na osnovu labela koji se nalaze unutar šablona. Pošto labele imaju tačno određeni format koji ih razlikuje od ostatka teksta u šablonu, mehanizam koristi prilagođenu memorijsku mapu u kojoj su ključevi za pretragu labele (Preci i Naslednici) a vrednosti su šablonske vrednosti ili popunjen šablon. Mapa je organizovana po principu jedan ključ, jedna vrednost. Svaka od struktura vezana je za jedan šablon. Fabrika datoteka prilikom pravljenja datoteke, obilazi stablo šablona i popunjava ga rekurzivno. Ukoliko se posmatra Slika 3.6, prvo će u potpunosti biti popunjeni šabloni iz *Skupa šablona dubine 3* koji će predstavljati vrednost za labelu predak iz prvog šablona iz *Skupa šablona dubine 2*. Zatim se popuni preostali naslednik iz prvog šablona i prvi naslednik u drugom šablonu iz *Skupa šablona dubine 2*. Nakon toga, popunjavaju se šabloni iz *Skupa šablona dubine 3*. Popunjeni, vraćaju se kao vrednost Pretka u

drugom šablonu iz *Skupa šablona dubine 2*. Detaljan algoritam procesa popunjavanja šablona i generisanja izlazne datoteke, prikazan je na Slici 3.7.



Slika 3.7 Algoritam popunjavanja šablona i pravljenja datoteke

Ime datoteke koju fabrika pravi, dobija se iz Mapera šablona i šablonskih vrednosti. Ukoliko je potrebno napraviti više datoteka za konkretnu implementaciju TR-069 klijentske biblioteke, fabrika ponavlja opisani proces. Informaciju o broju datoteka, Fabrika dobija takođe iz mapera. Opisanim algoritmom se postiže da je Fabrika u mogućnosti da napravi MAPI za bilo koji programski jezik, koji je potreban realizaciji TR-069 klijentske biblioteke. Spajanje šablona vrednostima iz mape vrši modul Mehanizam za spajanje šablona i šablonskih vrednosti.

3.1.4.1 Mehanizam za spajanje šablona i šablonskih vrednosti

Mehanizam vrši spajanje šablona i šablonskih vrednosti. On ne poznaje nijedan tip datoteke niti programski jezik. Struktura podataka koja čuva šablonske vrednosti i labele se naziva Mapa šablona. Kada se ta struktura popuni i zatraži se spajanje sa šablonom, mehanizam

uzima prvu labelu i pretražuje šablon. Ukoliko je ne pronađe u šablonu, nastavlja sa pretraživanjem. Ukoliko nađe labelu u šablonu, sve pogotke u pretrazi zamenjuje vrednošću koja je vezana za tu labelu. Ovaj proces se ponavlja dok mehanizam ne prođe kroz celu strukturu. Kada završi, vraća popunjen šablon onome koji je zatražio spajanje šablona sa šablonskim vrednostima.

3.2 Ugradnja Generatora u sistem za prevođenje

Sistem za prevođenje TR-069 klijentske biblioteke mora biti lako izmenljiv i prilagodljiv, da bi mogao podržati Generator API-ja. Pre pokretanja sistema za prevođenje, uvek je potrebno izvršiti podešavanja. Opšta podešavanja podrazumevaju podešavanja alata za prevođenje, putanje do direktorijuma gde se nalaze potrebne pomoćne biblioteke, putanja do direktorijuma gde prevedena biblioteka treba da se nalazi. U slučaju TR-069 klijentske biblioteke, potrebno je dodatno podesiti model podataka za uređaj za koji se prevodi. Sledeća lista predstavlja skup potrebnih podešavanja sistema za prevođenje zajedno sa podešavanjima za podršku Generatora API-ja:

- Opšta:
  - Podešavanje alata za prevođenje i arhiviranje;
  - Podešavanje pomoćnih biblioteka;
  - Podešavanje putanje za isporuku prevedene biblioteke;
  - Podešavanje putanje isporučenih resursa koje koristi prevedena biblioteka;
  - Podešavanje alata koji će vršiti generisanje dokumentacije;
- Za podršku Generatora API-ja:
  - Podešavanje imena modela podataka za pravljenje MAPI-ja;
  - Podešavanje putanje do modela podataka;
  - Podešavanje liste željenih profila modela podataka;
  - Podešavanje imena izlaznih MAPI datoteka iz Generatora API-ja;
  - Podešavanje putanje do direktorijuma isporuke MAPI datoteka;
  - Podešavanje pravila prevođenja, tako da sistem za prevođenje uključi napravljene MAPI datoteke

Sistem za prevođenje se proširuje zahtevima za generisanjem i prevođenjem datoteka MAPI-ja, tako što se navedu njihova imena. Imena ovih datoteka su jednaka imenima datoteka koji treba da budu izlaz Generator API-ja. Pozivanje Generatora API-ja se vrši u trenutku kada sistem za prevođenje zatraži datoteke za MAPI. Sistem za prevođenje obezbeđuje ime modela podataka, programski jezik u kom API treba da se napravi i precizira direktorijum u koji zatražene datoteke treba da se nalaze. Ukoliko je potrebno, sistem za prevođenje treba da

obezbedi spisak profila iz modela podataka koji uređaj podržava, tako da se umesto celog modela podataka, podrži samo podskup definisan tim profilima. Nakon ovoga, prevođenje se nastavlja dok se na kraju ne dobije TR-069 klijentska biblioteka sa MAPI-jem.

Dinamičnost sistema se ogleda u tome da ukoliko se malo izmeni model podataka za uređaj, izmeniće se i TR-069 klijentska biblioteka, bez potrebe za menjanjem bilo čega u samoj biblioteci. Isto tako, da bi biblioteka bila u mogućnosti da podrži neki drugi uređaj, potrebno je obezbediti model podataka za taj uređaj i u podešavanjima sistema za prevođenje biblioteke, izmeniti zahtev za datotekama MAPI-ja.

Ukoliko je navedeno u podešavanju sistema za prevođenje, nakon prevođenja biblioteke, pravi se dokumentacija na osnovu MAPI i dodatnih datoteka sa opštim API-jem TR-069 klijentske biblioteke. Ova dokumentacija se isporučuje zajedno sa TR-069 klijentskom bibliotekom.

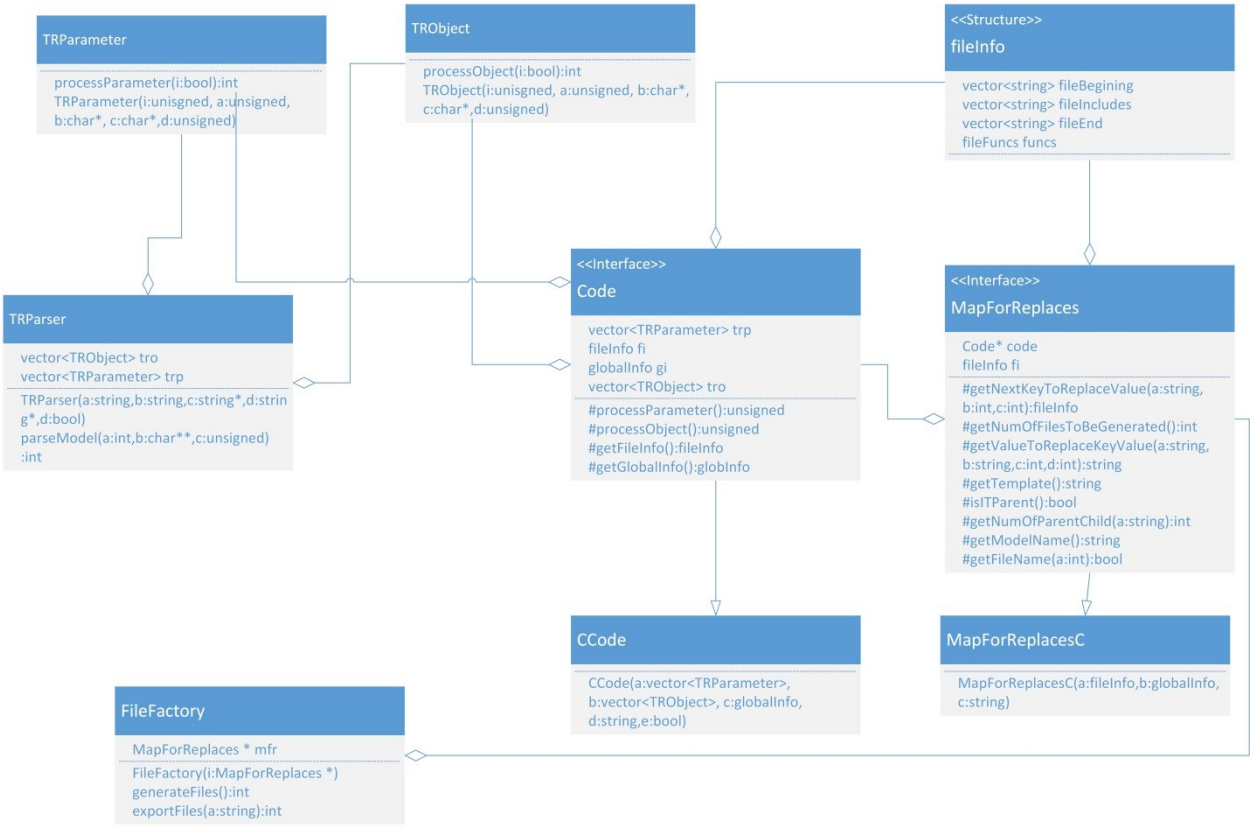
4. Programsko rešenje

Programsko rešenje odnosi se na konkretnu realizaciju Generatora API-ja i njegovo ugrađivanje u sistem prevođenja za Linux uređaje. Pri izradi Generatora API-ja korišćen je programski jezik C++. Jedan deo Parsera modula, realizovan je u programskom jeziku C. Preduslov za prevođenje i izvršavanje Generatora API-ja je postojanje sledećih biblioteka:

- libxml2 (učitavanje „.xml“ datoteka)
- boost_log (log zapis generatora)
- boost_pointer (potrebno mehanizmu koji spaja šablone sa mapom šablonskih vrednosti)

Mehanizam za spajanje šablona i šablonskih vrednosti, preuzet je kao gotovo rešenje sa [11]. Ovo rešenje je otvorenog koda sa odgovarajućom licencom i pozitivno je ocenjeno od strane korisnika.

Klase koje postoje u Generatoru API-ja, prikazane su klasnim dijagramom na Slici 4.1.



Slika 4.1 Klasni dijagram programskog rešenja

Pošto se MAPI pravi za parametre i objekte sa jednom ili više instanci, u daljem tekstu pojam objekat se odnosi na objekat sa jednom ili više instanci.

4.1 TRParameter

Klasa u koju se preslikavaju podaci o parametru iz modela podataka. Dostupne informacije obuhvataju ime parametra, imena objekata sa više instanci i njihove pozicije u putanji parametra. Takođe, na osnovu prava pristupa parametru, određuje broj potrebnih funkcija za parametar. Sadrži sledeće metode:

- `TRParameter::TRParameter(unsigned type, unsigned valueType, char* parUrl, char* parDesc, unsigned writable)`

  Opis: Konstruktor sa argumentima za klasu TRParameter. Postavlja instancu ove klase u početno stanje.

