Sustav za daljinsko bežično ažuriranje vozila

Tkalčec, David

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij Automobilskog računarstva i komunikacija

SUSTAV ZA DALJINSKO BEŽIČNO AŽURIRANJE VOZILA

Diplomski rad

David Tkalčec

Osijek, 2019.

Sadržaj

1. UVOD

Sustavi za bežično ažuriranje sustava (engl. *Over-the-Air - OTA updates*) godinama se koriste u softverskoj industriji za ispravljanje grešaka te za različita poboljšanja na operacijskim sustavima i aplikacijama kod stolnih, prijenosnih i mobilnih uređaja. U automobilskoj industriji do unazad nekoliko godina bežična ažuriranja bila su prilično rijetko korištena. Softverski sustavi su vrlo osjetljivi na propuste u sigurnosti i pouzdanosti kada se podvrgnu različitim promjenama. Izvođenje ažuriranja preko bežičnih komunikacijskih kanala potencijalno predstavlja značajan sigurnosni problem. Do nedavno vozila su bila zatvoreni sustavi koji nisu dozvoljavali pristup uređajima unutar vozila koristeći bežične komunikacijske tehnologije [1]. U današnje vrijeme uočene su brojene prednosti koje donosi povezanost vozila na globalnu mrežu te se sve više koristi pojam povezanog vozila [2]. Povezana vozila imaju omogućen pristup internetskoj mreži te su opremljena bežičnom lokanom mrežom (engl. *Wireless Local Area Network*). Ova značajka omogućava vozilu da razmjenjuje podatke s različitim uređajima unutar i izvan vozila.

Današnja vozila sadrže velik broj elektroničkih upravljačkih jedinica (engl. *Electronic Control Units – ECUs*) koje upravljaju različitim sustavima u vozilu te se očekuje da će u budućnosti broj ECU-ova po vozilu rasti. Trenutno moderna vozila sadrže više od 70 ECU-ova te više od 100 miliona linija koda, stoga je iznimno važno da navedeni programski kod bude ažuriran i sa što manje sigurnosnih propusta. ECU-ovi nadziru jako velik broj funkcionalnosti vozila kao što su: zaključavanje, klima, sustavi karoserije, napredni sustavi za sigurnost i izbjegavanje sudara, sustavi za nadzor tlaka u gumama i drugi [3]. Na svakom ECU se izvršava specijalizirani i neovisni *firmver*. Kada se identificiraju pogreške ili doda nova funkcionalnost nastaje nova verzija *firmvera* koju je potrebno instalirati na uređaj. Do nedavno ažuriranja *firmvera* provodili su serviseri u ovlaštenim servisima, što je značilo da vlasnici moraju uvijek putovati do servisa kada se jave određeni problemi u vozilu [4]. Ovakvo rješenje predstavlja velik gubitak vremena za vlasnike vozila te novčani gubitak za proizvođače koji moraju plaćati ovlaštene servise. Korištenjem bežičnih veza proces ažuriranja se značajno može ubrzati. U slučajevima kada je potrebno instalirati važna ažuriranja za sigurnosno kritične sustave, bežična mreža omogućava vrlo brzo slanje ažuriranja na velik broj vozila te tako može doprinijeti krajnjoj sigurnosti.

Iako bežično ažuriranje sustava donosi brojne prednosti, ovakvim načinom ažuriranja javljaju se nove ulazne točke za hakere koji imaju namjeru neovlašteno mijenjati softver u vozilu ili

neovlašteno dohvaćati i/ili mijenjati podatkovne pakete koji se šalju prema vozilu. Osim otvaranja novih točaka napada za hakere, u procesu ažuriranja više ne postoji obučena ovlaštena osoba koja nadzire proces te potvrđuje uspješnu instalaciju ovlaštenog softvera te osigurava da nema sigurnosnih opasnosti koje mogu proizaći iz ažuriranja. Zbog navedenih potencijalnih opasnosti vrlo je važno implementirati sigurnosne mehanizme koji će omogućiti provjeru izvornosti i integriteta za pakete preuzete preko bežične mreže. Nadalje, sustav ažuriranja treba biti robustan na greške pri ažuriranju.

Zadatak ovoga rada je proučiti metode za ažuriranje ugradbenih računalnih sustava sa svrhom ažuriranja vozila preko bežične veze te razviti prototip sustava za ažuriranje vozila preko bežične veze koji omogućava učinkovit prijenos velikih količina podataka, preuzimanje novih paketa, provjeru njihove izvornosti i integriteta te je robustan na greške pri ažuriranju

U sljedećem poglavlju ovoga rada biti će opisani dostupni mehanizmi za ažuriranje sustava te najkorištenije metode koje implementiraju neke od tih mehanizama. Zatim, u trećem poglavlju biti će opisani alati potrebni za izradu praktičnog dijela rada. U četvrtom poglavlju biti će detaljno opisano kompletno rješenje bežičnog ažuriranja uređaja koji upravlja informacijsko-zabavnim sustavom vozila (engl. *In-vehicle infotainment*).

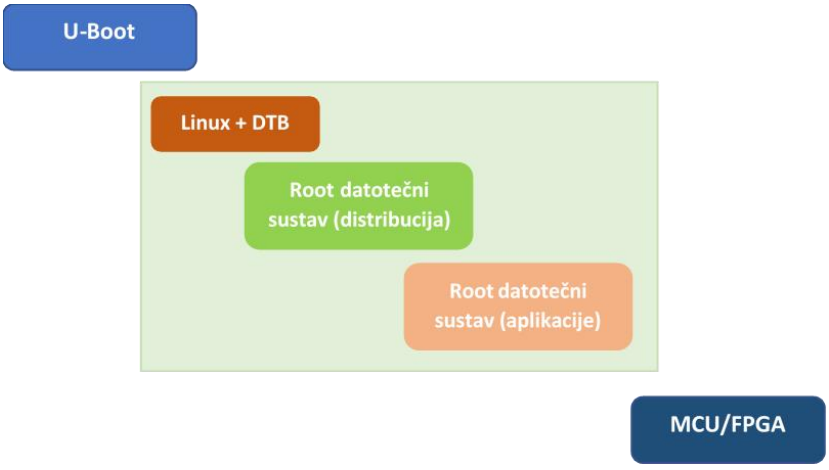
2. AŽURIRANJE LINUX OPERACIJSKOG SUSTAVA

U ovome poglavlju biti će navedeno nekoliko najkorištenijih načina ažuriranja Linux operacijskog sustava te njihove karakteristike. Na temelju usporedbe biti će odlučeno koji od njih je najpogodniji za primjenu u informacijsko-zabavnom sustavu vozila. Nakon toga će biti dana usporedba najpoznatijih mehanizama koji implementiraju prethodno odabrani način ažuriranja te će biti odlučeno koji od njih je najpogodniji za realizaciju ovoga rada.

2.1. Načini ažuriranja i osnovne karakteristike

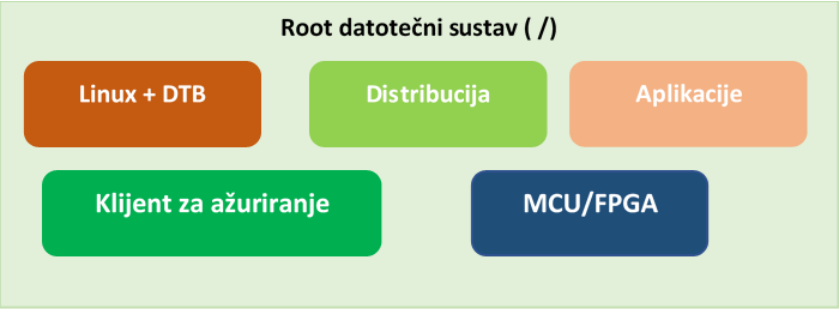
Linux operacijski sustav, vrlo je pogodan za informacijsko-zabavni sustav vozila zbog iznimne mogućnosti prilagodbe. Većina proizvođača prilagođava Linux sustav prema specifičnim potrebama za uređaje u vozilu. U većini slučajeva sustavi temeljeni na Linux operacijskom sustavu sastoje se od sljedećih elemenata:

1. *bootloader*
2. jezgra i DT (engl. *Device tree*) datoteka
3. *root* datotečni sustav
4. ostali datotečni sustavi koji se naknadno ugrađuju
5. podaci proizvođača
6. softver za specifičnu primjenu, primjerice *firmver* koji je potrebno instalirati na povezane mikroupravljače



Slika 2.1. Elementi Linux operacijskog sustava

Prilikom ažuriranja u većini slučajeva potrebno je ažurirati jezgru (engl. *kernel*) i *root* datotečni sustav (Slika 2.2.), čuvajući postojeće korisničke podatke. U nekim slučajevima je potrebno i ažuriranje *bootloadera*, ali ovaj proces je vrlo rizičan jer je moguće da problemi prilikom ažuriranja uzrokuju kvar samog uređaja. Kada dođe do kvara, popravak je moguć u nekim slučajevima te se većinom uređaj mora slati proizvođaču. Postoje brojni koncepti za ažuriranje softvera [5], neki od najčešće korištenih su: ažuriranje pomoću *bootloadera*, ažuriranje preko paket menadžera te korištenje aplikacije koja izvršava proces ažuriranja.



Slika 2.2. *Root* datotečni sustav

2.1.1. Ažuriranje pomoću *bootloadera*

Glavna uloga *bootloadera* je pokretanje jezgre. *Bootloader* posjeduje vlastitu ljusku (engl. *shell*) kojom se najčešće upravlja preko serijske komunikacije. U mogućnosti je pokretati skripte što znači da je moguće implementirati nekakav mehanizam ažuriranja. Prednost korištenja ovakvog načina ažuriranja je da je proces relativno jednostavniji. U slučaju kada se ažuriranje vrši pomoću zasebne aplikacije potrebno je da ona posjeduje vlastiti jezgru i *root* datotečni sustav, dok kod ažuriranja pomoću *bootloadera* ovo nije slučaj. Međutim, ovakav način ažuriranja ima nekoliko vrlo bitnih nedostataka:

1. *bootloader* ima ograničen pristup periferiji – podrška za periferiju se najčešće dodaje u jezgru te na taj način aplikacije mogu pristupati istoj. Stoga nema uvijek smisla ulagati dvostruki trud kako bi se pogonski programi (engl. *drivers*) omogućili u *bootloaderu*.
2. pogonski programi unutar *bootloadera* se ne ažuriraju – pogonski programi se najčešće prenose iz jezgre u *bootloader*. Promjene u jezgri je potrebno uvijek sinkronizirati s *bootloaderom* kako ne bi došlo do grešaka u radu sustava, Popravljanje takvih grešaka može biti vrlo složeno.
3. ograničen je broj podržanih datotečnih sustava – proces dodavanja podrške za novi datotečni sustav u *bootloader* je vrlo složen.

4. ograničen je pristup mreži – većinom ažuriranje je moguće samo putem UDP protokola
5. vrlo je složeno napraviti sučelje za proces ažuriranja

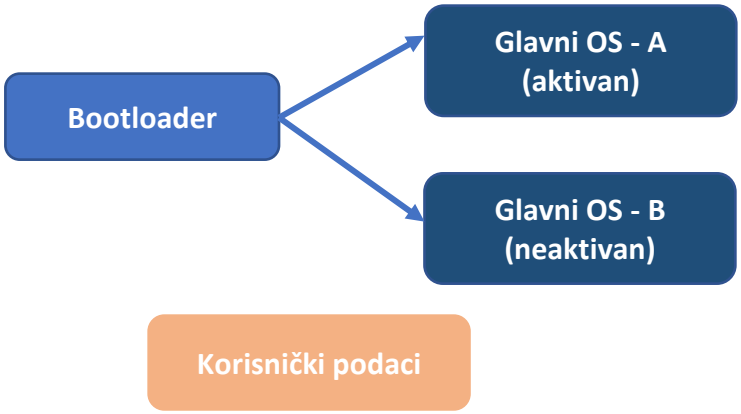
2.1.2. Ažuriranje koristeći paket menadžer

Ažuriranje pomoću paket menadžera najčešće se koristi kod poznatih Linux distribucija, ali nije pogodno za ugradbene računalne sustave. Razlog tome je taj da je ugradbeni sustav testiran vrlo dobro za specifičan softver. Korištenje paket menadžera može dovesti do problema jer softver više nije jedinstvena cjelina (engl. *atomic*), već je podijeljen u listu paketa. Razdvajanje u pakete može prouzrokovati brojne probleme za testere koji moraju locirati uzroke grešaka. Kod ugradbenih sustava gotovo se uvijek zahtjeva da ažuriranje bude *atomic*, odnosno da nije podijeljeno u pakete.