  Parametri:

  - **type:** Tip podatka (parametar ili objekat spram modela podataka...)

- **valueType:** Tip vrednosti parametra (unsigned, string, boolean...)
- **parUrl:** Putanja do parametra u modelu podataka
- **parDesc:** Opis parametra koji se nalazi u modelu(description)
- **writable:** Govori da li poslužilac može da menja vrednost parametra ili ne

Povratna vrednost: Nema

- `TRParameter::int processParameter(bool multiclient)`

  Opis: Metoda koja obrađuje primljene podatke i pretvara ih u format pogodan za sledeći modul koji će ih obrađivati. Metoda dobavlja ime parametra, imena objekata sa više instanci (ukoliko postoje na putanji) i pozicije njihovih instanci na putanji. Struktura koja sadrži ove informacije se naziva Mopi (eng. *Multiple object path information*)

  Parametri:

  - **multiclient:** Govori da li funkcije parametra treba da budu za višekorisničku TR-069 klijentsku biblioteku ili ne.

  Povratna vrednost: Informacija o uspešnosti, Bez greške: 0 Sa greškom: različito od 0

- `TRParameter::string getFuncPartialName(void)const`

  Opis: Metoda koja vraća deo imena funkcija koje treba da se naprave za parametar (ime parametra).

  Parametri: Nema

  Povratna vrednost: Ime parametra.

- `TRParameter::string getFuncParameterUrl(void)const`

  Opis: Metoda koja vraća putanju do parametra u modelu podataka.

  Parametri: Nema

  Povratna vrednost: putanja do parametra.

- `TRParameter::string getFuncParameterDescritpion(void)const`

  Opis: Metoda koja vraća opis parametra iz modela podataka.

  Parametri: Nema

  Povratna vrednost: Opis parametra.

- `TRParameter::bool getParameterWrittable(void)const`

  Opis: Metoda koja vraća informaciju da li poslužilac može da menja vrednost parametra ili ne.
  Parametri: Nema
  Povratna vrednost: True ako može, false ako ne može.

- `TRParameter::unsigned getSetGetTotalNumOfArguments(void)const`

  Opis: Metoda koja vraća ukupan broj argumenata koje funkcije Postavi(set) i Dobavi(get) treba da imaju.
  Parametri: Nema
  Povratna vrednost: Broj argumenata.

4.2 TRObject

Klasa u koju se preslikavaju podaci o objektu sa više instanci iz modela podataka. Na osnovu dobijenih informacija izvlači se ime objekta, izvlači se imena objekata sa više instanci i njihove pozicije u putanji objekta sa više instanci. Takođe, na osnovu prava pristupa objektu sa više instanci, određuje broj potrebnih funkcija za objekat. Sadrži sledeće metode:

- `TRObject::TRObject(unsigned type, unsigned valueType, char* objUrl, char* objDesc, unsigned writable)`

  Opis: Konstruktor sa argumentima za klasu TRObject. Postavlja instancu ove klase u početno stanje.
  Parametri:
  - **type:** Tip podatka (parametar ili objekat spram modela podataka...)
  - **valueType:** Tip objekta (objekat bez instance ili objekat sa jednom ili više instanci)
  - **objUrl:** Putanja do objekta u modelu podataka
  - **objDesc:** Opis objekta koji se nalazi u modelu(description)
  - **writable:** Govori da li poslužilac može da dodaje instance objekta ili ne

  Povratna vrednost: Nema

- `TRObject::int processObject(bool multiclient)`

Opis: Metoda koja obrađuje primljene podatke i pretvara ih u format pogodan za sledeći modul koji će ih obrađivati. Izdvaja ime objekta, imena objekata sa više instanci (ukoliko postoje na putanji) i pozicije njihovih instanci na putanji. Struktura koja sadrži ove informacije se naziva Mopi (eng. *Multiple object path information*)

Parametri:

- **multiclient:** Govori da li funkcije objekta treba da budu za višekorisničku TR-069 klijentsku biblioteku ili ne.

Povratna vrednost: Informacija o uspešnosti, Bez greške: 0 Sa greškom: različito od 0

- `TRObject::string getFuncPartialName(void)const`

Opis: Metoda koja vraća deo imena funkcija koje treba da se naprave za objekat (ime objekta).

Parametri: Nema

Povratna vrednost: Ime objekta.

- `TRObject::string getFuncObjectUrl(void)const`

Opis: Metoda koja vraća putanju do objekta u modelu podataka.

Parametri: Nema

Povratna vrednost: putanja do objekta.

- `TRObject::string getFuncObjectDescritpion(void)const`

Opis: Metoda koja vraća opis objekta iz modela podataka.

Parametri: Nema

Povratna vrednost: Opis objekta.

- `TRObject::bool getObjectWrittable(void)const`

Opis: Metoda koja vraća informaciju da li poslužilac može da doda/oduzme instance objekta ili ne.

Parametri: Nema

Povratna vrednost: True ako može, false ako ne može.

- `TRObject::unsigned getTotalNumOfArguments (void)const`

Opis: Metoda koja vraća ukupan broj argumenata koje funkcije Dodaj(set) i Ukloni(get) treba da imaju.
Parametri: Nema
Povratna vrednost: Broj argumenata.

4.3 TRParser

Klasa zadužena za parsiranje modela podataka i čuvanje isparsiranih podataka. Koristi kontejnersku klasu vector iz Standardne šablonske biblioteke(eng. STL – *Standard Template Library*)[12] u kojoj se nalaze instance obrađenih čvorova iz modela podataka(po jedna za TRParameter i TRObject). Sledećem modulu u procesu pravljenja API datoteka, prosleđuje prikupljene informacije u vidu dve kontejnerske klase vector gde svaka sadrži instanci klase TRParameter, odnosno TRObject. Instanci ovih klasa ima onoliko, koliko parser pronađe parametara u modelu podataka, odnosno objekata sa jednom ili više instanci. Deo ove klase, koji vrši parsiranje modela i izdvajanje čvorova iz modela pisan je u programskom jeziku C i koristi libxml2 biblioteku za parsiranje.

4.3.1 Deo pisan u programskom jeziku C

- `repo_err_t repo_create_data_tree(const char* root_name, xmlDocPtr model_doc, xmlDocPtr* data_doc, const char *folder_path, repository_handle_t* create_repo)`

  Opis: Funkcija koja pravi stablo na osnovu celog modela.
  Parametri:
  - **root_name:** Ime korenskog čvora.
  - **data_doc:** Pokazivač na stablo koje će se napraviti, izlazni parametar
  - **model_doc:** Korenski čvor u modela
  - **folder_path:** Putanja do direktorijuma koji sadrži zadati model podataka.
  - **create_repo:** Handle na napravljeni repozitorijum

  Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0.
- `repo_err_t repo_create_data_tree_from_profiles(const char* root_name, xmlDocPtr model_doc, xmlDocPtr* data_doc, const char *folder_path, repository_handle_t* create_repo, char** profiles, unsigned num_of_profiles)`

Opis: Funkcija koja pravi stablo na osnovu, profilima definisanim, delova modela.

Parametri: XY

- **root_name:** Ime korenskog čvora.
- **data_doc:** Pokazivač na stablo koje će se napraviti, izlazni parametar
- **model_doc:** Korenski čvor u modela
- **folder_path:** Putanja do direktorijuma koji sadrži zadati model podataka.
- **create_repo:** Handle na napravljeni repozitorijum
- **profiles:** Lista profila za model
- **num_of_profiles:** Broj prosleđenih profila

Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0.

- `repo_err_t repo_code_generator_hook(xmlNodePtr data_node, repository_handle_t repo_handle)`

Opis: Funkcija koja rekurzivno obilazi napravljeno stablo u potrazi za parametrima i objektima, i preko povratnog mehanizma(eng *callback*) obaveštava TRParser i prosleđuje mu podatke.

Parametri:

- **data_node:** Čvor u stablu.
- **repo_handle:** Handle na napravljeni repozitorijum

Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0.

- `repo_err_t repo_load_model_tree(const char* model_filename, const char* folder_path, xmlDocPtr * model_doc, const char* model_folder_root)`

Opis: Funkcija učitava model i sjedinjuje ga sa svim modelima od kojih zavisi. Model različit od TR-106 ne može da se spoji ukoliko TR-106 vec nije inicijalizovan, jer on obezbeđuje korenski objekat u stablu(„Device“).

Parametri:

- **folder_path:** Putanja do direktorijuma koji sadrži zadati model podataka.
- **model_filename:** Ime „.xml“ datoteke modela podataka
- **model_doc:** Korenski čvor u modelu
- **model_folder_root:** Putanja do direktorijuma koji sadrži sve modele podataka

Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0.

- `repo_err_t data_repo_instantiate(unsigned *repo_id, const char* model_filename, const char* model_folder, const char* initialization_filename, const char* backup_filename, char** profiles, unsigned num_of_profiles, const char *model_folder_root, unsigned *should_hook)`

  Opis: Funkcija koja objedinjuje sve do sad navedene funkcije. Ona se poziva unutar klase TRParser.

  Parametri:

  - **repo_id:** Izlazni argument, identifikacioni broj repozitorijuma.
  - **model_filename:** Ime „.xml“ datoteke modela podataka
  - **initialization_filename:** Ime stabla repozitorijuma
  - **backup_filename:** Ime kopije stabla repozitorijuma
  - **model_folder:** Putanja do direktorijuma koji sadrži zadati model podataka
  - **profiles:** Lista profila za dati model podataka, može biti prazna
  - **num_of_profiles:** Broj profila ukoliko su zadati, 0 u suprotnom
  - **model_folder_root:** Putanja do direktorijuma koji sadrži sve modele podataka
  - **should hook:** Parametar koji govori da li je potrebno vraćati callback za prosleđeni model podataka.

  Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0.

- `int data_repo_wrap_reg_callback(int type, void* instancePointer, paramDRECallbackCPP func)`

  Opis: Funkcija koja prijavljuje callback metodu iz klase TRParser. Na osnovu vrednosti argumenta *type* određuje se da li je prosleđeni poakzivač na metodu za parametre ili za objekte.

  Parametri:

  - **type:** Tip callback-a koji se prijavljuje, za parametre ili za objekte.
  - **instance_pointer:** Pokazivač na instancu klase TRParser jer TRParser nije jedno-instancna klasa (eng. *singleton*).

- **func:** Statička metoda iz klase TRParser koja se poziva kada se pri obilasku kroz stablo naiđe na parametar/objekat

Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0.

4.3.2 Deo pisan u programskom jeziku C++

Metode klase TRParser:

- `TRParser::TRParser(string rootDirectory, string modelName, string* intermediateModelPath = NULL, string* tr106Model = NULL, bool multiclient = false)`

  Opis: Konstruktor sa argumentima. Baca izuzeće ukoliko je neki od prosleđenih argumenata, prazan (NULL), loše formatiran (putanje) ili nije uspešno alociranje memorije. Pravi podatke potrebne funkciji data_repo_instantiate.

  Parametri:

  - **rootDirectory:** Putanja do korenskog direktorijuma koji sadrži sve modele podataka
  - **modelName:** Ime modela podataka za koji treba da se pravi MAPI
  - **intermediateModelPath:** Putanja između lokacije modela podataka i rootDirectory ukoliko postoji.
  - **tr106Model:** Ime TR-106 modela podataka. Model različit od TR-106, ne može biti parsiran ukoliko TR-106 nije prethodno inicijalizovan, jer TR-106 definiše korenski čvor.
  - **multiclient:** Definiše da li se MAPI pravi za TR-069 klijentsku biblioteku koja podržava višekorisničku upotrebu ili ne.

  Povratna vrednost: Nema.

- `TRParser::TRParser()`

  Opis: Konstruktor bez argumenata

  Parametri: Nema

  Povratna vrednost: Nema

- `TRParser::static void callbackForParams(void * entry, void * userData)`

  Opis: Statička metoda unutar TRParser klase. Koristi se kao callback za parametre.

  Parametri:

- **entry:** Struktura koja sadrži sve informacije potrebne za parametar.
- **userData:** Pokazivač na instancu TRParser klase koja je registrovala callback.

Povratna vrednost: Nema

- `TRParser::static void callbackForObjects(void * entry, void * userData)`

  Opis: Statička metoda unutar TRParser klase. Koristi se kao callback za objekte.
  Parametri:
  - **entry:** Struktura koja sadrži sve informacije potrebne za objekat.
  - **userData:** Pokazivač na instancu TRParser klase koja je registrovala callback.

  Povratna vrednost: Nema
- `TRParser::int registerCallbacks(void)`

  Opis: Metoda koja registruje callback metode za parametre i objekte.
  Parametri: Nema
  Povratna vrednost: Nema
- `TRParser:: int parseModel(int logState = 0, char** profiles = NULL, unsigned num_of_profiles = 0)`

  Opis: Metoda koja poziva data_repo_instantiate i tad započinje parsiranje modela. Kad se popune vectore-i TRParametra i TRObjecta, onda se prolazi kroz obe strukture pozivajući metode processParameter i process Object respektivno.
  Parametri: Nisu obavezni
  - **log_state:** Nivo loga prilikom parsiranja modela podataka.
  - **profiles:** Lista profila za model koji se želi parsirati.
  - **num_of_profiles:** Broj profila u listi

  Povratna vrednost: 0 ukoliko sve prođe bez greške, različito od nule u suprotnom
- `TRParser:: vector<TRParameter> getParameters()const`

  Opis: Metoda dobavlja vector popunjen TRParameter instancama.
  Parametri: Nema
  Povratna vrednost: vector popunjen TRParameter instancama.

- `TRParser::vector<TRObject> getObjects()const`

  Opis: Metoda dobavlja vector popunjen TRObject instancama.
  Parametri: Nema
  Povratna vrednost: vector popunjen TRObject instancama.

4.4 Code

Code klasa predstavlja spregu (interfejs) koja obezbeđuje dovoljan skup funkcionalnosti i metoda, tako da se može podržati bilo koji programski jezik za koji se MAPI pravi i bilo koja implementacija TR-069 klijentske biblioteke. Zadatak ove klase je da na osnovu svake instance TRParameter-a i TRObject-a, napravi šablonske vrednosti koje će biti umetnute u šablon umesto labela. TRParser dobavlja vector-e sa instancama ovih klasa. Kao izlaz, Code interfejs daje strukturu koja sadrži sve šablonske vrednosti. Definiše kontejnersku strukturu koja se popuni šablonski vrednostima nakon obrade TRParameter i TRObject instanci. Ime ove strukture je *fileInfo*. Takođe, definisana je i *globalInfo* struktura koja sadrži globalne šablonske vrednosti kao što su: ime datoteke, ime modela podataka, revizija modela, datum pravljenja datoteke i godina.

- `Code::virtual unsigned processParameter (void)=0`

  Opis: Metoda koja treba da obradi sve parametre koje Code dobije da bi napravila šablonske vrednosti i sve tevrednosti stavi u odgovarajuće kontejnere.
  Parametri: Nema
  Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0

- `Code::virtual unsigned processObject (void)=0`

  Opis: Metoda koja treba da obradi sve objekte koje Code dobije da bi napravila šablonske vrednosti i sve tevrednosti stavi u odgovarajuće kontejnere.
  Parametri: Nema
  Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0

- `Code::virtual fileInfo getFileInfo(void)const=0`

  Opis: Metoda koja dobavlja napunjenu *fileInfo* strukturu. Ova struktura se isporučuje sledećem modulu u obradi, MapForReplaces.
  Parametri: Nema
  Povratna vrednost: popunjena struktura fileInfo

- `Code:: virtual globalInfo getGlobalInfo(void)const=0`

  Opis: Metoda koja dobavlja napunjenu globalInfo strukturu. Ova struktura se isporučuje sledećem modulu u obradi, MapForReplaces.

  Parametri: Nema

  Povratna vrednost: popunjena struktura globalInfo

- ```
  struct fileInfo{
   vector<string> fileBegining;
   vector<string> fileIncludes;
   fileFuncs      funcs;
   vector<string> fileEnd;
  }
  ```

  Opis: Struktura koja sadrži šablonske vrednosti za sve datoteke za MAPI koje treba da se naprave. Organizacija ove strukture, prati organizaciju svake datoteke specifične za bilo koji programski jezik.

  Članovi:

  - **fileBegining:** Struktura *vector* sa *string* elementima. Označava početak datoteke. Na primer u programskom jeziku C, početakom se smatraju definisani makroi, enumeracije, makro po kom se „.h“ datoteka prepoznaje. Broj elemenata u ovoj strukturi je jednak broju datoteka koje Generator API-ja pravi. Elementi mogu da budu prazni.
  - **fileIncludes:** Struktura *vector* sa *string* elementima. Apstrahuje uvlačenje datoteka/paketa za umetanje. Na primer u programskom jeziku C, to su #include direktive, u programskom jeziku Java to su paketi. Broj elemenata u ovoj strukturi je jednak broju datoteka koje Generator API-ja pravi. Elementi mogu da budu prazni.
  - **fileEnd:** Struktura *vector* sa *string* elementima. Apstrahuje kraj datoteka. Na primer u programskom jeziku C, to je kraj deklaracije direktive prema kome se „.h“ datoteka prepoznaje. Broj elemenata u ovoj strukturi je jednak broju datoteka koje Generator API-ja pravi. Elementi mogu da budu prazni.
  - **fileFuncs:** Struktura koja apstrahuje API funkcije za parametre i objekte. U sebi sadrži dve instance struktura *paramFuncs* i *objFuncs*.

- ```
  struct paramFuncs{
   vector<string> parUrl;
  ```

```
 vector<string> funcsDesc;
 vector<string> groupOrdinal;
 vector<codeNode> setFuncInfo;
 vector<codeNode> getFuncInfo;
 vector<codeNode> regFuncInfo;
 vector<codeNode> unregFuncInfo;
};
```

Opis: Struktura koja sadrži sve informacije da za pravljenje API-ja za parametre. Svaki *vector* ima podjednak broj elemenata i on je jednak broju parametara izvučenih iz modela podataka. Redni broj u svakoj od struktura odgovara istom parametru. Na primer, peti element u svakom od *vector*-a vezan je za isti parametar.

Članovi:

- **parUrl:** Struktura *vector* sa *string* elementima. URL parametra u modelu (putanja).
- **funcsDesc:** Struktura *vector* sa *string* elementima. Opis parametra izvučen iz modela podataka, obrađen. Koristi se za dokumentaciju, za opis grupe funkcija vezanih za parametar (Postavi, Dobavi, Prijavi, Odjavi).
- **groupOrdinal:** Struktura *vector* sa *string* elementima. Redni broj parametra. Koristi se za grupisanje funkcija vezanih za parametar u dokumentaciji.
- **setFuncInfo:** Struktura *vector* sa *codeNode* elementima. Sadrži šablonske vrednosti za Postavi funkcije.
- **getFuncInfo:** Struktura *vector* sa *codeNode* elementima. Sadrži šablonske vrednosti za Dobavi funkcije.
- **regFuncInfo:** Struktura *vector* sa *codeNode* elementima. Sadrži šablonske vrednosti za Prijavi funkcije.
- **unregFuncInfo:** Struktura *vector* sa *codeNode* elementima. Sadrži šablonske vrednosti za Odjavi funkcije.

-
```
struct objFuncs{
 vector<string> objUrl;
 vector<string> funcsDesc;
 vector<string> groupOrdinal;
 vector<codeNode> addFuncInfo;
 vector<codeNode> delFuncInfo;
```

```
 vector<codeNode> regFuncInfo;
 vector<codeNode> unregFuncInfo;
};
```

Opis: Struktura koja sadrži sve informacije da za pravljenje API-ja za objekte. Svaki *vector* ima podjednak broj elemenata i on je jednak broju objekata izvučenih iz modela podataka. Redni broj u svakoj od struktura odgovara istom objektu. Na primer, peti element u svakom od *vector*-a vezan je za isti objekat.

Članovi:

- **objUrl:** Struktura *vector* sa *string* elementima. URL objekta u modelu (putanja).
- **funcsDesc:** Struktura *vector* sa *string* elementima. Opis objekta izvučen iz modela podataka, obrađen. Koristi se za dokumentaciju, za opis grupe funkcija vezanih za objekat (Dodaj, Ukloni, Prijavi, Odjavi).
- **groupOrdinal:** Struktura *vector* sa *string* elementima. Redni broj objekta. Koristi se za grupisanje funkcija vezanih za objekat u dokumentaciji.
- **setFuncInfo:** Struktura *vector* sa *codeNode* elementima. Sadrži šablonske vrednosti za Dodaj funkcije.
- **getFuncInfo:** Struktura *vector* sa *codeNode* elementima. Sadrži šablonske vrednosti za Ukloni funkcije.
- **regFuncInfo:** Struktura *vector* sa *codeNode* elementima. Sadrži šablonske vrednosti za Prijavi funkcije.
- **unregFuncInfo:** Struktura *vector* sa *codeNode* elementima. Sadrži šablonske vrednosti za Odjavi funkcije.

-
```
struct codeNode{
 bool empty;
 map<string,string> nodeInfo;
 void clear(){
  empty = false;
  nodeInfo.clear();
 }
};
```

Opis: Struktura koja sadrži sve informacije za pravljenje jednog tipa API funkcija, bilo za parametre ili za objekte. U mapu *nodeInfo* se dodaju šablonske vrednosti po

ključu labela Naslednika iz šablona. Ima metodu koja čisti sadržaj u strukturi i dovodi u početno stanje instancu objekta ove strukture.