2.1.3. Korištenje aplikacije koja izvršava proces ažuriranja

Sljedeća mogućnost je korištenje aplikacije koja upravlja procesom ažuriranja. Takva aplikacija treba implementirati mehanizme provjere ispravnosti paketa i instalacije softvera te može koristiti servise operacijskog sustava. Aplikacija također treba biti u mogućnosti detektirati može li softver biti instaliran na uređaj te provjeriti da li je softver izdala autorizirana strana. Unutar takve aplikacije moguće je dodavati velik broj novih značajki. Ovisno o dostupnim resursima uređaja postoje dvije osnovne metode koje se mogu koristiti kod instalacije ažuriranja pomoću aplikacije: korištenje jedne kopije sustava u kombinaciji s particijom za oporavak te korištenje dvije kopije s implementiranom metodom za oporavak. Korištenje dvije kopije najčešće se provodi ukoliko postoji dovoljno memorije za njihovu pohranu. U tom slučaju moguće je garantirati da će uvijek postojati jedna ispravna kopija, čak i u slučaju kada dođe do iznenadnog prekida ažuriranja ili greške tijekom ažuriranja. Svaka kopija treba sadržavati jezgru i *root* datotečni sustav te svaku komponentu koja će se ažurirati. Zadatak aplikacije je instalirati ažuriranje na kopiju koja se trenutno ne koristi pri čemu trenutno pokrenuta kopija treba ostati netaknuta (Slika. 2.3.). Ovakva metoda zahtjeva komunikaciju s *bootloaderom* jer *bootloader* mora odlučiti koja kopija će biti pokretana. U *bootloaderu* mora postojati mogućnost promjene aktivne kopije te nakon ponovnog pokretanja sustava *bootloader* odlučuje koja kopija treba biti pokrenuta. Nedostatak ove metode je potreba za dvostruko više memorije za pohranu, ali ažuriranje sustava je uvijek sigurno čak i u slučaju iznenadnog prekida ažuriranja. Ukoliko sustav

posjeduje dovoljno memorije preporučeno je korištenje koncepta dvije kopije kako bi se osigurala mogućnost povratka na staru verziju u slučaju greške tijekom ažuriranja.



Slika 2.3. Korištenje dvije kopije za ažuriranje sustava

S druge strane jedna kopija s particijom za oporavak najčešće se koristi u slučaju kada su memorijski resursi vrlo ograničeni te ne dopuštaju korištenje dvije kopije. Ova metoda se temelji na postojanju operacijskog sustava za oporavak koji je najčešće minimalna Linux jezgra i minimalni *root* datotečni sustav koji sadrži samo aplikaciju za ažuriranje i njezine biblioteke. Koncept korištenja ove metode prikazan je na slici 2.4. Sustav za oporavak je značajno manji od jedne kopije sustava. Usporedba karakteristika navedenih dviju metoda prikazana je u tablici 2.1.

Slika 2.4. Korištenje jedne kopije za ažuriranje sustava

Ažuriranje metodom jedne kopije najčešće se sastoji od sljedećih koraka:

1. Uređaj provjerava postoji li dostupno ažuriranje slanjem upita na poslužitelj
2. Podaci za ažuriranje se preuzimaju na podatkovnu particiju i korisniku se nudi izbornik za instalaciju softvera

3. Pokreće se operacijski sustav za oporavak
4. Dohvaćaju se podaci za ažuriranje s podatkovne particije i pokreće se instalacija ažuriranja
5. Pokreće se ažurirani operacijski sustav

Tablica 2.1. Karakteristike metoda ažuriranja

Jedna kopija + sustav za oporavak	Dvije kopije
Uređaj se ne može koristiti tijekom ažuriranja	Uređaj se može koristiti tijekom ažuriranja
Potreban dodatni prostor za pohranu podataka za ažuriranje	Moguće je *streamati* ažuriranje na neaktivnu kopiju
Korisnik mora pokrenuti instalaciju	Ne mora se obavijestiti korisnika o ažuriranju (iako to nije čest slučaj)
Ukoliko je instalacija neispravna ili iznenadno prekinuta operacijski sustav se neće pokrenuti u normalnom načinu	Nema dodatnog čekanja da se izvrši instalacija
	Ukoliko ažurirani softver ne radi na očekivan način moguće je vratiti staru kopiju i ponovno pokušati ažurirati sustav

Na temelju opisanih načina ažuriranja može se zaključiti da za ugradbene računalne sustave najviše odgovara ažuriranje korištenjem aplikacije. Postoji nekoliko dostupnih rješenja koja implementiraju ovakav koncept. Odabir odgovarajuće metode ovisi o specifičnim zahtjevima ugradbenog sustava. U sljedećem potpoglavlju su uspoređena tri najkorištenija mehanizma koje je moguće koristiti za ažuriranje operacijskog sustava u vozilu.

2.2. Dostupni mehanizmi ažuriranja operacijskog sustava

U ovome potpoglavlju biti će opisano i uspoređeno tri različita mehanizma za ažuriranje ugradbenog sustava temeljenog na Linux operacijskom sustavu: SWUpdate, Mender i RAUC. Svaki od njih nudi specifične značajke na temelju kojih je potrebno odlučiti koji bi najbolje odgovarao za implementaciju na IVI operacijskom sustavu vozila.

U ovome poglavlju biti će navedeni samo mehanizmi koji vrše ažuriranje na razini blokova. Mehanizmi koji vrše ažuriranja na temelju datoteka u većini slučajeva nisu pogodni za ažuriranje ugradbenih sustava zbog veće mogućnosti pojave grešaka u radu sustava te zatim teže identifikacije istih. U Tablici 2.2. navedeni su parametri na temelju kojih će se vršiti usporedba te opisi tih parametara.

Tablica 2.2. Parametri i opisi parametara za usporedbu [6]

Parametar	Opis
Tip	*Block-based* mehanizmi direktno mijenjaju blokove na particijama koje ažuriraju. To znači da particije moraju biti jednake za sve uređaje te uređaji moraju koristiti jednake veličine particija. *File-based* mehanizmi mijenjaju datoteke i direktorije u datotečnom sustavu zbog čega je moguće koristiti ista ažuriranja za više uređaja s različitim particijama.
Struktura diska	Zahtjevi vezani za *bootloader*, broj i vrstu particija. Mehanizmi koji su definirani kao fleksibilni ne daju nikakve pretpostavke o sustavu ili daju samo nekoliko osnovnih pretpostavki. Zbog toga je potreban dodatan rad kod integracije za specifičan sustav.
Rootfs	Particije koje sadrže operacijski sustav. Mogu biti samo za čitanje (kod *block-based* mehanizama) ili mogu biti za čitanje i pisanje. Neki mehanizmi mogu podržavati ažuriranje samo dijela operacijskog sustava.
Ažuriranje iz	Opisuje odakle mehanizam za ažuriranje dohvaća podatke za ažuriranje
Ažuriranje čega	Opisuje koje dijelove krajnjeg sustava mehanizam ažurira
Stabilnost koda	Temeljeno na osobnom iskustvu, vremenu korištenja koda, sigurnosnim zapisima u postojećim implementacijama
Yocto integracija	Podržava li mehanizam integraciju s Yocto Project-om
Resursni zahtjevi na poslužitelju	Utječu na vrijeme izgradnje i kapacitet pohrane podataka.
Resursni zahtjevi na klijentu	Potreban dodatni privremeni memorijski prostor, jačina procesora, brzina mreže i drugi.
Otpornost na greške	Opisuje kako se mehanizam nosi s potencijalnim problemima.
Složenost	Opisuje koliko je složeno ispravno implementirati i koristiti mehanizam
Vrijeme instalacije	Koliko dugo će biti onemogućeno normalno korištenje uređaja.
Sigurnost	Podrška za različite sigurnosne mehanizme te zaštita samog mehanizma za ažuriranje.

2.2.1. Mender

Mender je *open source* aplikacija koja se sastoji od klijenta pokrenutog na ugradbenom Linux uređaju te poslužitelja koji upravlja instalacijom ažuriranja na određene uređaje. Rad ove

aplikacije temelji se na konceptu dvije kopije opisanom u prethodnom potpoglavlju. Mender pruža vrlo dobru integraciju s U-Boot *botloaderom* kako bi omogućio sigurni oporavak u slučaju greške. Nadalje, postoji vrlo dobro napisana dokumentacija na službenoj stranici koja opisuje kako podesiti i integrirati različite komponente. U njoj postoje primjeri implementacije ove aplikacije na poznate platforme poput Raspberry Pi i BeagleBone. Ovi primjeri mogu dati predodžbu o vremenu i načinu implementacije za specifičnu platformu.

Mender omogućuje integraciju s Yocto Project-om koristeći meta-mender [8] sloj. Nakon integracije i izgradnje dobije se slika (engl. *image*) sustava koja sadrži *bootloader* i datotečni sustav te se također dobiju artefakti koji sadrže novi datotečni sustav i različite podatke za ažuriranje. Artefakte je moguće instalirati ručno koristeći Mender CLI (engl. *Command Line Interface*) alat ili ih je moguće prenijeti na mrežni servis za ažuriranje pomoću mrežnog sučelja te nakon toga poslati na određene uređaje. Mrežno sučelje koje upravlja poslužiteljem za ažuriranje je dobro dizajnirano i jednostavno za korištenje. Postavljanje servera se vrši pomoću Docker platforme. Navedeno sučelje omogućuje prikaz svih dostupnih uređaja te prijenos ažuriranja na iste (individualno i u grupama) [9]. Mender je većim dijelom pisan u Go programskom jeziku te je potrebna određena razina poznavanja ovoga jezika kod razvoja ažuriranja temeljenog na ovome alatu.

Tablica 2.3. Karakteristike Mender mehanizma (1 dio) [6]

Mehanizam	Mender
Tip	Ažuriranje na razini blokova.
Struktura diska	Minimalan broj potrebnih particija je četiri. Ovisi o *bootloaderu* koji se koristi.
Rootfs	Pohranjen je na jednoj aktivnoj i jednoj pasivnoj particiji. Moguće je čitanje i pisanje, ali tijekom ažuriranja zapisane promjene se gube stoga važni podaci trebaju biti pohranjeni na zasebnoj podatkovnoj particiji.
Ažuriranje iz	Mender poslužitelja preko HTTPS protokola ili lokalnog datotečnog sustava
Ažuriranje čega	Ugrađena podrška za ažuriranje datotečnog sustava, uključujući jezgru, Moguće dodatno podešavanje za ažuriranje datoteka, paketa i vanjskih komponenti. Ne podržava *raw* NOR, NAND te UBI *volume* i particije.
Stabilnost koda	Relativno stabilan. potpuno podržan i testiran
Yocto integracija	meta-mender
Resursni zahtjevi na poslužitelju	Memorijski zahtjevi za pohranu arhivirane slike datotečnog sustava i meta podataka

Tablica 2.4. Karakteristike Mender mehanizma (2. dio) [6]

Resursni zahtjevi na klijentu	Potrebne dvije *rootfs* particije. Mali dodatni prostor potreban za Mender binarnu datoteku i bazu podataka.
Otpornost na greške	Automatski oporavak u slučaju greške prilikom pokretanja sustava ili greške prilikom povezivanja na Mender poslužitelj
Složenost	Relativno jednostavna izgradnja koristeći Yocto Project
Vrijeme instalacije	Ažuriranje se preuzima i instalira tijekom normalnog rada uređaja nakon čega je potrebno ponovno pokretanje sustava.
Sigurnost	Sigurno povezivanje na poslužitelj (TLS). Moguće potpisivanje artefakata s RSA ili ECDSA

2.2.2. SWUpdate

SWUpdate [5] je Linux agent kojemu je cilj ponuditi učinkovit i siguran način za ažuriranje ugradbenog sustava. SWUpdate podržava lokalna i bežična ažuriranja, više različitih strategija za ažuriranje te može biti vrlo lako integriran u Yocto sustav za izgradnju dodavanjem meta-swupdate sloja. Dostupan je i meta-swupdate-boards Yocto sloj koji podržava Raspberry Pi i BeagleBone platforme te koji može biti korišten kao primjer za specifičnu platformu. Artefakt za ažuriranje se sastoji od CPIO arhive koja sadrži meta podatke (engl. *metadata*) u obliku opisne datoteke i ostale podatke potrebne za instaliranje ažuriranja (Slika 2.5.)).