Članovi:

- **empty:** Informacija o tome da li je struktura prazna. Znači da za vezani parametar/objekat ne treba da se pravi API funkcija.
- **nodeInfo:** Memorijski kontejner *map* koja sadrži vrednosti tipa *string* prema ključu tipa *string*. Vrednosti su šablonske vrednosti, ključevi su labele Naslednici iz šablona.

- `nodeInfo::clear ()=0`

Opis: Metoda strukture *nodeInfo* koja dovodi instancu objekta ove strukture u početno stanje

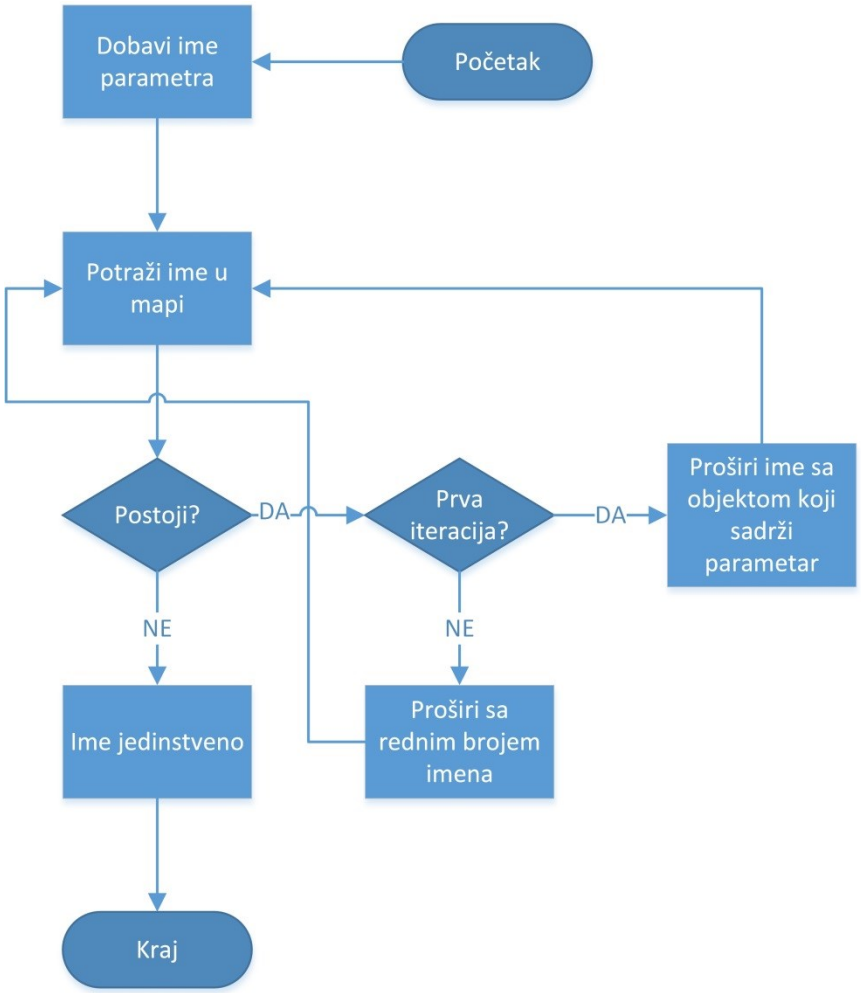
Parametri: Nema.

Povratna vrednost: Nema.

4.4.1 Primena interfejsa Code za programski jezik C: CCode

Klasa koja pravi šablonske vrednosti za programski jezik C i prilagodno realizaciji TR-069 klijentske biblioteke opisane u Poglavlju 2.2 se naziva CCode i primenjuje interfejs Code. Ovakva klasa realizuje funkcionalnost definisanu Code interfejsom. Prikazan je deo pomoćnih metoda koje su omogućile pravljenje šablonskih vrednosti.

Prilikom pravljenja imena za funkciju koja je vezana bilo za parametar ili objekat, potrebno je paziti da to ime bude jedinstveno. Prilikom instanciranja objekta klase CCode, pravi se mapa imena parametara i mapa imena objekata, čije su vrednosti broj pojavljivanja parametara odnosno objekata sa istim imenom, a ključevi su imena parametara odnosno objekata. Ukoliko je vrednost vezana za ime parametra/objekta 1, ime tog parametra/objekta je jedinstveno i ne treba ga menjati. U suprotnom, ime je potrebno proširiti. Ime se proširuje iz minimalno dve iteracije. Prva iteracija podrazumeva spajanje imena parametra/objekta sa imenom objekta koji ga sadrži. Vrši se provera jedinstvenosti imena, ukoliko prođe ide se dalje, ukoliko ne, započinje se druga iteracija. U drugoj iteraciji se na ime spaja redni broj pojavljivanja tog imena. Nakon ove iteracije, ime funkcije za parametar/objekat je jedinstveno. Algoritam dodeljivanja jedinstvenog imena je prikazan na Slici 4.2.



Slika 4.2 Algoritam dodeljivanja imena funkcijama za parametar/objekat

- `CCode::CCode(vector<TRParameter> trp, vector<TRObject> tro, globalInfo globInfo, string manufacturer="RT-RK", bool multiclient = 0)`

  Opis: Konstruktor sa argumentima. Iz klase TRParser, prosleđuju se strukture koje sadrže instance objekata klasa TRParameter i TRObject, kao i donekle popunjena globalna struktura *globInfo*.

  Parametri:

  - **trp:** Struktura *vector* sa elementima klase TRParameter. Isparsirani parametri iz modela podataka.
  - **tro:** Struktura *vector* sa elementima klase TRObject. Isparsirani objeki iz modela podataka.
  - **manufacturer:** Proizvođač TR'069 klijentske biblioteke. Podrazumevana vrednost „RT-RK" (Računarska tehnika i računarske komunikacije).

  Povratna vrednost: Nema.

- `CCode:: string generateFuncNameParam(int num, string paramType)`

  Opis: Metoda koja pravi ime za određenu funkciju koja je vezana za parametar.

  Parametri:

  - **num:** Redni broj parametra.
  - **paramType:** Tip funkcije parametra za koju je potrebno napraviti ime. Postavi, Dobavi, Prijavi ili Odjavi.

  Povratna vrednost: Ime funkcije.

- `CCode:: string generateFuncNameObj(int num, string objType)`

  Opis: Metoda koja pravi ime za određenu funkciju koja je vezana za objekat.

  Parametri:

  - **num:** Redni broj objekta.
  - **objType:** Tip funkcije objekta za koju je potrebno napraviti ime. Dodaj, Ukloni, Prijavi ili Odjavi.

  Povratna vrednost: Ime funkcije.

- `CCode:: string generateFuncDescsParam(string paramType)`

  Opis: Metoda koja pravi opis funkcionalnosti za određenu funkciju koja je vezana za parametar.

  Parametri:

  - **paramType:** Tip funkcije parametra za koju je potrebno napraviti opis funkcionalnosti. Postavi, Dobavi, Prijavi ili Odjavi.

  Povratna vrednost: Opis funkcije.

- `CCode:: string generateFuncDescsObj(string objType)`

  Opis: Metoda koja pravi opis funkcionalnosti za određenu funkciju koja je vezana za objekat.

  Parametri:

  - **objType:** Tip funkcije objekta za koju je potrebno napraviti opis funkcionalnosti. Dodaj, Ukloni, Prijavi ili Odjavi.

  Povratna vrednost: Opis funkcije.

- `CCode::unsigned reformatUrl(string &url, unsigned type, unsigned num);`

  Opis: Metoda koja vrši formatiranje URL bilo parametra ili objekta.

  Parametri:

  - **url:** URL (putanja) do parametra/objekta.
  - **type:** Određuje da li je u pitanju URL do parametra ili objekta
  - **num:** Redni broj parametra/objekta za koji je URL vezan.

  Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0

- `CCode:: unsigned reformatDescription(string &desc, unsigned type, unsigned num);`

  Opis: Metoda koja vrši formatiranje opisa bilo parametra ili objekta izvučenog iz modela podataka.

  Parametri:

  - **desc:** Opis parametra/objekta iz modela podataka.
  - **type:** Određuje da li je u pitanju opsi parametra ili objekta
  - **num:** Redni broj parametra/objekta za koji je opis vezan.

  Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0

4.5 MapForReplaces

MapForReplaces klasa je interfejs koji obezbeđuje dovoljan skup funkcionalnosti i metoda, tako da se može podržati bilo koji programski jezik za koji se MAPI pravi i bilo koja implementacija TR-069 klijentske biblioteke. Zadatak ove klase je da poveže (mapira) šablone sa labelama Precima i labele Naslednika iz šablona na šablonske vrednosti. Skup metoda koje obezbeđuje je potreban i dovoljan da bi sledeći modul FileFactory, mogao da pravi datoteke. U sebi sadrži popunjene strukture *fileInfo* i *globalInfo* koje dobije od prethodnog modula Code. Sa Slike 3.6, prva kolona šablona je korenski šablon (Dubina 1 u stablu šablona) i uvek postoji samo jedan koji odgovara jednoj datoteci. Druga kolona (Dubina 2 u stablu šablona) predstavlja uniju šablona ili za parametre ili za objekte. U uniji se nalazi onoliko šablona koliko je u modelu pronađeno parametara/objekata. Treća kolona šablona predstavlja skup funkcija za jedan parametar/objekat i ima ih četiri i za parametre i za objekte(Dubina 3 u stablu šablona).

- `MapForReplaces:: virtual string getNextKeyToReplaceValue (string parent, int ordinal, int fileType) = 0`

  Opis: Metoda koja dobavlja sledeću labelu iz šablona.

  Parametri:

  parent: Ime labele Pretka. Prazno ukoliko je u pitanju korenski šablon. Vidi sliku **Error! Reference source not found.**.

  ordinal: U zavisnosti od dubine strukture stabla šablona do koje se stiglo, predstavlja ili redni broj parametra/objekta ili redni broj funkcije za parametar/objekat.

  fileType: Tip datoteke koja se pravi.

  Povratna vrednost: Naredna labela ukoliko postoji.

- `MapForReplaces:: virtual int getNumOfFilesToBeGenerated (void) = 0`

  Opis: Metoda koja dobavlja broj datoteka koji treba da se napravi. Svaki broj u nizu od 1 do broja datoteka predstavlja tip datoteke.

  Parametri: Nema

  Povratna vrednost: Broj datoteka koje treba da se naprave.

- `MapForReplaces::virtual string getValueToReplaceKeyValue (string keyToReplace, string parent, int ordinal, int parentOrdinal) = 0`

  Opis: Metoda koja šablonsku vrednost za labelu.

  Parametri:

  keyToReplace: Ime labele koja treba da se zameni.

  parent: Ime labele Pretka, prazno ukoliko nema Pretka.

  ordinal: U zavisnosti od dubine strukture stabla šablona u kom se trenutno nalazi, predstavlja ili redni broj parametra/objekta ili redni broj funkcije za parametar/objekat.

  parentOrdinal: Redni broj koji je ili 1 ili redni broj parametra/objekta za koji se trenutno popunjava šablon za funkcije.

  Povratna vrednost: Šablonska vrednost.