Slika 2.5. Izgled strukture CPIO arhive

Tablica 2.5. Karakteristike SWUpdate mehanizma [6]

Mehanizam	SWUpdate
Tip	Moguće ažuriranje na razini blokova i na razini datoteka.
Struktura diska	Nema ograničenja na način pohrane. Podržava *raw flash* (NOR,NAND), UBI *volume*, particije diska te može ažurirati datoteke u postojećem datotečnom sustavu. Svaki artefakt je moguće pohraniti na različit uređaj za pohranu.
Rootfs	Nema ograničenja na lokaciju pohrane. Tijekom ažuriranja moguća je instalacija na jednu ili više particija.
Ažuriranje iz	Moguće ažuriranje pomoću lokalnih uređaja za pohranu (USB, SD, …), dohvaćanjem preko HTTP(S) ili FTP protokola koristeći libcurl biblioteku, dohvaćanje koristeći mrežni poslužitelj na uređaju ili vanjski pozadinski konektor (Suricatta) za povezivanje s vanjskim pozadinskim poslužiteljem (hawkBit)
Ažuriranje čega	*Bootloadera* (rizično!), jezgre, sučelja *bootloadera*, disk particija, FPGA uređaja, vanjskih mikroupravljača i drugih. Moguće dodavanje vlastitih rukovoditelja pisanih u C programskom jeziku.
Stabilnost koda	Relativno stabilan, ažuriranje koda svaka 3 mjeseca
Yocto integracija	meta-swupdate sloj
Resursni zahtjevi na poslužitelju	Potrebni memorijski resursi za pohranu izgrađenih arhiva
Resursni zahtjevi na klijentu	U slučaju korištenja mehanizma jedne kopije, sustav za oporavak zauzima otprilike 8MB (*bootloader*, jezgra za SWUpdate te *ramdisk*) što omogućava instalaciju na uređaje s malim memorijskim resursima poput SPI-NOR dok se glavni sustav pohranjuje u NAND ili eMMC uređaje.Dodatni prostor za pohranu artefakata nije potreban ukoliko se instalacija ažuriranja vrši *streamanjem* s mrežnog poslužitelja.
Otpornost na greške	Moguće implementirati različite mehanizme za oporavak
Složenost	Jednostavan za korištenje, ali potrebne su dodatne prilagodbe sustava
Vrijeme instalacije	Kod mehanizma jedne kopije potrebno je ponovno pokretanje uređaja u modu za oporavak te nakon instalacije se vrši ponovno pokretanje ažuriranog sustava.
Sigurnost	Moguće je HTTPS povezivanje s vanjskim poslužiteljem, digitalno potpisivanje slika korištenih za ažuriranje radi provjere integriteta, podjela instalacije u više procesa te šifriranje artefakata.

Opisna datoteka sadrži listu datotečnih sustava te popis blok uređaja na koje trebaju biti instalirani. Za razliku od Mendera, SWUpdate ne definira striktno kako sustav mora biti podešen. Mender sam podešava U-Boot okruženje dok je kod SWUpdate mehanizma potrebno ručno na niskoj razini definirati kako će sustav dvije kopije raditi.

SWUpdate sustav je vrlo prilagodljiv i može biti konfiguriran koristeći kconfig sustav [9]. Podržava brojne opcije, neke od njih su: korištenje nekoliko različitih *bootloadera* (U-Boot, GRUB, EFI BG), potpisivanje i verifikaciju slika na temelju ponuđenog ključa te šifriranje slika koristeći simetrično šifriranje. SWUpdate nudi CLI alat za instalaciju ažuriranja te također kao i Mender omogućuje korištenje pozadinskog (engl. *Back-end*) sustava za ažuriranje koji se temelji na Eclipse hawkBit-u. HawkBit radi u sprezi s *daemonom* Suricatta koji se pokreće na uređaju te koji pokreće SWUpdate CLI proces. Nadalje, SWUpdate također može pokrenuti i ugrađeni HTTP server na uređaju s kojega je moguće prenijeti i instalirati ažuriranja.

U meta-swupdate sloju dostupan je Bitbake recept za izgradnju slike sustava dizajniranog da se pokreće kao *initramfs* sustav za oporavak. Sustav je prvenstveno dizajniran za rad na konceptu jedne kopije , iako je moguća i implementacija koncepta dvije kopije. SWUpdate je većinom pisan u C programskom jeziku s kojim je većina razvojnih inženjera upoznata što može značajno olakšati modifikacije i otklanjanje pogrešaka tijekom razvoja sustava temeljenog na ovom alatu. Kao i Mender, SWUpdate postavlja određene korisničke zahtjeve, ali su oni minimalni te su mnogi opcionalni zbog prilagodljive prirode sustava.

2.2.3. RAUC

RAUC [10] je klijent za ažuriranje koji se pokreće na Linux ugradbenom sustavu te pouzdano kontrolira postupak ažuriranja. RAUC se također pokreće i na sustavu za izgradnju te omogućuje stvaranje, pregled i izmjenu artefakata ažuriranja za određeni uređaj. Dizajniran je da bude memorijski što manje zahtjevan (engl. *lightweight*). Nakon pokretanja sustava izgradnje kao izlaz se dobiju takozvane *bundle* datoteke koje sadrže kompresirani datotečni sustav i meta podatke. Jedan od osnovnih zahtjeva RAUC alata je da *bundle*ovi moraju uvijek biti digitalno potpisani. RAUC instalira datotečne slike datotečnih sustava u različite *slotove* koji zatim mogu biti označeni kao neispravni ili kao spremni za ažuriranje. Ovaj alat zahtjeva omogućavanje određenih opcija u *jezgri* (primjerice SQUASHFS) te u *root* datotečnom sustavu. Kao i SWUpdate, RAUC je pisan u C programskom jeziku te podržava korištenje Eclipse hawkBit-a kao

pozadinskog sustava za ažuriranje. Za razliku od Mender i SWUpdate alata, RAUC ne podržava *streamanje* ažuriranja.

Tablica 2.6. Karakteristike RAUC mehanizma [6]

Mehanizam	RAUC
Tip	Moguće ažuriranje na razini blokova i na razini datoteka. Podržava datotečne sustave na blok uređajima, MTD (NAND/NOR) i *raw* slike (*bootloader*, FPGA)
Struktura diska	Ne ovisi o fiksnoj strukturi diskova, već omogućuje fleksibilnu konfiguraciju. Podržava koncept jedne i dvije kopije kao i složenije postavke.
Rootfs	Nema ograničenja na *root* datotečni sustav, može biti definiran samo za pisanje ili za čitanje i pisanje. Mora sadržavati RAUC konfiguracijsku datoteku, *keyring* za verifikaciju te malu RAUC binarnu datoteku.
Ažuriranje iz	RAUC je jezgra za ažuriranje koja je namijenjena za integraciju u prilagođenom okruženju. Zbog toga podržava instalaciju *bundleova* s lokalnih putanji, dok mora postojati dodatna aplikacija odgovorna za preuzimanje artefakata. Podržava i instalaciju s vanjskih uređaja za pohranu (USB, SD).
Ažuriranje čega	Nema posebnih ograničenja. Postoji zadani rukovoditelj za često korištene datotečne sustave (ext4, NAND, UBI, *raw*). Moguće kreiranje prilagođenih rukovoditelja za različite uređaje.
Stabilnost koda	Relativno stabilan, aktivno se radi na razvoju
Yocto integracija	meta-rauc
Resursni zahtjevi na poslužitelju	Memorijski prostor za pohranu izgrađenih arhiva
Resursni zahtjevi na klijentu	Minimalno jedan *rootfs* i sustav za oporavak. Privremeno mjesto za pohranu *bundle*ova za ažuriranje u slučaju OTA ažuriranja.
Otpornost na greške	Dostupno sučelje za komunikaciju s *bootloaderom* (Barebox, U-Boot, GRUB, UEFI). Implementaciju odgovarajućih mehanizama oporavka je potrebno napraviti u *bootloaderu*. Dostupno sučelje koje označava ažuriranje kao uspješno ili neuspješno nakon određenih provjera u pokrenutom ažuriranom sustavu.
Složenost	Integracija alata u datotečni sustav relativno jednostavna
Vrijeme instalacije	Uobičajeno, preuzimanje i instalacija ažuriranja se odvija u pozadini. Potrebno je ponovno pokretanje sustava.
Sigurnost	Obavezno je digitalno potpisivanje slika. CMS se koristi za potpisivanje *bundle*ova i podržana je potpuna infrastruktura javnog ključa. Verifikacija pokretanja sustava – kompatibilno s IMA, SELinux, dm-verify i drugim.

Odluka o tome koji od navedenih sustava koristiti ovisi o specifičnom projektu. Mender se čini kao rješenje s kojim je najjednostavnije krenuti, dok SWUpdate može biti vrlo koristan za integraciju sa sustavom koji je već djelomično razvijen na određenom operacijskom sustavu. Mender nudi potpuno rješenje sustava ažuriranja (klijent strana i poslužitelj strana) te nameće određene alate te zahtjeve koji moraju biti zadovoljeni. SWUpdate i RAUC su rješenja koja ne pružaju cjelovit sustav ažuriranja već su namijenjena za integraciju sa specifičnim sustavom stoga ih je moguće jednostavnije prilagoditi i konfigurirati prema specifičnim potrebama.

U svrhu realizacije ovog rada biti će korišten SWUpdate mehanizam zbog iznimne prilagodljivosti te ugrađene mogućnosti korištenja šifriranih artefakata. RAUC se čini kao vrlo dobra alternativa SWUpdate mehanizmu te može biti predmet istraživanja budućih sličnih radova. Nadalje, uz dostupne sigurnosne mehanizme i mehanizme za oporavak, SWUpdate omogućava definiranje i korištenje vlastitih. Sljedeća prednost ovoga mehanizma je ta da je napisan u C programskom jeziku te je stoga vrlo lako izmjenjiv za većinu programera. SWUpdate u usporedbi s ostala dva mehanizma definira najmanje sistemskih zahtjeva koji moraju biti zadovoljeni za pravilan rad.

3. OPIS ALATA POTREBNIH ZA IMPLEMENTACIJU SUSTAVA

U ovome poglavlju detaljno su opisani alati korišteni za implementaciju praktičnog dijela ovoga rada. Najprije su opisani Yocto Project i Bitbake alati za izgradnju, zatim SWUpdate mehanizam ta na kraju AWS mrežni servis.

3.1. Yocto Project i Bitbake alati za izgradnju paketa

Yocto Project je suradnički projekt otvorenog koda koji programerima omogućuje stvaranje Linux sustava neovisno o sklopovlju (engl. *hardware*). Dodatne značajke se dodaju koristeći slojeve. Primjerice meta-swupdate je sloj koji se koristi za unakrsno prevođenje (engl. *cross-compile*) SWUpdate aplikacije te za generiranje artefakata koji se koriste za ažuriranje.

Bitbake je alat za izgradnju koji se fokusira na distribucije i pakete za unakrsno prevođenje ugradbenih Linux operacijskih sustava, iako nije ograničen samo na ovu primjenu. Paketi se izgrađuju pomoću Bitbake recepata koji definiraju način izgradnje. Recepti se sastoje od URL reference za određeni paket, različitih međuovisnosti (engl. *dependencies*) te opcija za prevođenje i instalaciju. Yocto slojevi se sastoje od velikog broja recepata te konfiguracijskih datoteka koje omogućuju izgradnju recepata koristeći Bitbake alat. Konfiguracijska datoteka *bblayers.conf* definira koji će Yocto slojevi biti korišteni tijekom izgradnje.

3.2. SWUpdate

U ovome potpoglavlju biti će opisana sintaksa i oznake, korištenje potpisivanja i šifriranja artefakata te integracija unutar Yocto Projecta za SWUpdate alat.

3.2.1. Sintaksa i oznake

Opisna datoteka (engl. *description file*) sadrži osnovne informacije o ažuriranju i definira načine na koje će paketi biti instalirani u operacijski sustav. SWUpdate koristi biblioteku *libconfig* koja služi kao osnovni alat za izvlačenje informacija (engl. *parser*) iz opisne datoteke. Međutim, moguće je i dodavanje vlastitog *parsera* temeljenog na različitoj sintaksi i jeziku od onog definiranog u *libconfig*. Prva oznaka koja se navodi u opisnoj datoteci je *software* te je cijeli opis sadržan unutar te oznake. Na početku opisa najčešće se navode verzija i opis ažuriranja te podržane verzije sklopovlja (Programski kod 3.1.)

Podržano je dodavanje više uređaja (Programski kod 3.2.) te je na taj način moguće imati jednu sliku koja sadrži ažuriranje za veći broj uređaja. Informacija o trenutnoj verziji sklopovlja treba biti spremljena u datoteci */etc/hwrevision* koja mora sadržavati jednu liniju formata *<ime uređaja> <revizija>*.

```
software =
{
version = "0.1.0";
description = "Firmware update for XXXXX Project";

hardware-compatibility: [ "1.0", "1.2", "1.3"];
```

Programski kod 3.1. Primjer početka opisne datoteke

```
software =
{
    version = "0.1.0";

    target-1 = {
            images: (
                    {
                            ...
                    });};
    target-2 = {
            images: (
                    {
                            ...
                    });};
}
```

Programski kod 3.2. Primjer dodavanja podrške za više uređaja

Programske kolekcije (engl. *software collections*) i načini rada (engl. *operation modes*) mogu biti korišteni za implementaciju koncepta dvije kopije. Najjednostavniji slučaj je definirati dvije lokacije za instalaciju ažuriranja te pozvati SWUpdate s definiranom željenom slikom (Programski kod 3.3.). Na ovaj način moguće je definirati da kopija-1 bude instalirana na jedan uređaj, a kopija-2 na drugi.

Provjera sklopovske kompatibilnosti moguća je korištenjem oznake *hardware-compatibility: [„major.minor", "major.minor", ...]*. U zagradi trebaju biti navedene sve podržane verzije sklopovlja, primjerice *hardware-compatibility: ["1.0", "1.2", "1.3"];*. U navedenom primjeru je definirano da je ažuriranje kompatibilno samo sa sklopovljem verzija 1.0, 1.2 i 1.3. SWUpdate

omogućava korištenje oznaka za upravljanje particijama, skriptama, datotekama, slikama i *bootloaderom* (Programski kod 3.4 – 3.9.).

```
software =
{
        version = "0.1.0";
        stable = {
                copy-1: {
                        images: (
                        {
                                device = "/dev/mtd4"
                                ...
                        });
                }
                copy-2: {
                        images: (
                        {
                                device = "/dev/mtd5"
                                ...
                        });
                }};}
```

Programski kod 3.3. Korištenje koncepta dvije kopije

```
partitions: (
        {
                name = <volume name>;
                size = <size in bytes>;
                device = <MTD device>;
        },);
```

Programski kod 3.4. Oznaka za mijenjanje strukture za MTD *volume*

```
images: (
        {
                filename[mandatory] = <Name in CPIO Archive>;
                volume[optional] = <destination volume>;
                device[optional] = <destination volume>;
                mtdname[optional] = <destination mtd name>;
                type[optional] = <handler>;
                /* optionally, the image can be copied at a specific offset */
                offset[optional] = <offset>;
                /* optionally, the image can be compressed if it is in raw mode */
                compressed;
        },
        /* Next Image */ ..... );
```

Programski kod 3.5. Oznaka za instalaciju slika sustava

```
scripts: (
        {
                filename = <Name in CPIO Archive>;
                type = "shellscript";
        },);
```

Programski kod 3.6. Oznaka za upravljanje skriptama

```
files: (
        {
                filename = <Name in CPIO Archive>;
                path = <path in filesystem>;
                device[optional] = <device node >;
                filesystem[optional] = <filesystem for mount>;
                properties[optional] = {create-destination = "true";}
        });
```

Programski kod 3.7. Oznaka za instalaciju datoteka

Shell skripte se pozivaju koristeći sistemski poziv. SWUpdate pretražuje dostupne skripte i poziva ih prije i polije instalacije slika ili datoteka. SWUpdate prosljeđuje *preinst* ili *postinst* kao prvi argument skripte. Ako je podatkovni atribut definiran, njegova vrijednost se prosljeđuje kao zadnji argument. Podržani su *lua* i *shell* tipovi skripti. Varijable okruženja *bootloadera* moguće je ažurirati na dva načina. Prvi način je dodavanjem datoteke s popisom varijabli koje trebaju biti promijenjene te postavljanje tipa slike *bootloader* (Programski kod 3.8.). Ovakav način obavještava SWUpdate da pozove rukovoditelja *bootloadera* da obradi datoteku (zahtjeva se omogućavanje podrške za *bootloader* rukovoditelja u konfiguracijskoj datoteci). Drugi način je definiranje imena i vrijednosti za svaku varijablu koju želimo postaviti ili promijeniti (Programski kod 3.9.).

```
images: (
        {
                filename = "bootloader-env";
                type = "bootloader";
        },)
```

Programski kod 3.8. Oznaka za ažuriranje varijabli okruženja *bootloadera* koristeći datoteku s popisom varijabli

```
bootenv: (
        {
                name = <Variable name>;
                value = <Variable value>;
        },)
```

Programski kod 3.9. Oznaka za ažuriranje varijabli okruženja *bootloadera* definiranjem imena i vrijednosti za svaku varijablu

Sljedeća bitna značajka je da SWUpdate može provjeriti jesu li određena ažuriranja već instalirana te ako je verzija koja se želi instalirati potpuno jednaka postojećoj, moguće je preskočiti njihovu instalaciju. SWUpdate traži datoteku */etc/sw-versions* (moguće definirati drugi naziv i putanju)

koja sadrži sve verzije instaliranih slika. Ta datoteka treba biti kreirana prije pokretanja SWUpdate aplikacije te mora sadržavati parove s imenom slike i njenom verzijom (Programski kod 3.10.)

```
bootloader              2015.01-rc3-00456-gd4978d
kernel                  3.17.0-00215-g2e876af
```

Programski kod 3.10. Datoteka koja sadržava instalirane verzije softvera

3.2.2. Digitalno potpisivanje i šifriranje artefakta

Osim sigurnog ažuriranja uređaja, vrlo je bitno verificirati potječe li dostavljena slika s poznatog izvora te provjeriti da podaci nisu neovlašteno mijenjani. Kako bi se zadovoljili navedeni zahtjevi SWUpdate nudi mogućnost verifikacije dolaznih slika tako da kombinira digitalno potpisanu opisnu datoteku s verifikacijom *hash*eva za svaku pojedinu sliku. To znači da samo opisna datoteka generirana od strane ovlaštenog izvora može biti prihvaćena kao ispravna. Opisna datoteka sadrži *hash*eve za sve slike kako bi potvrdio da svaka od njih uistinu pripada odgovarajućem ažuriranju.

Željeni mehanizam za digitalno potpisivanje može biti odabran u konfiguracijskoj datoteci. Trenutno SWUpdate nudi dva mehanizma koji mogu biti korišteni: RSA s javnim i privatnim ključem te CMS s certifikatima. Ključ ili certifikat prosljeđuju se SWUpdate aplikaciji pomoću -k parametra. Za generiranje ključeva koristi se openssl alat. Način generiranja ključeva i potpisivanja opisan je u dokumentaciji za SWUpdate.

Nadalje, SWUpdate omogućuje i simetrično šifriranje slika za ažuriranje koristeći 256 bitno AES blokovsko šifriranje u CBC načinu. Najprije je potrebno kreirati ključ koristeći *openssl* naredbu: openssl enc -aes-256-cbc -k <PASSPHRASE> -P -md sha1. Ključ i inicijalizacijski vektor generiraju se na temelju upisane *passphrase*. Izlaz komande izgleda kako je prikazano na slici 3.1.

```
salt=CE7B0488EFBF0D1B
key=B78CC67DD3DC13042A1B575184D4E16D6A09412C242CE253ACEE0F06B5AD68FC
iv =65D793B87B6724BB27954C7664F15FF3
```

Slika 3.1. Izgled generiranog ključa

Šifriranje datoteka se vrši pomoću naredbe openssl enc -aes-256-cbc -in <ulazna datoteka> -out <izlazna datoteka> -K <ključ> -iv <IV> -S <SALT>. <ulazna datoteka> predstavlja nešifriranu

sliku, dok <izlazna datoteka> predstavlja šifriranu sliku koja mora biti referencirana u opisnoj datoteci (*sw-description*). Prilikom instalacije ažuriranja datoteka koja sadrži ključ predaje se SWUpdate aplikaciji koristeći parametar -K. Ključ, *iv* i *salt* trebaju biti napisani u jednoj liniji te odvojeni razmakom. U programskom kodu 3.11. prikazan je primjer sadržaja opisne datoteke za Beaglebone uređaj s uključenim šifriranjem. Podršku za šifriranje slike potrebno je uključiti u konfiguracijskoj datoteci postavljanjem ENCRYPTED_IMAGES parametra.

```
software =
{
        version = "0.0.1";
        images: ( {
                        filename = "core-image-full-cmdline-beaglebone.ext3.enc";
                        device = "/dev/mmcblk0p3";
                        encrypted = true;
                });}
```

Programski kod 3.11. Primjer sadržaja opisne datoteke za Beaglebone uređaj

3.2.3. Integracija u Yocto Project

Za integraciju u Yocto dostupan je meta-swupdate sloj za unakrsno prevođenje SWUpdate aplikacije te za generiranje SWU slika sustava koje sadrže određenu verziju ažuriranja. Kako bi bilo moguće izgraditi komponente ovog sloja, potrebno ga je dodati u *bblayers.conf* konfiguracijsku datoteku vlastitog projekta. meta-swupdate sloj ovisi o meta-oe sloju tako da ovaj sloj također treba biti uključen u projekt. Nakon dodavanja sloja u projekt, pomoću dostupnog recepta moguće je izgraditi inicijalni RAM disk (engl. *Initial ramdisk - initrd*) s sustavom za oporavak koji sadrži SWUpdate aplikaciju. Ovakav sustava koristi se kod koncepta ažuriranje temeljenog na jednoj kopiji sustava. Instalacija i pokretanje *initrda* je vrlo specifično za različite platforme te je definirano u dokumentaciji *bootloadera* koji ta platforma koristi.