- `MapForReplaces:: virtual string getTemplate (string parent, int parentOrdinal, int fileType) = 0`

  Opis: Metoda koja dobavlja šablon za labelu Predak ukoliko Predak postoji.

  Parametri:

  fileType: Tip datoteke koja se pravi.

  parent: Ime labele Pretka, prazno ukoliko nema Pretka.

  parentOrdinal: Redni broj koji je ili 1 ili redni broj parametra/objekta za koji se trenutno popunjava šablon za funkcije.

  Povratna vrednost: Šablon.

- `MapForReplaces:: virtual bool isItParent (string keyToReplace) = 0`

  Opis: Metoda koja proverava tip labele.

  Parametri:

  keyToReplace: Labela.

  Povratna vrednost: True ako je labela Predak, False ako nije.

- `MapForReplaces:: virtual int getNumOfParentChildren (string parent) = 0`

  Opis: Metoda koja dobavlja broj šablona koji čine uniju za labelu Pretka.

  Parametri:

  parent: Labela Predak.

  Povratna vrednost: Broj šablona u uniji.

- `MapForReplaces:: virtual string getModelName(void) = 0`

  Opis: Metoda koja dobavlja ime modela podataka za koji se pravi MAPI.

  Parametri: Nema

  Povratna vrednost: Ime modela podataka.

- `MapForReplaces:: virtual string getFileName (int fileType) = 0`

  Opis: Metoda koja dobavlja ime datoteke za rposleđeni tip datoteke.

  Parametri:

  fileType: Tip datoteke koja se pravi.

Povratna vrednost: Ime datoteke.

4.5.1 Primena interfejsa MapForReplaces za programski jezik C: MapForReplacesC

Klasa MapForReplacesC predstavlja realizaciju interfejsa MapForReplaces. Realizacija prati koncept definisan u Poglavlju 3.1.3. Za realizaciju ove klase, nisu bile potrebne dodatne metode, već samo konstruktor.

- `MapForReplacesC::MapForReplacesC (fileInfo fInfo, globalInfo gInfo, string pathToTemplates)`

  Opis: Konstruktor sa argumentima. Iz prethodnog modula Code, prosleđuju se popunjene strukture *fileInfo* i *globalInfo*. Takođe, potrebno je da se prosledi putanja u sistemu datoteka da bi objekat klase kako bi modulu MapForReplacesC bilo poznato gde da pronađe šablone i da bi mogao da ih učita.

  Parametri:

  - **fInfo:** Struktura koja sadrži sve šablonske vrednosti za parametre i objekte.
  - **gInfo:** Struktura koja sadrži globalne šablonske vrednosti.
  - **pathToTemplates:** Putanja u sistemu datoteka (eng. *filesystem*) do šablona.

  Povratna vrednost: Nema.

4.6 FileFactory

FileFactory klasa je realizovana prema algoritmu sa Slike 3.7. Objekat klase FileFactory prilikom instanciranja dobija instancu MapForReplaces. Koristeći opisane metode iz MapForReplaces interfejsa, FileFactory pravi datoteke sa MAPI-jem.

- `FileFactory:: FileFactory (MapForReplaces * mfr)=0`

  Opis: Konstruktor sa argumentima. Dobija pokazivač na instancu bilo koje realizacije interfejsa MapForReplaces.

  Parametri:

  - **mfr:** Pokazivač na instancu bilo koje realizacije MapForReplaces interfejsa

  Povratna vrednost: Nema.

- `FileFactory:: int generateFiles(void)`

  Opis: Metoda koja pravi datoteke. Poziva metodu *fillTemplate*

Parametri: Nema

Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0

- `FileFactory:: int exportFiles(string directory)`

  Opis: Metoda koja snima datoteke na zadatu putanju u sistemu datoteka.

  Parametri:

  - **directory:** Putanja na koju treba da se snime napravljene datoteke.

  Povratna vrednost: Bez greške: 0 Sa greškom: različito od 0

- `FileFactory:: string removeBlankNewlines(string file)`

  Opis: Metoda koja uklanja višak praznih redova(eng. *new line*) iz napravljene datoteke.

  Parametri:

  - **file:** Datoteka.

  Povratna vrednost: Datoteka sa uklonjenim viškom praznih redova.

- `FileFactory:: string fillTemplate(string parent, int parentOrdinal, int fileType, int children)`

  Opis: Metoda koja realizuje algoritam sa Slike 3.7. Rekurzivna.

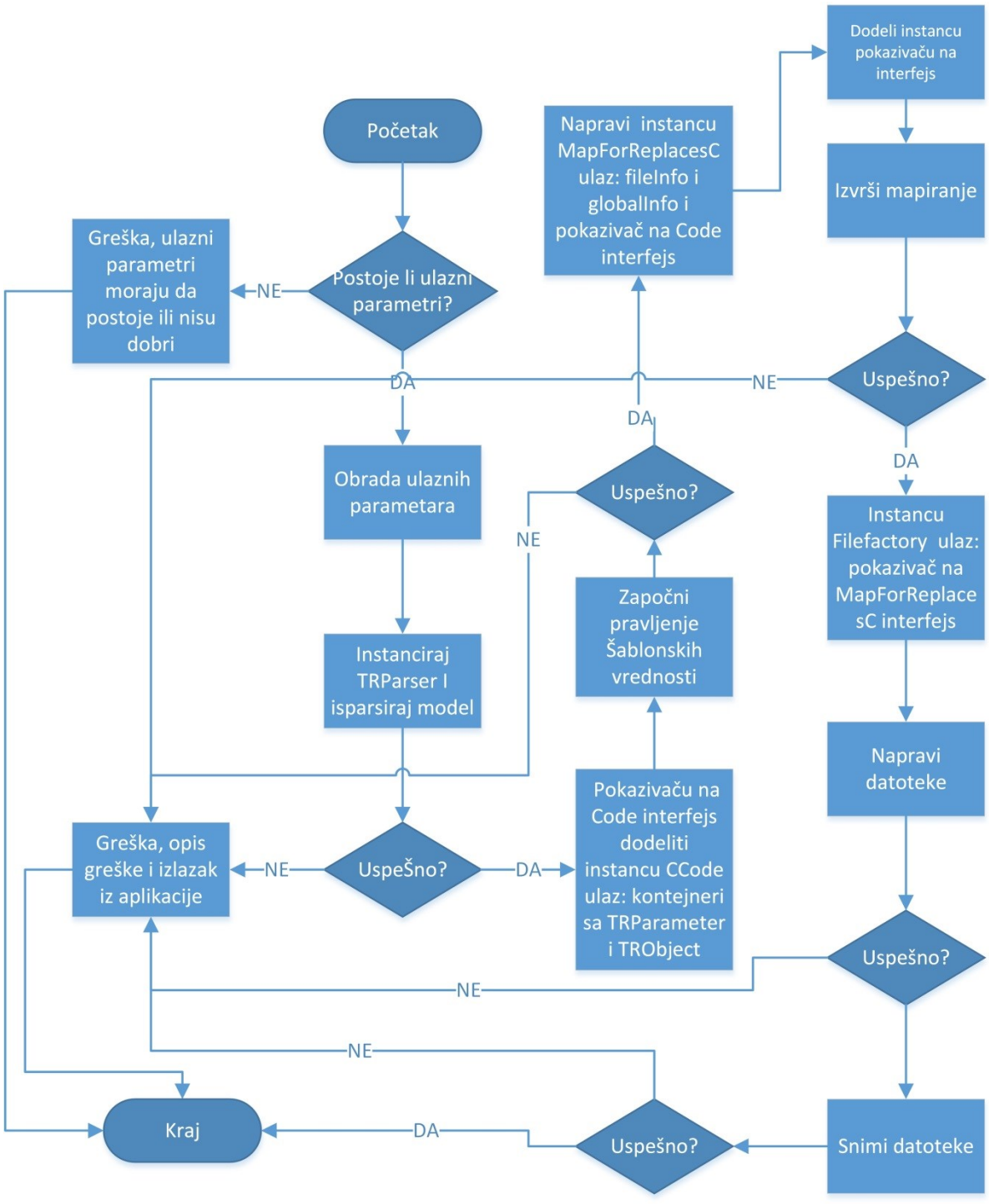
  Parametri:

  - **parent:** Labela Predak. Može biti prazna.
  - **parentOrdinal:** Ukoliko labela Predak nije prazna, predstavlja redni broj šablona iz kog je predak.
  - **fileType:** Tip datoteke.
  - **children:** Broj šablona u uniji koji će zameniti vrednost pretka.

  Povratna vrednost: Datoteka.

4.7 Algoritam aplikacije Generator API funkcija

Generator API funkcija je aplikacija koja se pokreće i izvršava preko komandne linije. Pri svom izvršavanju, korisniku ispisuje uputstvo kako se koristi. Na ekran ispisuje spisak komandi, šta svaka od njih predstavlja i koje komande su obavezne, a koje nisu. Algoritam aplikacije prikazan je na Slici 4.3.



Slika 4.3 Algoritam Generator API aplikacije

4.8 Podrška u sistemu za prevođenje

Sistem za prevođenje koji se koristi za TR-069 klijentsku biblioteku je CMAKE. Pošto je organizacija TR-069 klijentske biblioteke modularna, izmena koja je potrebna da bi se podržalo dinamičko prevođenje je minimalna. CMakeLists.txt datoteka za taj modul se proširuje uvlačenjem izvršne datoteke (aplikacije) Generatora API-ja i dodavanjem namenske komande (eng. *custom command*). Prema pravilima za definisanje namenske komande, dodaje se poziv Generatora API-ja i prosleđuju mu se argumenti iz okruženja sistema za prevođenje. Ovi argumenti su prethodno podešeni putem konfiguracione datoteke. Kao izlaz, namensko odredište preuzima napravljenje datoteke sa MAPI-jem i smešta ih na putanju Deval modula koji se prevodi. Ove datoteke se postavljaju kao zavisnosti odredišta Deval modula, i sistem za prevođenje ih traži i prevodi. Linije kojima se proširuje CMakeLists.txt modula, prikazane su na Slici 4.4.

```
13
14    add_executable(code_api_generator IMPORTED GLOBAL)
15
16    add_custom_command(
17    OUTPUT ${MODULE_PATH}/${INCLUDE_API} ${MODULE_PATH}/${SOURCE_API}
18    COMMAND ${CMAKE_BINARY_DIR}/code_api_generator ${GENERATOR_ARGUMENTS}
19    )
20    add_library(deval_adaptation STATIC ${SRC_FILES_SDK}
21                ${INCLUDE_API} ${SOURCE_API})
22
```

Slika 4.4 Proširenje Deval modula podrškom za Generator API.ja

- `add_executable(code_api_generator IMPORTED GLOBAL)`

  Opis: CMAKE funkcija kojom se dodaje izvršna datoteka.

  Parametri:

  - **IMPORTED:** Indikator.Označava da se izvršna datoteka uvlači spolja.
  - **GLOBAL:** Indikator. Označava da će izvršna datoteka biti dostupna svima u sistemu za prevođenje pod datim imenom.

- `add_custom_command(OUTPUT ${MODULE_PATH}/${INCLUDE_API} ${MODULE_PATH}/${SOURCE_API}COMMAND ${CMAKE_BINARY_DIR}/code_api_generator ${GENERATOR_ARGUMENTS})`

  Opis: CMAKE funkcija kojom se izvršava namenska komanda.

  Parametri:

- **OUTPUT:** Indikator.Označava da se nakon njega navode putanje do i imena izlaznih datoteka kada se izvrši namenska komanda.
- **MODULE_PATH:** Varijabla okruženja. Sadrži apsolutnu putanju do modula u sistemu datoteka.
- **INCLUDE_API:** Ime izlazne datoteke. Datoteka za umetanje, „.h“
- **SOURCE_API:** Ime izlazne datoteke. Izvorna datoteka, „.c“
- **COMMAND:** Indikator.Označava da se nakon njega navodi komanda koja se izvršava.
- **CMAKE_BINARY_DIR:** Varijabla okruženja, specifična za CMAKE (ref)dodati.
- **GENERATOR_ARGUMENTS:** Varijabla okruženja. U njoj se nalaze argumenti koji se prosleđuju Generatoru API-ja, uključujući i ime modela podataka. Podešava je konfiguraciona datoteka.

- `add_library(deval_adaptation STATIC ${SRC_FILES_SDK} ${INCLUDE_API} ${SOURCE_API})`

  Opis: CMAKE funkcija kojom se dodaje biblioteka modula i njene zavisnosti.

  Parametri:

  - **deval_adaptation:** Ime biblioteke modula.
  - **STATIC:** Indikator. Označava da će biblioteka modula biti statička.
  - **INCLUDE_API:** Ime izlazne datoteke. Datoteka za umetanje, „.h“
  - **SOURCE_API:** Ime izlazne datoteke. Izvorna datoteka, „.c“
  - **SRC_FILES_SDK:** Varijabla okruženja. Sadrži ostale datoteke od kojih zavisi biblioteka.

4.9 Pravljenje dokumentacije

Dokumentacija za korišćenje MAPI datoteka koje Generator API-ja daje kao izlaz se pravi pomoću alata *Doxygen*. Njemu se na ulaz proslede „.h“ MAPI datoteke modela, iz kojih parsiranjem komentara, dobija potrebne informacije za dokumentaciju. *Doxygen* omogućava pravljenje dokumentacije u jednoj od ekstenzija: .html ili .pdf.

5. Ispitivanje i rezultati

5.1 Platforma za ispitivanje opterećenja

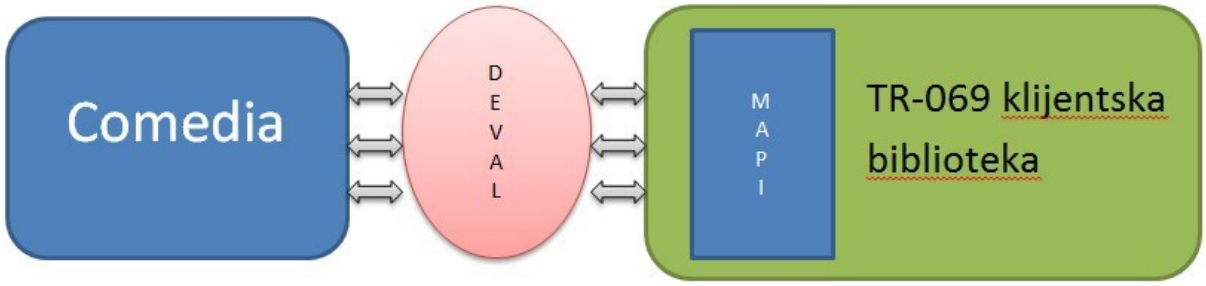
Digitalni televizijski prijemnik na kom je ispitivano predloženo rešenje za dinamičko prevođenje TR-069 klijentske biblioteke je uređaj RK-1000, proizveden na Institutu za sisteme zasnovane na računaru koji ima sledeće karakteristike fizičke arhitekture:

- Procesor –SH4 (eng. SuperH) familija, čip iDecode7105, takta od 450MHz
- Radna Memorija – 256MB RAM
- Spoljna memorija – NAND fleš čip od 128MB i NOR od 4MB (smešten na isti čip kao i procesor i radna memorija)
- Mrežni sprežni modul (tjuner i demodulator), antenski ulaz
- HDMI
- CVBS
- Zvučni izlaz (eng. audio)
- SPDIF
- USB 2.0
- Mrežni interfejs (eng. Ethernet)

Operativni sistem koji naleže na predstavljeni set-top-boks je Linuks (eng. Linux) [5]. Ovaj operativni sistem je distribucija firme ST Mikroelektroniks (eng. ST Microelectronics), posebno modifikovan da odgovara arhitekturi set-top-boksa, sa odgovarajućim rukovaocima.

Programska podrška uređaja RK-1000 je Comedia. Potrebno je da Comedia i TR-069 klijentska biblioteka međusobno komuniciraju i razmenjuju podatke.. Između njih se nalazi prilagodni sloj - Sloj za apstrakciju uređaja (eng. DEVAL, Device Abstraction Layer). DEVAL služi da podatke koji se nalaze u modelu podataka i koji su za čitanje (npr. kvalitet signala), potraži u programskoj podršci Comedia, a podatke koji su za pisanje (npr. menjanje servisa sa

ACS) izmeni unutar Comedia-e. DEVAL se oslanja na MAPI prilikom prosleđivanja parametara iz Comedie u TR-069 klijentsku biblioteku. Na Slici 5.1 je prikazana veza i moduli između TR-069 klijenta i Comedia-e.



Slika 5.1 Veza TR-069 klijenta sa programskom podrškom digitalnog televizijskog prijemnika (Comedia)

5.2 Ispitni slučajevi

Ispitivanje sistema je vršeno u više iteracija. Većina ispitnih slučajeva vršena je na ciljnoj platformi opisanoj u Poglavlju 5.1. Na ciljnoj platformi ispitivano je opterećenje sistema sa aplikacijom koja koristi TR-069 biblioteku:

- koja ima MAPI napravljen Generatorom API-ja;
- koja ima MAPI pravljen bez Generatora API-ja.

Dobijene vrednosti u oba slučaja su poređene i opisani su dobijeni rezultati. Ispitni slučajevi su:

- Ponašanje prilikom pogrešno prosleđenih argumenata Generatoru API-ja;
- Vreme izvršavanja Generatora API-ja za različite modele podataka;
- Veličina osnove (eng. *footprint*) biblioteke i izvšne aplikacije u koju je ugrađena TR-069 klijentska biblioteka, prevedeni za ciljnu platformu (digitalni televizijski prijemnik);
- Veličina koda potrebnog za ostvarivanje iste funkcionalnosti TR-069 klijentske biblioteke koja je postojala pre korišćenja Generatora API-ja;
- Vreme integracije TR-069 klijentske biblioteke;
- Ispitivanje funkcionalnosti TR-069 klijentske biblioteke;
- Opterećenje sistema ciljne platforme.

5.2.1 Pogrešni argumenti

Prilikom verifikacije Generatora API.ja, potrebno je proveriti reagovanje aplikacije na pogrešno unesene parametre i u to potpadaju:

- poziv bez prosleđenih argumenata;
- izostavljanje nekog od obaveznih argumenata;
- prosleđenih argument je nepodržan ili pogrešno formatiran.

U svim ovim slučajevima, aplikacija treba da obezbedi opis greške i da ispiše uputstvo korisniku. Na Slici 5.2 je prikazan slučaj kada se Generatoru API-ja na ulaz prosledi model podataka koji ne postoji na zadatoj korenskoj putanji.

```
./code_api_generator -l c -n tr-135-0-1.xml -r /home/mveljko/qt_projs/code_generator/res/ -m
output directory is: ./
[2014-06-19 03:27:01.172654] [0x000007f5dcd98b74] [error] TRParser: File tr-135-0-1.xml doesn't exist in provided root directory, please check it's existence
Usage: (null) options [ inputfile ... ]


-h --help Display this usage information.
-o --output path Write output to specified path. Optional
-p --profiles fileName Name of the file containing the names of desired profiles for specified datamodel. Must be in the same dir as binary. Optional
-r --root path Set the root path to directory containing templates, data models and xml schema files, all in separate directories. Optional
Default names are datamodels and xmlSchemas. Templates differ from desired language.
-d --datamodels folderName Specify the name of the folder in root that containes data models. Default name: datamodels. Optional
-x --xmlSchemas folderName Specify the name of the folder in root that containes xml schema files. Default name: xmlSchemas. Optional
-n --nameOfDataModel fileName Specify the name of the model you want to generate code for.
Example tr-xxx-m-n.xmlxxx -- data model number; m -- major version; n -- minor version. Must be in this format!!! Required
-l --language languageType Specify for which language you want to generate the code.
c -- for C language; java -- for JAVA language; jni -- for JNI code. Required
-b --builder manName Specify the name of the manufacturer for which you are generating the code (to be displayed in documentation). Optional
-q --prerequisite fileName Specify the data model that needs to be instantiated prior to parsing your desired data model. Default for cwmp is TR 106. Optional
-g --logLevel num Specify log level from 0 to 6. trace, debug, info, warning, error, fatal. Optional
-e --DRELogLevel Enable debug trace from Dre parser to trace errors. Optional
-m --multiclient Generate files for muliticlient instance of libcpe
```

Slika 5.2 Reagovanje Generator API-ja na prosleđeni nepostojeći model podataka

5.2.2 Vreme izvršavanja Generatora API-ja

Vreme potrebno za izvršavanje Generatora API-ja za različite modele podataka, prikazano je u Tabeli 5.1. Iz prikazane tabele se vidi da vreme izvršavanja Generatora API-ja odnosno vreme za koje Generator API-ja napravi datoteke sa MAPI-jem je u direktnoj vezi sa veličinom modela podataka za koji se MAPI pravi. Svako ispitivanje je vršeno nekoliko puta i dobijena vrednost predstavlja aritmetičku sredinu dobijenih rezultata.

Modeli podataka	Vreme pravljenja datoteka(s)	Veličina datoteka [KB]
TR-135-1-3.xml	6.8	360,5
TR-106-1-2.xml	1.8	104,6
TR-450-0-0.xml	0.3	10
TR-106-1-1.xml	0.5	87,1

Tabela 5.1 Vremena potrebna za pravljenje MAPI-ja u odnosu na veličinu modela podataka

Vreme koje bi bilo potrebno da se podrže svi parametri iz TR-135 modela podataka korišćenjem Generatora API-ja je 6.8 sekundi. Da bi se postigla ista podrška ručnim pisanjem potrebnog broja funkcija za sve parametre, potrebno je nekoliko meseci intenzivnog rada sa ispravljanjem grešaka, koje su sastavni deo ručno pisanog koda. Isto tako, Generatorom API-ja je jednostavno priširiti ili smanjiti broj podržanih parametara jednostavnim preciziranjem željenih profila iz modela podataka. Generator API-ja znatno smanjuje vreme integracije TR-069 klijentske biblioteke u ciljni uređaj.