SWUpdate aplikacija također ovisi o *libuboot* biblioteci (u slučaju kada se koristi U-Boot *bootloader*) koja omogućava sigurnu izmjenu U-Boot okruženja. Ova biblioteka je razvijena da bude neovisna o sklopovlju kako ne bi došlo do grešaka prilikom izmjene varijabli okruženja. Nadalje, meta-swupdate sadrži klasu koja pomaže kod generiranja SWU slika ažuriranja koristeći slike izgrađene unutar Yocto projekta. Klasa zahtjeva da sve komponente, odnosno artefakti koji su dio SWU slike, budu prisutni u Yocto direktoriju koji sadrži sve izgrađene slike sustava (engl. *deploy directory*). Navedenu klasu treba naslijediti recept koji generira SWU sliku. Klasa definira nove varijable s prefiksom SWUPDATE_ u imenu. Opisi ovih varijabli navedeni su u tablici 3.1.

SWUpdate klasa također može i automatski računati i dodavati sha256 *hash*eve o opisnoj datoteci (*sw-description*). Atribut sha256 uvijek mora biti postavljen u slučaju kada se koristi digitalno potpisivanje. Za svaki artefakt mora postojati sha256 atribut, primjerice sha256 = „@ime-artefakta". U opisnoj datoteci je moguće dohvaćanje vrijednosti Bitbake varijabli korištenjem znakova „@@“ prije i poslije imena varijable. Primjerice version = „@@DISTRO_VERSION@@“. Predložak za kreiranje Bitbake recepta, koristeći swupdate klasu, prikazan je u programskom kodu 3.12.

Tablica 3.1. Imena i opisi varijabli sadržanih u SWUpdate klasi

IME	OPIS
SWUPDATE_IMAGES	popis artefakata koji trebaju biti upakirani zajedno
SWUPDATE_IMAGES_FSTYPES	ekstenzija artefakata
SWUPDATE_SIGNING	ako je postavljena, SWU slika se potpisuje (moguće su tri vrijednosti: RSA; CMS i CUSTOM)
SWUPDATE_SIGN_TOOL	postavlja se u slučaju kada se ne koristi openssl za potpisivanje, već prilagođeni način (SWUPDATE_SIGNING treba biti postavljeno u CUSTOM
SWUPDATE_PRIVATE_KEY	lokacija privatnog ključa koji se koristi za RSA potpisivanje
SWUPDATE_PASSWORD_FILE	lokacija do lozinke koja se koristi za privatni ključ kod RSA
SWUPDATE_CMS_KEY	lokacija privatnog ključa koji se koristi za CSM potpisivanje
SWUPDATE_CMS_CERT	lokacija certifikata koji se koristi kod potpisivanja
SWUPDATE_AES_FILE	lokacija datoteke s AES lozinkom za šifriranje artefakta (SWUpdate klasa nudi podršku za enc tip datotečnog sustava)

```
DESCRIPTION = "Example recipe generating SWU image"
SECTION = ""
LICENSE = ""

# Add all local files to be added to the SWU
# sw-description must always be in the list.
# You can extend with scripts or wahtever you need
SRC_URI = " \
    file://sw-description \
    "

# images to build before building swupdate image
IMAGE_DEPENDS = "core-image-full-cmdline virtual/kernel"

# images and files that will be included in the .swu image
SWUPDATE_IMAGES = "core-image-full-cmdline uImage"

# a deployable image can have multiple format, choose one
SWUPDATE_IMAGES_FSTYPES[core-image-full-cmdline] = ".ubifs"
SWUPDATE_IMAGES_FSTYPES[uImage] = ".bin"

inherit swupdate
```

Programski kod 3.12. Primjer recepta koji generira SWU sliku

3.3. Qt programsko okruženje i Qt Creator

Qt programsko okruženje [11] najčešće se koristi kao alat za izradu grafičkih aplikacija, iako je vrlo koristan i kod izrade konzolnih aplikacija. Može biti korišten na više različitih platformi poput Windows, macOS, Linux, različiti mobilni operacijski sustavi te ugradbeni računalni sustavi. Qt se sastoji od velike kolekcije modula i dodataka (engl. *add-ons*) koji su vrlo korisni za razvoj Qt aplikacija. Primjeri takvih modula navedeni su u tablici 3.2.

Tablica 3.2. Osnovni Qt moduli i njihovi opisi [12]

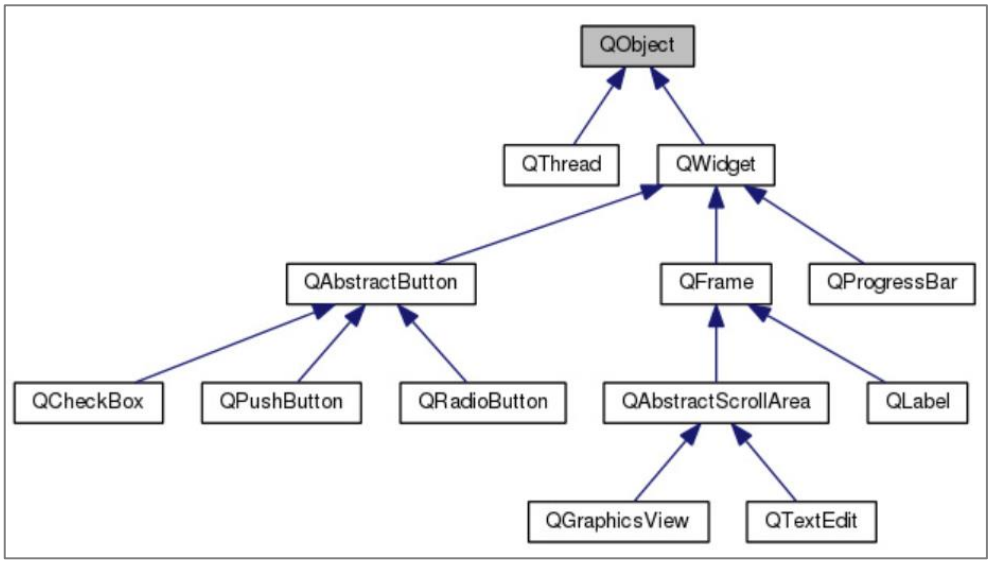
Modul	Opis
Qt Core	osnovna biblioteka koja sadrži spremnike (engl. *containers*), upravljanje procesima, upravljanje događajima te brojne klase koje koriste drugi moduli
Qt GUI	sadrži osnovne klase za kreiranje komponenti grafičkog korisničkog sučelja te OpenGL grafičku biblioteku.
Qt Multimedia	omogućava rad s audio, video, radio i drugim sličnim elementima
Qt Network	sadrži klase za mrežnu komunikaciju
Qt Webkit	omogućuje korištenje internetskih stranica i aplikacija u Qt aplikaciji
Qt Test	sadrži klase za Unit testiranje Qt aplikacija i biblioteka

Add-On moduli donose dodatne mogućnosti za posebne primjene. Primjeri takvih modula su: Qt Bluetooth, Qt D-Bus, Qt-NFC, Qt Sensors, Qt Speech i drugi. Qt Creator je integrirano razvojno okruženje (engl. *Integrated Development Environment*) za C++ programski jezik koje pruža vrlo dobro dizajnirano sučelje za izradu aplikacija. *QApplication* je vrlo bitna klasa koja između ostalog, obrađuje ulazne argumente te izvršava *event loop*. *Event loop* je petlja koja čeka korisnički unos u grafičkoj aplikaciji. Pokreće se pozivom funkcije *app.exec()*. Qt koristi qmake alat za izgradnju koji kreira make datoteku (*makefile*) koja se koristi za prevođenje Qt aplikacije.

U najjednostavnijem slučaju Qt aplikacije se prevode u tri koraka:

1. pisanje .pro datoteke koja opisuje projekt koji je potrebno prevesti
2. generiranje make datoteke pomoću qmake alata
3. izgradnja aplikacije koristeći make alat

Qt uvelike primjenjuje mehanizam nasljeđivanja, posebice u Widgets modulu (Slika 3.2.)



Slika 3.2. Primjer nasljeđivanja u Qt-u [13]

QObject je najosnovnija klasa u Qt-u koju nasljeđuju gotovo sve ostale klase. QObject nudi nekoliko iznimno korisnih mogućnosti/mehanizama:

1. dodjeljivanje imena objektu u obliku znakovnog niza (engl. *string*) te pretraživanje objekata prema imenu.
2. mehanizam nasljeđivanja
3. mehanizam signala i *slotova*
4. upravljanje događajima (engl. *Event management*)

Mehanizam signala i *slotova* koristi se za komunikaciju između objekata. Ovo je centralna značajka Qt-a koja ga razlikuje od dostupnih sličnih razvojnih okruženja. Umjesto tehnike povratnog poziva (engl. *Callback technique*), Qt koristi signale koji se šalju kada se pojavi određeni događaj te *slotove* koji se pozivaju kao odgovor na određeni signal.

3.4. Amazon S3 i Boto3 SDK

Amazon S3 ili Amazon Simple Storage Service je servis [14], izdan od strane Amazon Web Services kompanije, koji nudi uslugu pohrane i preuzimanja podataka koristeći internetsko sučelje.

Osnovne značajke S3 servisa su:

1. kreiranje i imenovanje *bucketa* za pohranu podataka – *bucketi* predstavljaju osnovne spremnike za pohranu podataka
2. pohrana podataka u *bucket*
3. preuzimanje podataka
4. prava pristupa – omogućuju odobravanje ili uskraćivanje prava pristupa za preuzimanje i slanje podataka u *bucket*
5. sučelja za pristup – moguće je koristiti REST (engl. *Representational State Transfer*) aplikacijsko programsko sučelje (engl. *Application Program Interface*), internetsku aplikaciju, AWS konzolnu aplikaciju ili AWS SDK

Najjednostavniji način upravljanja *bucketima* i objektima je pomoću internetske aplikacije. Aplikacija pruža vrlo jednostavno i intuitivno grafičko sučelje. Drugi način upravljanja je koristeći AWS konzolnu aplikaciju koja koristi Amazon S3 API-je za slanje zahtjeva na servis. Ovaj način je pogodniju u slučaju kada koristimo sustav koji nema dostupno grafičko sučelje. Konzolnu aplikaciju moguće je instalirati na Windows, Linux i macOS sustave. Neke od osnovnih naredbi za upravljanje *bucketima* i objektima koristeći konzolnu aplikaciju prikazane su u tablici 3.3.

Za upravljanje Amazon S3 servisima na ugradbenim računalnim sustavima koji imaju ograničene memorijske resurse koriste se AWS SDK-ovi za različite programske jezike (Java, .NET, Ruby, C++, Python). U ovome radu biti će korišten Boto3 SDK za Python zbog izrazito malih memorijskih zahtjeva. Boto3 je Python paket koji pruža sučelje za S3 servis te brojne druge Amazon internetske servise. U Linux operacijski sustav moguće ga je dodati koristeći Bitbake

recept. Primjer recepta za izgradnju Boto3 paketa prikazan je na slici 3.17. Način pozivanja Python skripte koja preuzima objekt s S3 *bucketa* biti će opisan u sljedećem poglavlju.

Tablica 3.3. Upravljanje *bucketima* i objektima koristeći AWS konzolnu aplikaciju [15]

Naredba	Opis
aws configure	postavljanje korisničkih podataka za pristup Amazon S3 servisu
aws s3 mb s3://tgsbucket	kreiranje novog *bucketa* naziva *tgsbucket*
aws s3 rb s3://tgsbucket	brisanje *bucketa* naziva *tgsbucket*
aws s3 ls s3://tgsbucket	ispis objekata koji se nalaze u *bucketu*
aws s3 cp test.txt s3://tgsbucket	kopiranje lokalne datoteke test.txt u *bucket*
aws s3 cp s3://tgsbucket/test.txt	preuzimanje test.txt datoteke s *bucketa* u trenutni direktorij

```
SUMMARY = "The AWS SDK for Python"
HOMEPAGE = "https://github.com/boto/boto3"
DESCRIPTION = "\
  Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python
  "
AUTHOR = "Amazon Web Services <>"
LICENSE = "Apache-2.0"
LIC_FILES_CHKSUM = "file://LICENSE;md5=2ee41112a44fe7014dce33e26468ba93"
SRC_URI = "https://files.pythonhosted.org/packages/7a/f0/\
           6a2265eecb1c17c33c8b7a226407e1f17e51c1fa54bc9d7e6d3cde6de2c5/\
           boto3-1.9.185.tar.gz"
SRC_URI[md5sum] = "bcf5b44834036e9e054fd202b136d2ef"
SRC_URI[sha256sum] = "18fdfbaf8f82059307b8eab31f87c65e7d0e8873f4a5b0c891bd7ce85fc23d41"
S = "${WORKDIR}/boto3-1.9.185"
RDEPENDS_${PN} = "\
    ${PYTHON_PN}-jmespath     ${PYTHON_PN}-dateutil     \
    ${PYTHON_PN}-docutils     ${PYTHON_PN}-urllib3      \
    ${PYTHON_PN}-six          ${PYTHON_PN}-botocore     \
    ${PYTHON_PN}-s3transfer   ${PYTHON_PN}-utils        \
    ${PYTHON_PN}-progressbar  \
"
inherit setuptools3
```

Programski kod 3.13. Bitbake recept za Boto3 SDK

4. RAZVOJ I IMPLEMENTACIJA RJEŠENJA

U ovome poglavlju je detaljno opisano kompletno rješenje bežičnog ažuriranja Linux ugradbenog operacijskog sustava. U prvome potpoglavlju opisan je način dodavanja meta-swupdate sloja u Yocto Project te način izgradnje slike operacijskog sustava. U sljedećem potpoglavlju opisana je struktura recepta koji se koristi za izgradnju slike koja se koristi za ažuriranje sustava, dok je u zadnjem potpoglavlju kreirana konzolna aplikacija koja upravlja cijelim procesom ažuriranja operacijskog sustava.

4.1. Dodavanje meta-swupdate sloja u Yocto Project

Operacijski sustav na kojemu će se vršiti implementacija mehanizma ažuriranja izgrađen je koristeći Yocto Project i Bitbake alat. Osnovni sloj koji sadrži recepte za izgradnju potrebnih paketa i aplikacija operacijskog sustava naziva se meta-concept. Bitbake recept koji će biti korišten za izgradnju sustava prikazan je u programskom kodu 4.1.

.

```
##########################################################################
##
## Copyright (C) 2019 Rimac Automobili
## David Tkalcec, david.tkalcec@rimac-automobili.com
##
##########################################################################
##Include Base IVI Concept image
include concept-image-ivi.inc
##Inherit swupdate class
inherit concept-embedded-image-swupdate

DESCRIPTION = "Build Concept one IVI project"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${THISDIR}/concept-image-
shared/COPYING.MIT;md5=****"
FILESEXTRAPATHS_prepend := "${THISDIR}/concept-image-shared:"
CONCEPT-ONE_FILESPATH = "${THISDIR}/concept-image-concept-one"
FILESEXTRAPATHS_prepend := "${CONCEPT-ONE_FILESPATH}:"
## Additions for Concept one Qt app
IMAGE_INSTALL += "            \
     concept-one              \
     python3-boto3            \
  "
```

Programski kod 4.1. Izgled recepta za izgradnju slike sustava

U ovome receptu uključuje se baza operacijskog sustava te dodatni elementi specifični za Concpet_one projekt. Neki od tih elemenata su *patchevi*, recepti koji trebaju biti dodani u sliku te dodatne datoteke. Kao što je opisano u prethodnom poglavlju, meta-swupdate sloj je potrebno dodati unutar Yocto Project-a koji se koristi za izgradnju prilagođenog sustava. Ovaj korak radi se tako da se meta-swupdate sloj najprije preuzme s Git repozitorija u izvorni direktorij gdje se nalaze svi slojevi o kojima sustav ovisi (uključujući meta-concept) te se zatim ime sloja doda u *bblayers.conf* konfiguracijsku datoteku Bitbake alata (Programski kod 4.2.). Nakon ovog koraka sve recepte koji se nalaze unutar meta-swupdate sloja možemo koristiti u meta-concept sloju. Unutar meta-swupdate sloja nalaze se recepti za izgradnju *intrid* slike sustava, SWUpdate aplikacije koja se koristi za ažuriranje, servisa za hawkBit te servisa za ispis napretka ažuriranja. Osim recepata, unutar navedenog sloja vrlo je bitna *defconfig* datoteka koja služi za konfiguriranje značajki koje će biti uključene u SWUpdate.

```
POKY_BBLAYERS_CONF_VERSION = "2"
BBPATH = "${TOPDIR}"
BBFILES ?= ""
BSPDIR := "${@os.path.abspath(os.path.dirname(d.getVar('FILE')) + '/../..')}"
BBLAYERS ?= " \
${BSPDIR}/sources/poky/meta \
  ${BSPDIR}/sources/poky/meta-poky \
  ${BSPDIR}/sources/meta-freescale \
  ${BSPDIR}/sources/meta-freescale-3rdparty \
  ${BSPDIR}/sources/meta-toradex-bsp-common \
  ${BSPDIR}/sources/meta-toradex-nxp \
  ${BSPDIR}/sources/meta-openembedded/meta-oe \
  ${BSPDIR}/sources/meta-openembedded/meta-python \
  ${BSPDIR}/sources/meta-openembedded/meta-networking \
  ${BSPDIR}/sources/meta-openembedded/meta-initramfs \
  ${BSPDIR}/sources/meta-openembedded/meta-multimedia \
  ${BSPDIR}/sources/meta-boot2qt/meta-boot2qt \
  ${BSPDIR}/sources/meta-boot2qt/meta-boot2qt-distro \
  ${BSPDIR}/sources/meta-boot2qt/meta-fsl-extras \
  ${BSPDIR}/sources/meta-boot2qt/meta-toradex-extras \
  ${BSPDIR}/sources/meta-mingw \
  ${BSPDIR}/sources/meta-qt5 \
  "
BBLAYERS += " \
  ${BSPDIR}/sources/meta-swupdate \
"
```

Programski kod 4.2. Izgled *bblayers.conf* datoteke s dodanim meta-swupdate slojem

U meta-concept sloju najprije je potrebno dodati novu klasu koju će biti naslijeđena u *concept-image-concept-one.bb* receptu. Klasa naziva *concept-embedded-image-swupdate.bbclass* kreirana je kako bi olakšala uključivanje alata i datoteka specifičnih za SWUpdate u različite recepte. Ova klasa obavlja sljedeće zadatke:

1. dodavanje SWUpdate alata u sliku sustava (Programski kod 4.3.)
2. izdvajanje *boot* i *opt* slika iz *rootfs* slike i enkripcija *boot* i *opt* slike – svaka slika će se koristiti za instalaciju na određenu particiju u memoriji (Programski kod 4.4.)
3. šifriranje *rootfs* slike (Programski kod 4.5.)

```
## Additions for SWUpdate

IMAGE_INSTALL += "                          \
     swupdate                               \
     swupdate-tools                         \
     lua                                    \
     u-boot-fw-utils                        \
  "
```

Programski kod 4.3. Dodavanje SWUpdate alata u sliku sustava

```
SWUPDATE_ENCRYPT_KEY = "/etc/keys/aes-cbc-key.txt"
SALT="$(cat ${SWUPDATE_ENCRYPT_KEY} | grep -o "salt=\S*" \
| awk -F "=" '{print $2}')"
KEY="$(cat ${SWUPDATE_ENCRYPT_KEY} | grep -o "key=\S*" \
| awk -F "=" '{print $2}')"
IV="$(cat ${SWUPDATE_ENCRYPT_KEY} | grep -o "iv =\S*" \
| awk -F "=" '{print $2}')"
#openssl enc -aes-256-cbc -in <INFILE> -out <OUTFILE> -K <KEY> -iv <IV>
-S <SALT>

do_encrypted_rootfs_image(){

   cd ${WORKDIR}/deploy*
   openssl enc -aes-256-cbc -in ${PN}-${MACHINE}.tar.gz \
   -out ${PN}-${MACHINE}.tar.gz.enc -K ${KEY} -iv ${IV} -S ${SALT}
}
addtask encrypted_rootfs_image after do_image_tar before
do_image_complete
```

Programski kod 4.4. Šifriranje *rootfs* slike

```
## Make boot tar.gz image and remove content of boot dir in rootfs
do_boot_image(){
  if [ x"${SEPARATE_IMAGES}" = "xtrue" ]; then
    echo "Separating boot directory to distinct image..."
    cd ${WORKDIR}/rootfs/boot
    tar -zcvf ${PN}-BOOT-${MACHINE}.tar.gz *
    openssl enc -aes-256-cbc -in ${PN}-BOOT-${MACHINE}.tar.gz \
    -out ${PN}-BOOT-${MACHINE}.tar.gz.enc -K ${KEY} -iv ${IV} -S ${SALT}
    mv ${PN}-BOOT-${MACHINE}.tar.gz ${DEPLOY_DIR_IMAGE}
    mv ${PN}-BOOT-${MACHINE}.tar.gz.enc ${DEPLOY_DIR_IMAGE}
    rm ${WORKDIR}/rootfs/boot/*
  fi
}
addtask boot_image after do_rootfs before do_image

## Make opt tar.gz image and remove rapp dir from rootfs
do_rapp_image(){
  echo "SEPARATE_IMAGES: "${SEPARATE_IMAGES}
  if [ x"${SEPARATE_IMAGES}" = "xtrue" ]; then
    echo "Separating opt directory to distinct image..."
    cd ${WORKDIR}/rootfs/opt
    tar -zcvf ${PN}-OPT-${MACHINE}.tar.gz rapp
    openssl enc -aes-256-cbc -in ${PN}-OPT-${MACHINE}.tar.gz \
    -out ${PN}-OPT-${MACHINE}.tar.gz.enc -K ${KEY} -iv ${IV} -S ${SALT}
    mv ${PN}-OPT-${MACHINE}.tar.gz ${DEPLOY_DIR_IMAGE}
    mv ${PN}-OPT-${MACHINE}.tar.gz.enc ${DEPLOY_DIR_IMAGE}
    rm -r ${WORKDIR}/rootfs/opt/rapp
  fi
}
addtask rapp_image after do_rootfs before do_image
```

Programski kod 4.5. Izdvajanje *boot* i *opt* slika iz *rootfs* slike i šifriranje

Sljedeći korak je kreiranje *swupdate_%.bbappend* recepta u meta-concept sloju unutar direktorija *recipes-swupdate/SWUpdate*. Osnovna uloga ovoga recepta je definiranje vlastite *defconfig* datoteke koja će se koristiti umjesto one u meta-swupdate sloju. Unutar ovoga recepta također se mogu dodati vlastiti servisi ili ukloniti postojeći ukoliko nam nisu potrebni. Sadržaj *swupdate_%.bbappend* recepta prikazan je u programskom kodu 4.6. Varijable *defconfig* datoteke za potrebe ovoga projekta potrebno je postaviti kako je definirano u tablicama 4.1. i 4.2 Nakon obavljenih prethodno opisanih koraka prilagodili smo SWUpdate alat prema vlastitim potrebama..

```
##############################################################################
##
## Copyright (C) 2019 Rimac Automobili
## David Tkalcec, david.tkalcec@rimac-automobili.com
##
##############################################################################
DESCRIPTION = "Including defconfig file"
FILESEXTRAPATHS_append := "${THISDIR}/${PN}:"
PACKAGECONFIG_CONFARGS = ""

SRC_URI += " file://defconfig "
do_install_append() {
   rm -rf ${D}${systemd_unitdir}/system/swupdate.service
   rm -rf ${D}${systemd_unitdir}/system/swupdate-progress.service
   rm -rf ${D}${systemd_unitdir}/system/swupdate-usb@.service
   rm -rf ${D}/www
   rm -rf ${D}${bindir}/swupdate-hawkbitcfg
   rm -rf ${D}${bindir}/swupdate-sendtohawkbit
    if ${@bb.utils.contains('DISTRO_FEATURES','systemd','true','false',d)}; then
      rm -rf ${D}${sysconfdir}/udev/rules.d/swupdate-usb.rules
   fi
}
FILES_${PN} += "\
   /lib/*
"
SYSTEMD_SERVICE_${PN} = ""
```