5.2.3 Veličina osnove i veličina koda

Veličina osnove (eng. *footprint*) je veličina izvršne datoteke odnosno biblioteke. Ispitni slučaj koristi ispitnu aplikaciju koja statički povezuje i uključuje TR-069 klijentsku biblioteku.Postoje dva ispitna slučaja:

- Aplikacija se povezuje sa bibliotekom koja ima ručno pisan MAPI – slučaj A;
- Aplikacija se povezuje sa bibliotekom koja ima MAPI iz Generatora API-ja – slučaj B.

U oba slučaja podržana je jednaka funkcionalnost manipulacije TR-106 repozitorijuma u TR-069 klijentskoj biblioteci. Alat za proveru veličine osnove je aplikacija *size* koja se nalazi u većini Linux distribucija. Ona raščlanjuje datoteku u tri sekcije:

- **.bss**: Sekcija koja sadrži statičke varijable koje su postavljene na nulu (bss sekcija);
- **.text**: Kodna sekcija koja sadrži sve instrukcije izvršne datoteke odnosno biblioteke (text sekcija);
- **.data**: Sadrži globalne i statičke varijable koje je inicijalizovao softverski inženjer (data sekcija).

Sledeća Tabela 5.2 prikazuje veličine osnova za oba slučaja

Generator API-ja	.text[B]	.bss[B]	.data[B]	Ukupna veličina
Ne	334454	3200	1800	339454
Da	302281	2856	1792	306929

Tabela 5.2 Veličina osnove ispitne aplikacije

Iako je očekivano da rezultati budu veoma slični za slučaj A i B, to nije slučaj, jer bi implementacije MAPI funkcija trebalo da budu identične, i ručno pisane i automatski napravljene. To nije slučaj jer u trenutku ispitivanja, ručno pisani kod nije bio optimizovan. U ispitnu aplikaciju se ne povezuje cela biblioteka, već samo deo koda koji se koristi i taj deo koda pravi razliku u veličinama osnova. Svaka API funkcija u svojoj definiciji sa Generatorom API-ja

ima najviše četiri linije koda, dok su ručno pisane funkcije imale najmanje osam linija, a često i više.

Pri realizaciji rešenja za Generator API-ja, korišćen je ručno pisan MAPI, koji nije bio optimizovan, kao referentni. Puna podrška za svaki parametar iz modela (na primer TR-135) iziskuje da datoteka MAPI-ja sadrži i do nekoliko hiljada linija programskog koda, a time se povećava veličina osnove za TR-069 klijentsku biblioteku. Na veličinu osnove bitno utiče veličina datoteke MAPI-ja i stoga je potrebno obezbediti da veličina MAPI datoteka bude što manja. To se postiglo smanjivanjem tela tipa funkcija: *Postavi*, *Dobavi*, *Dodaj*, *Ukloni*, *Prijavi*, *Odjavi* na minimum.

Rast veličine osnove biblioteke u direktnoj je vezi sa veličinom modela podataka za uređaj. U Tabeli 5.3 prikazan je odnos veličine osnove TR-069 klijentske biblioteke sa brojem linija koje sadrže njene izvorne datoteke. U prvom ispitnom slučaju, podržan je podskup parametara iz modela podataka TR-106 od 47 API finkcija u MAPI datotekama. Puna podrška za TR-106 model podataka podrazumeva 378 funkcija u MAPI datotekama.

Generator API-ja	Veličina [KB]	Broj podržanih funkcija	Izvorni kod biblioteke [linije]
Da	723,3	47	32055
Da	916,6	378	34780

Tabela 5.3 Odnos veličine osnove i veličine izvornog koda

Iz prikazane tabele se jasno uočava relacija veličine osnove i broja linija koda. Razlika od samo par dodatnih linija koda po svakoj od 378 API funkcija, može dovesti do značajnog povećanja veličine osnove TR-069 klijentske biblioteke, stoga funkcije koje proizvodi Generator API-ja, moraju biti optimizovane.

5.2.4 Vreme integracije TR-069 klijentske biblioteke

Ispitivanje sprovedeno za ovaj slučaj, temelji se na projektu koji je zahtevao prilagođavanje programske podrške *Comedia*-ja na digitalni televizijski prijemnik. Deo ovog projekta je podrazumevao je integraciju TR-069 klijentske biblioteke, opisane u Poglavlju 2.2, unutar programske podrške digitalnog televizijskog prijemnika za prijem zemaljskog i IP televizijskog signala. Proizvođač je obezbedio svoj model podataka, definisan prema TR-106. Za ovaj deo projekta izdvojeni su sledeći podaci:

- 616 čovek-dana je bilo potrebno za ukupnu realizaciju integracije TR-069 klijentske biblioteke;
- Oko 150 čovek-dana je utrošeno na pravljenje MAPI-ja (četiri čoveka što je oko mesec i po dana u realnom vremenu);

- MAPI se sastoji od oko 230 funkcija za ceo model podataka;
- Ostatak vremena je utrošen na pretragu parametara u programskoj podršci digitalnog televizijskog prijemnika i njeno prilagođavanje za dobavljanje parametara.

Problemi koji su uzrokovali utrošak vremena prilikom pravljenja MAPI-ja:

- Model podataka se menjao na nedeljnom nivou, i već napisane API funkcije su zahtevale prilagođavanje
- Greške prilikom pisanja API funkcija i njihovo lociranje i uklanjanje;
- Pisanje ovih API funkcija i stalna pretraga modela podataka zarad dobavljanja informacija potrebnih da bi se realizovale funkcije (tip parametra/objekta, određivanje imena funkcije, opis parametra/objekta)

Iz Tabele 5.4 se vidi procentualni utrošak vremena na projektu koji je zauzela realizacija MAPI-ja.

Čovek-dan [h]	Ukupno čovek-dana	Pravljenje MAPI-ja	Procenat čovek-dana utrošenog na MAPI [%]
8	616	150	24

Tabela 5.4 Utrošak čovek-dana prilikom realizacije MAPI-ja

Sa Generatorom API-ja moguće je napraviti identičan MAPI za nekoliko sekundi. Uzimajući u obzir da je 150 čovek-dana mnogo veće od nekoliko sekundi, možemo reći da je utrošeno vreme na pravljenje MAPI-ja nula. Ovime se dolazi do zaključka da se korišćenjem Generatora API-ja za pravljenje MAPI-ja postiže ušteda od 24% čovek dana, i kompletna integracija TR-069 klijenta u programsku podršku bi trajala znatno manje.

Svi problemi sa kojima smo se susreli prilikom pravljenja MAPI-ja na ovom projektu, uklonjeni su upotrebom Generatora:

- Da bi se podržale izmene u modelu podataka, potrebno je samo pokrenuti Generator API-ja sa novim modelom i nov MAPI je podržan;
- MAPI koji pravi Generator API-ja je ispitan (vidi Poglavlje 5.2.5), bez greške i sa odgovarajućom dokumentacijom;
- MAPI funkcije su napravljenje gotovo istog trenutka sa svim podacima iz modela podataka.

Takođe, postoje funkcionalnosti koje nije moguće ostvariti samo jednom MAPI funkcijom, kao što je dodavanje servisa. Prilikom dodavanja servisa, potrebno je postaviti nekoliko parametara vezanih za taj servis. Ova funkcionalnost se postiže kombinovanjem više MAPI

funkcija. Ove funkcionalnosti se nazivaju *Proširenja*. Prilikom rada na ovom projektu, bilo je potrebno realizovati nekoliko *Proširenja*. U Tabeli 5.5 je prikazan odnos linija koda za podršku *Proširenja* Dodaj servis:

- Kada se koristi MAPI iz Generatora API-ja;
- Kada se koristi ručno pisani MAPI.

Generator API-ja	Proširenje Dodaj servis[linije koda]	Vreme pisanja Proširenja [čovek-dan]
Ne	817	5
Da	211	1

Tabela 5.5 Veličina Proširenja Dodaj servis

Korišćenjem MAPI-ja iz Generatora API-ja, ostvarena je ušteda i na veličini osnove (skoro četiri puta manji broj linija koda) i u vremenu potrebnom da se podrži *Proširenje*. Tačan broj linija koda utvrđen je uz pomoć Linux aplikacije *cloc* [15]. Budući da u početku pravljenja MAPI-ja, nije postojala podrška za sve funkcije iz modela, pri pisanju *Proširenja* Dodaj servis, pristupalo se direktno Modul za upravljanje repozitorijumima. Ovime se izgubilo na uštedi u linijama koda, jer zasebne celine nisu izdvojene kao posebne MAPI funkcije.

5.2.5 Ispitivanje funkcionalnosti TR-069 biblioteke sa MAPI-jem iz Generatora API-ja

Funkcionalnost TR-069 biblioteke sa API-jem iz Generatora API-ja je ispitana CUnit ispitnim slučajevima, jer je TR-069 klijentska biblioteka realizovana u programskom jeziku C. Sproveden je određen broj ispitnih slučajeva nad stablom TR-135 modela podataka. Ispitni slučajevi su podeljeni na pet skupova ispitnih slučajeva:

- Dodavanje i uklanjanje servisa (televizijskog kanala) – 30 ispitnih slučajeva;
- Provera RTP (eng. *Real Time Protocol*) podataka – 45 ispitnih slučajeva;
- Provera podataka MPEG-2 prenosnog toka (eng. *transport stream*) – 25 ispitnih slučajeva;
- Provera podataka video dekodera – 57 ispitnih slučajeva;
- Provera pretrage kanala i uklanjanje – 223 ispitna slučaja.

Rezultati sa Slike 5.3 pokazuju da je ispitivanje prošlo bez grešaka (svih 5 ispitnih skupova) i da je API iz Generatora API.ja funkcionalan i ispravan.

```
--Run Summary: Type      Total     Ran  Passed  Failed
               suites        1       1     n/a       0
               tests         5       5       5       0
               asserts     380     380     380       0

************************************************************
                 LIBCPE MODULE TESTING
************************************************************
```

Slika 5.3 CUnit rezultati testova TR-069 klijentske biblioteke sa MAPI-jem iz Generatora API-ja

5.2.6 Opterećenje sistema ciljne platforme

TR-069 klijentska biblioteka se ugrađuje u namenske (eng. *embedded*) uređaje, u ovom slučaju digitalni televizijski prijemnik. Ispitivanje opterećenja ciljne platforme je nužno sprovesti, da bi se utvrdilo da li se korišćenjem MAPI-ja iz Generatora API-ja unosi dodatno opterećenje na sistem ili ne. Namenski uređaji raspolažu ograničenim resursima, te se o tome mora voditi računa. Testiranjem na ciljnoj platformi, ustanovljeno je da se ne unosi dodatno opterećenje na ceo sistem. Opterećenje procesorske jedinice varira u rasponu od 0.1% do 0.3%. Slika 5.4 je isečak iz *top* procesa na ciljnoj platformi. Poslednja cifra označava opterećenje procesorske jedinice u procentima.

```
364   135 root     S    98828 45.2   1  0.3 ./cpetest
```

Slika 5.4 Opterećenje procesorske jedinice ispitnom aplikacijom sa API-jem napravljenim Generatorom API-ja

Opterećenje procesorske jedinice je zanemarljivo i ne narušava funkcionisanje celog sistema na ispitivanoj platformi opisanoj u Poglavlju 5.1.

5.3 Primer dokumentacije

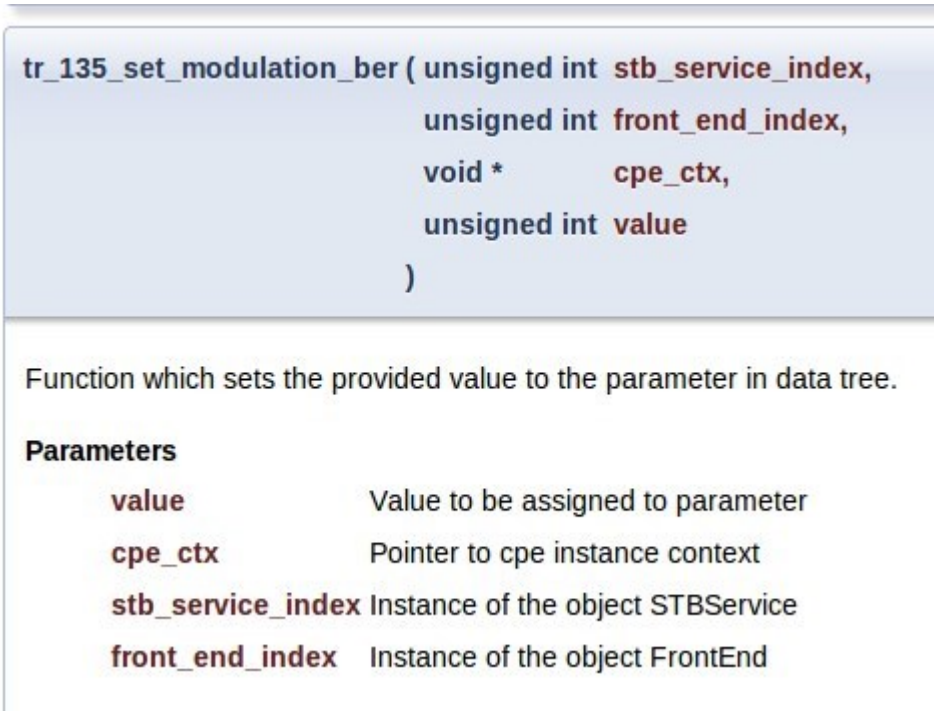
Dokumentacija koja se može napraviti na osnovu MAPI datoteka koje napravi Generator API-ja je jasna, precizna i razumljiva. Svaka API funkcija ima svoj opis funkcionalnosti. Skup funkcija koja pripada jednom parametru/objektu sadrži opis tog parametra/objekta. Moguća je jednostavna pretraga celog API- ja. Slika 5.5 prikazuje formatirani komentar za parametar *Modulation.BER* (iz TR-135 modela podataka) i jednu funkciju koja odgovara ovom parametru.

```
/**
 * @defgroup grouptr_135105 Device.STBService.%u.Components.FrontEnd.%u.DVBT.Modulation.BER
  Description from data model file:
  Bit Error Ratio before correction, expressed in multiples of 1e-6.
Parameter type: REPO_PARAM_TYPE_UNSIGNED
 * @{
 */
/**
    @fn tr_135_set_modulation_ber( unsigned int stb_service_index, unsigned int front_end_index, void* cpe_ctx, unsigned int value )
    @brief Function which sets the provided value to the parameter in data tree
@param value Value to be assigned to parameter
@param cpe_ctx Pointer to cpe instance context
@param stb_service_index Instance of the object STBService
@param front_end_index Instance of the object FrontEnd

*/
tr_error_t tr_135_set_modulation_ber( unsigned int stb_service_index, unsigned int front_end_index, void* cpe_ctx, unsigned int value );
```

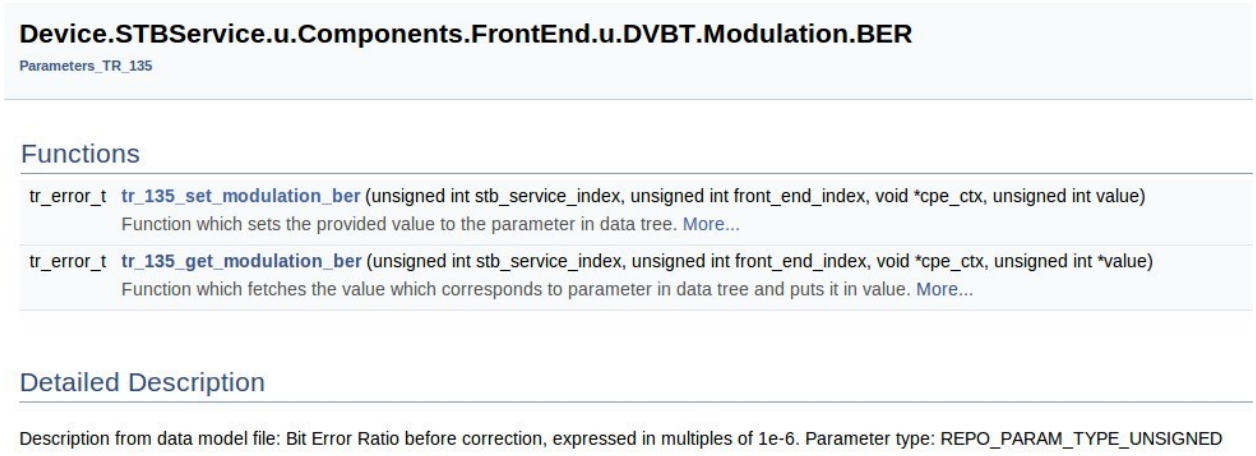
Slika 5.5 Jedna API funkcija koju pravi GeneratorAPI-ja za TR-135 model podataka

Na sledećoj Slici 5.6 prikazan je opis *Modulation.Ber* API funkcije koji se nalazi u dokumentaciji napravljenoj na bazi komentara u MAPI datoteci.



Slika 5.6 Opis API funkcije u dokumentaciji

Takođe, u dokumentaciji je moguće dobiti informaciju o prirodi samo parametra, šta njegova vrednost predstavlja i skup tipova funkcija koje podržava i koji je napravljen za njega u MAPI-ju, što se vidi sa Slike 5.7.



Device.STBService.u.Components.FrontEnd.u.DVBT.Modulation.BER

Parameters_TR_135

Functions

tr_error_t **tr_135_set_modulation_ber** (unsigned int stb_service_index, unsigned int front_end_index, void *cpe_ctx, unsigned int value)
Function which sets the provided value to the parameter in data tree. More...

tr_error_t **tr_135_get_modulation_ber** (unsigned int stb_service_index, unsigned int front_end_index, void *cpe_ctx, unsigned int *value)
Function which fetches the value which corresponds to parameter in data tree and puts it in value. More...

Detailed Description

Description from data model file: Bit Error Ratio before correction, expressed in multiples of 1e-6. Parameter type: REPO_PARAM_TYPE_UNSIGNED

Slika 5.7 Skup funkcija koje su vezane za parametar

6. Zaključak

U prethodnim poglavljima, u potpunosti je opisan problem, objašnjeno je predloženo rešenje i konkretna realizacija rešenja i ceo sistem je potkrepljen detaljnim ispitivanjima.

Glavni cilj predloženog sistema je ušteda inženjerskog vremena i ušteda na ugrađivanju TR-069 klijentske biblioteke u ciljni uređaj. Iz rezultata opisanih u Poglavlju 5, ušteda je u nekim slučajevima i preko mesec dana (vidi Poglavlje 5.2.4). Sistem za dinamičko prevođenje TR-069 klijentske biblioteke može da podrži novi model podataka koji prati pravila CWMP protokola, za nekoliko sekundi, sa odgovarajućom dokumentacijom.

Takođe, velika prednost predloženog sistema za dinamičko prevođenje TR-069 klijentske biblioteke, je podrška za profile iz modela podataka. Po želji, moguće je proširivati i smanjivati MAPI pomoću profila modela podataka za koji se MAPI pravi. Česte izmene na modelu podataka za uređaj ne uzrokuju dodatni posao prilagođavanja API-ja , s obtirom na postojanje Generatora sekundi. Ovo je posebno korisno ukoliko se model podataka neprestano menja. Isto tako, predloženo rešenje zbog koncepta šablona i mogućnosti prilagođenja modula Pretvarač parsiranih podataka i Maper šablona i šablonskih vrednosti je moguće primeniti na bilo koju implementaciju TR-069 klijentske biblioteke.

Dalje istraživanje na temu dinamičkog sistema za prevođenje TR-069 klijentske biblioteke obuhvata dalji razvoj postojećeg rešenja i prilagođavanje drugim programskim jezicima. Moguće je istražiti mogućnost dodavanja podrške za proširenja u toku rada TR-069 klijentske biblioteke.

7. Literatura

[1] TR-069 (CPE Wan Management Protocol), Broadband Forum, http://www.broadband-forum.org/cwmp

[2] TR-135 Data Model for a TR-069 Enabled STB, Broadband Forum, http://www.broadband-forum.org/technical/download/TR-135_Amendment-3.pdf

[3] TR-106 Data Model Template for TR-069-Enabled Devices, Broadband Forum, http://www.broadband-forum.org/technical/download/TR-106_Amendment-6.pdf

[4] Veljko Mihailović: Реализација механизама за ажурирање програмске подршке STB заснованих на TR-069 протоколу

[5] STLinux, distribucija Linux operativnog sistema za ugrađene uređaje, napravljena od strane kompanije ST Microelectronics http://www.stlinux.com/

[6] B. Adams, "Co-Evolution of Source Code and the Build System" Software Analysis and Intelligence Lab School of Computing Queen's University. Software Maintenance, 2009. ICSM 2009. September 2009

[7] M. de Jonge. "Build-level components" IEEE Trans. Softw. Eng., 31(7):588–600, 2005.

[8] A. Acuri, X. Yao, "Co-evolutionary Automatic Programming for Software Development" IEEE International Conference on Automated Software Engineering 2007, July, 2009.

[9] Freecwmp, http://freecwmp.org/

[10] libCWMP Embedded TR-069 Library http://www.avsystem.com/products/libcwmp/

[11] cpptempl – C++ mehanizam za šablone https://bitbucket.org/ginstrom/cpptemplate/wiki/Home

[12] Standardna šablonska biblioteka http://en.cppreference.com/w/cpp/container

[13] QtWebkit projekat https://trac.webkit.org/wiki/QtWebKit

[14] Doxygen http://www.stack.nl/~dimitri/doxygen/

[15] CloC http://cloc.sourceforge.net/

[16] GNU Make http://www.gnu.org/software/make/

[17] CMake http://www.cmake.org/

[18] QMake http://qt-project.org/doc/qt-4.8/qmake-manual.html