Programski kod 4.6. Sadržaj *swupdate_%.bbappend* recepta

Tablica 4.1. Postavke *defconfig* datoteke (1.dio)

swupdate konfiguracija	opcije vezane za sigurnost i bootloader
CONFIG_HAVE_DOT_CONFIG=y; CONFIG_SCRIPTS=y CONFIG_HW_COMPATIBILITY=y CONFIG_HW_COMPATIBILITY_FILE="/etc/hwrevision" CONFIG_SW_VERSIONS_FILE="/etc/sw-versions" CONFIG_MTD=y CONFIG_LUA=y CONFIG_LUAPKG="lua"	CONFIG_BOOTLOADER=y CONFIG_UBOOT=y CONFIG_UBOOT_FWENV="/etc/fw_env.config" CONFIG_DOWNLOAD=y CONFIG_HASH_VERIFY=y CONFIG_SIGNED_IMAGES=y CONFIG_ENCRYPTED_IMAGES=y

Tablica 4.2. Postavke *defconfig* datoteke (2.dio)

rukovoditelji	ostalo
CONFIG_RAW=y	CONFIG_SURICATTA_SERVER_NONE=y
CONFIG_LUASCRIPTHANDLER=y	CONFIG_GUNZIP=y
CONFIG_SHELLSCRIPTHANDLER=y	CONFIG_LIBCONFIG=y
CONFIG_ARCHIVE=y	CONFIG_PARSERROOT=""
CONFIG_BOOTLOADERHANDLER=y	CONFIG_CFI=y

4.2. Kreiranje opisne datoteke i recepta za izgradnju SWU slike

Nakon obavljenih koraka u prethodnom potpoglavlju definirali smo koji će alati dodatno biti izgrađeni u našem operacijskom sustavu. Sada je potrebno definirati opisnu datoteku (*sw-description*) te napisati recept koji će izgrađivati SWU sliku potrebnu za instalaciju ažuriranja na uređaj. Sintaksa za pisanje opisne datoteke opisana je u potpoglavlju 3.2.1.

Važna stavka koja mora biti definirana prije pisanja opisne datoteke je broj i raspored particija u memoriji uređaja. Na Apalis iMX6 SoC-u dostupno je 4GB eMMC memorije. Cilj je implementirati sustav ažuriranja temeljen na konceptu dvije kopije. Memorija će biti particionirana na način da će postojati ukupno sedam particija: boot1, boot2, root1, root2, opt1, opt2 i var. *Boot* particije sadrže jezgru i *Device Tree, root* sadrže datotečni sustav, *opt* sadrže aplikacije specifične za Concept_one projekt te *var* sadrži promjenjive podatke operacijskog sustava. Particije označene brojem jedan koriste se za prvu kopiju sustava, dok se one označene brojem dva koriste za drugu kopiju sustava. *Var* particija je zajednička. Nakon definiranja particija možemo implementirati mehanizam „dvije kopije unutar opisne datoteke. Najprije je potrebno definirati kostur opisne datoteke kako je prikazano u programskom kodu 4.7. Na početku opisne datoteke najprije se navode verzija i opis ažuriranja. Zatim se dodaje ime uređaja za koji je ažuriranje namijenjeno (u našem slučaju apalis-imx6) unutar kojega se dodaju podržane verzije uređaja (*hardware-compatibility: [1.0]*). Zatim slijedi definiranje načina rada dvije kopije. Svaka kopija sadrži oznake (*files, scripts, uboot*) unutar kojih je potrebno definirati slike, skripte i okruženje *bootloadera* koje će se koristiti u procesu ažuriranja. Za svaku od ovih oznaka potrebno je podesiti parametre koji definiraju na koji način će one biti instalirane tijekom ažuriranja. U programskom kodu 4.8. prikazan je dio opisne datoteke koji se koristi za instalaciju ažuriranja na prvu grupu particija (prva kopija sustava). Analogno ovome piše se i dio za drugu grupu particija.

```
software =
{
    version = "C1_v0.0";
        description = "Software update for Concept one IVI project";
    apalis-imx6 = {
        hardware-compatibility: ["1.0"];
        stable : {
            copy1 : {
                files: (...);
                scripts: (...);
                uboot: (...); };
            copy2 : {
                files: ( ... );
                scripts: (...);
                uboot: (...);};
};
       }
}
```

Programski kod 4.7. Kostur opisne datoteke koja implementira mehanizam dvije kopije

Unutar *files* oznake potrebno je definirati način na koji će biti instalirane *root, bot* i *opt* slike. Za svaku sliku definirani su sljedeći parametri: ime slike, verzija slike, tip rukovoditelja, putanja uređaja na koji će slika biti instalirana, tip datotečnog sustava, putanja na koju će slika biti instalirana te sha256 izraz. Parametar *install-if-different*, ukoliko je postavljen u *true*, daje naznaku da će ažuriranje biti instalirano samo ukoliko se trenutna verzija sustava razlikuje od one koja je dostupna u ažuriranju. Parametar *encrypted* daje naznaku koriste li se šifrirane slike. Zatim unutar *scripts* oznake navode se skripte koje će biti korištene u procesu ažuriranja. Skripte su definirane imenom, tipom, sha256 izrazom te *encrypted* parametrom. *uboot* oznaka navodi imena i vrijednosti varijabli koje će biti podešene u U-Boot okruženju.

Nakon kreiranja opisne datoteke, još preostaje kreirati Bitbake recept za izgradnju SWU slike koja sadrži komponente operacijskog sustava (rootfs, boot i opt), skripte, okruženje *bootloadera* i opisnu datoteku. SWU slika se prilikom instalacije ažuriranja predaje kao parametar SWUpdate aplikaciji. Recept se kreira na način kako je opisano u potpoglavlju 3.2.3. Konačni izgled recepta za ovaj projekt prikazan je u programskom kodu 4.9.

```
copy1 : {
    files: (
        {
            filename = "concept-image-concept-one-apalis-imx6.tar.gz.enc";
            version = "C1_v0.0";
            type = "archive";
            device = "/dev/mmcblk0p3";
            filesystem = "ext4";
            path = "/";
            sha256 = "@concept-image-concept-one-apalis-imx6.tar.gz.enc";
            encrypted = true; },
        {
            filename = "concept-image-concept-one-BOOT-apalis-imx6.tar.gz.enc";
            version = "C1_v0.0";
            type = "archive";
            device = "/dev/mmcblk0p1";
            filesystem = "ext4";
            path = "/";
            sha256 = "@concept-image-concept-one-BOOT-apalis-imx6.tar.gz.enc";
            encrypted = true; },
        {
            filename = "concept-image-concept-one-OPT-apalis-imx6.tar.gz.enc";
            version = "C1_v0.0";
            type = "archive";
            device = "/dev/mmcblk0p5";
            filesystem = "ext4";
            path = "/";
            sha256 = "@concept-image-concept-one-OPT-apalis-imx6.tar.gz.enc";
            encrypted = true; } );
    );
    scripts: (
        {   filename = "setup.sh.enc";
            type = "shellscript";
            sha256 = "@setup.sh.enc";
            encrypted = true; } );
    uboot: (
        {   name = "emmcselectedpartition";
            value = "primary"; } );
};
```

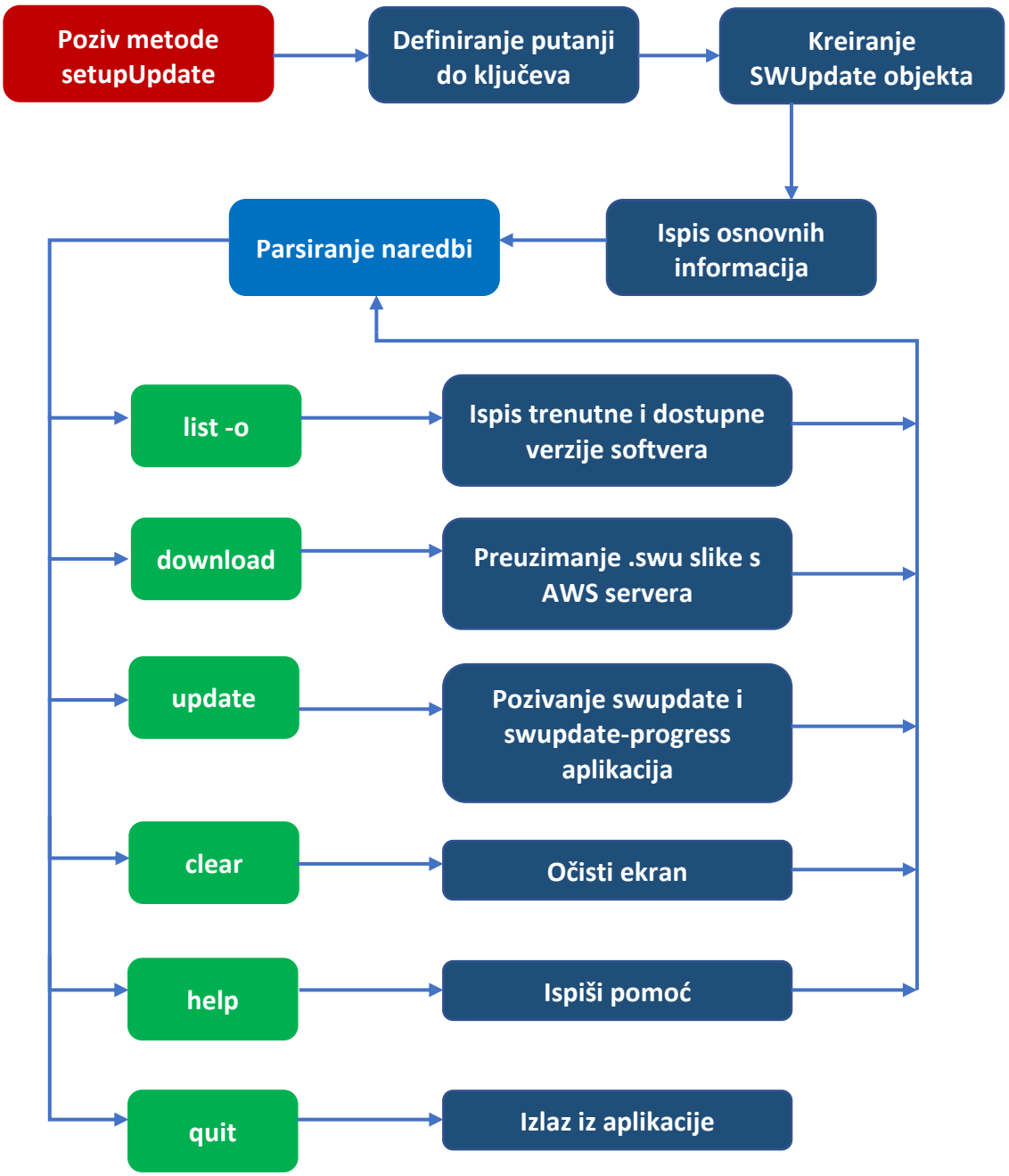
Programski kod 4.8. Struktura ažuriranja koje se instalira na prvu grupu particija

```
################################################################################
## Copyright (C) 2019 Rimac Automobili
## David Tkalcec <david.tkalcec@rimac-automobili.com>
################################################################################
DESCRIPTION = "Build Concept One IVI .swu file for SWUpdate"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${THISDIR}/files-
shared/COPYING.MIT;md5=838c366f69b72c5df05c96dff79b35f2"
FILESEXTRAPATHS_prepend := "${THISDIR}/files-shared:"

do_configure_c1_update () {
   PROJECT=concept-one
   PTAG=$(cat ${DEPLOY_DIR_IMAGE}/SW-VERSIONS/${PROJECT})
   echo "PTAG:  ${PTAG}"
   cd ${WORKDIR}
   export SW_VERSION=$(grep -m 1 C1 sw-description \
   | cut -d'_' -f 2 | cut -d'"' -f 1)
   echo "Software version: $SW_VERSION"
   sed -i -e "s/${SW_VERSION}/${PTAG}/g" sw-description
}
addtask configure_c1_update after do_patch before do_swuimage

DEPENDS += "concept-image-concept-one"
SRC_URI = " \
    file://sw-description        \
    file://setup.sh.enc          \
    file://uEnv-concept-one.enc  "

# slike koje moraju biti izgrađene prije izgradnje .swu slike
IMAGE_DEPENDS = "concept-image-concept-one"
# slike i datoteke koje će biti uključene u .swu sliku
SWUPDATE_IMAGES = " \
    concept-image-concept-one \
    concept-image-concept-one-BOOT \
    concept-image-concept-one-OPT "
SWUPDATE_IMAGES_FSTYPES[concept-image-concept-one] = ".tar.gz.enc"
SWUPDATE_IMAGES_FSTYPES[concept-image-concept-one-BOOT] = ".tar.gz.enc"
SWUPDATE_IMAGES_FSTYPES[concept-image-concept-one-OPT] = ".tar.gz.enc"
SWUPDATE_SIGNING = "RSA"
SWUPDATE_PRIVATE_KEY = "/**/priv.pem"
SWUPDATE_PASSWORD_FILE = "/**/passout"
inherit swupdate
```

Programski kod 4.9. Recept za izgradnju SWU slike za Concpet_one projekt

4.3. Izrada Qt konzolne aplikacije za bežično ažuriranje sustava

Qt konzolna aplikacija izrađena je kako bi olakšala proces instalacije ažuriranja na uređaj. Pruža naredbe za dohvaćanje verzije dostupnog ažuriranja s Amazon S3 *bucketa* i usporedbu s trenutnom verzijom, preuzimanje ažuriranja te instalaciju ažuriranja koristeći SWUpdate aplikaciju. Način rada konzolne aplikacije prikazan je na blok dijagramu 4.1.



Blok dijagram 4.1. Način rada SWUpdate konzolne aplikacije

Za parsiranje naredbi koje korisnik unosi u aplikaciju implementirana je metoda *parseLine* koja kao argument prima unesenu naredbu te na osnovu iste emitira odgovarajući signal (Programski kod 4.10.). Metoda *parseLine* pripada klasi *SWUpdateConsoleInputParser*. Ova klasa nasljeđuje postojeću klasu *ConsoleInputParser* koja sadržava osnovne atribute i metode za parsiranje ulaza konzolne aplikacije. Konzolna aplikacija opisana je klasom *SWUpdateConsole* koja nasljeđuje *QCoreApplication* te sadrži metodu *setupUpdate* koja definira varijable potrebne za kreiranje SWUpdate objekta te povezivanje signala *SWUpdateConsoleInputParser* klase s metodama SWUpdate klase (Programski kod 4.11.). *QCoreApplication* klasa sadrži glavnu petlju događaja (engl. *Event loop*) unutar koje se obrađuju i šalju svi događaji operacijskog sustava. Ova klasa također izvršava inicijalizaciju i finalizaciju aplikacija, kao i postavke na razini sustava i na razini aplikacije.

```
void SWUpdateConsoleInputParser::parseLine(const QString &line){
    ....
    if (line == "list -o"){
        emit listOtaCommand();
    }
    else if (line =="download"){
        emit downloadOtaCommand();
    }

    else if (line == "update"){
        std::cout << "Starting OTA update...\n";
        emit installUpdateCommand();
    } ... }
```

Programski kod 4.10. Dio koda metode *parseLine*

SWUpdate klasa sadrži atribute i metode za cjelokupan proces bežičnog ažuriranja sustava. Neke od osnovnih metoda ove klase su:

1. *listOTAInfo* - ova metoda ispisuje trenutno instaliranu verziju softvera na uređaju te zatim provjerava dostupnu verziju pohranjenu na AWS serveru. Za provjeru dostupne verzije koristi se *s3_download* Python skripta koja koristi Boto3 modul. Skripta se poziva koristeći QProcess objekt kako je prikazano u programskom kodu 4.12. Parametri koji se koriste prilikom poziva su -o (prima lokaciju objekta na AWS serveru) i -v (ispisuje verziju).
2. *downloadSWU* - uspoređuje trenutno instaliranu verziju s dostupnom verzijom. Ukoliko su različite, vrši se preuzimanje ažuriranja pomoću s3_download skripte. Tijekom preuzimanja

ispisuje se napredak unutar konzole te nakon uspješnog preuzimanja ispisuje se odgovarajuća poruka nakon koje je moguće pozvati *updateOTA* metodu. Za pokretanje preuzimanja, *s3_download* skriptu je potrebno pozvati s parametrima -o (prima lokaciju objekta na AWS serveru) i -d (prima lokaciju na koju će .swu slika biti pohranjena).

3. *updateOTA* - poziva aplikacije za instalaciju ažuriranja. Najprije se utvrđuju trenutno korištene particije diska kako bi se odredile odgovarajuće particije na koje ažuriranje treba biti instalirano. Zatim se koriste dva objekta klase QProcess (*m_swuProcess i m_swuProgress*) koji pozivaju *swupdate* i *swupdate-progress* aplikacije (Programski kod 4.13.). *Swupdate* i *swupdate-progress* su osnovne aplikacije koje pruža meta-swupdate sloj. Swupdate aplikacija izvršava proces ažuriranja na temelju podešenih parametara, dok *swupdate-progress* ispisuje napredak ažuriranja unutar konzole te na ekran.

```
...
m_updatePtr= new SWUpdate(this, usbPath, swu, keyPath,
                          pubKeyName, encryptKeyName);
m_updatePtr->printInfo();
m_updatePtr->appInit(); ...
connect(
    &m_inputParser,
    &SWUpdateConsoleInputParser::downloadOtaCommand,
    m_updatePtr,
    &SWUpdate::downloadSWU
);
connect(
    &m_inputParser,
    &SWUpdateConsoleInputParser::installUpdateCommand,
    m_updatePtr,
    &SWUpdate::updateOta
); ...
```

Programski kod 4.11. Dio koda metode *setupUpdate*

```
QProcess process;
//qDebug() <<"Checking available update version...";
process.start("bash", QStringList() << "-c" << "s3_download -o
                              /ivi-swu/" + image_name + " -v");
process.waitForFinished();
QString output = process.readAllStandardOutput();
qDebug().noquote() << output2.replace("\n", "");
QString err2 = process.readAllStandardError();
if (err2 !="") qDebug() << "ERROR: " <<err2;
if (process.errorString()!="Unknown error"){
    qDebug() << process.errorString();
}
```

Programski kod 4.12. Korištenje *s3_download* skripte za provjeru dostupne verzije ažuriranja

```
        m_swuProcess.setEnvironment(QProcess::systemEnvironment());
        m_swuProgress.setEnvironment(QProcess::systemEnvironment());
        m_swuProgress.start("bash", QStringList() << "-c"
                                    <<"swupdate-progress -w -c -p");
        m_swuProgress.waitForStarted();
        m_swuProcess.start("bash", QStringList() << "-c"
                           << "swupdate -v " + selection
                           + " -k " + m_keyPath+m_publicKeyName
                           + " -K " + m_keyPath+m_encryptKeyName
                           + " -i " + m_swuPath + "/" + file);
        m_swuProcess.waitForStarted();
```

Programski kod 4.13. Pokretanje *swupdate* i *swupdate-progress* aplikacija

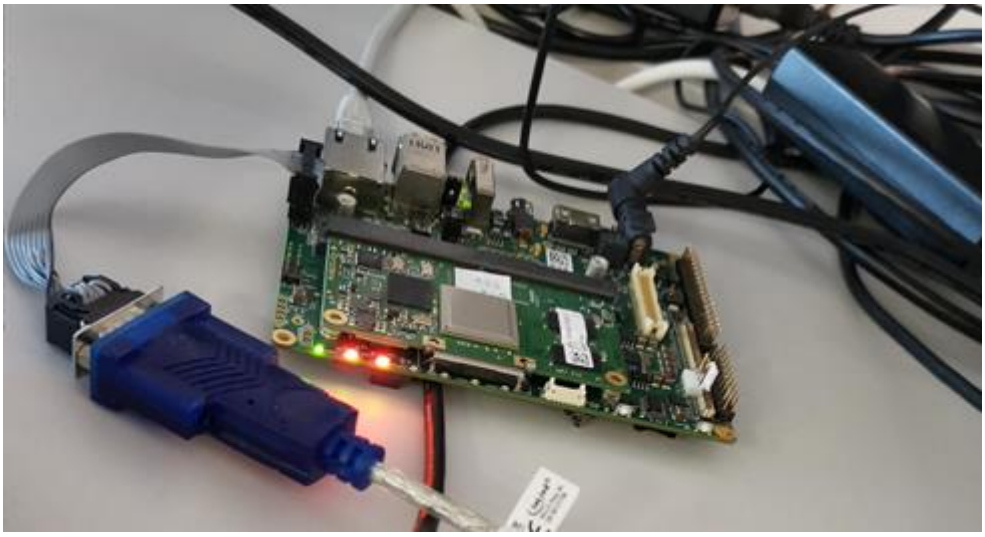
U kodu na slici 4.13. prilikom poziva *swupdate* aplikacije potrebno je postaviti sljedeće parametre: -v (odabrane particije za instalaciju), -k (putanja do ključa korištenog za potpisivanje), -K putanja do ključa korištenog za šifriranje te -i (putanja do .swu slike). Za *swupdate-progress* definirani su parametri -w (čekanje povezivanja sa swupdate aplikacijom), -c (korištenje ispisa u boji) i -p (slanje informacija *psplash* procesu). Python skripta s3_download koristi Boto3 modul za preuzimanje objekta s AWS S3 *bucketa* te za dohvaćanje informacija o objektu. Skriptu je moguće pokrenuti koristeći odgovarajuće parametre prikazane na slici 4.1.

```
root@concept-central:/opt/rapp/bin# s3_download -h
Script for downloading files from AWS S3 bucket.

Command line usage:
    s3_download [-o OBJECT_PATH] [-d DOWNLOAD_PATH] [-h]
Options:
  -o/--object_path /<bucket_name>/<object_name>: location of file on AWS S3
  -d/--download_path /<download_location>: downloading location on device
  -v/--object_version: print version of file on AWS S3
  -h/--help: print this usage message and exit
```

Slika 4.1. Ispis pomoći za *s3_download* skriptu

5. PRIMJER USPJEŠNOG AŽURIRANJA SUSTAVA

U ovome poglavlju prikazan je kompletan postupak ažuriranja Linux operacijskog sustava pokrenutog na Apalis iMX6 modulu koji je povezan na Ixora razvojnu ploču (Slika 5.1.). Za realnu primjenu u vozilu, umjesto Ixora ploče koristi se specifično dizajnirana ploča koja zadovoljava automobilske standarde. Postupak ažuriranja je potpuno jednak u oba slučaja. Najprije je potrebno razvojnu ploču povezati na napajanje, mrežu te na računalo preko serijske veze te zatim pokrenuti operacijski sustav. U ovome radu razvijena je konzolna aplikacija koja upravlja cijelim postupkom ažuriranja kako je opisano u prethodnom poglavlju. Nakon pokretanja sustava potrebno je pozicioniranje u direktorij u kojem se nalazi aplikacija te pokretanje iste. Nakon ovoga koraka dobije se ispis kako je prikazano na slici 5.2.



Slika 5.1. Apalis iMX6 SOC povezan na Ixora razvojnu ploču

```
################################################################################
##                      SWUpdater Console Application                         ##
################################################################################

Usage:

OTA update:
list -o                 --- List available update version
download                --- Download update file
update                  --- Install update

clear                   --- Clear screen
quit                    --- Exit application
```

Slika 5.2. Ispis nakon pokretanja konzolne aplikacije

Nakon pokretanja konzolne aplikacije možemo vidjeti ispis svih dostupnih naredbi. Opisi ovih naredbi dani su u blok dijagramu 4.1 u prethodnom poglavlju. Proces ažuriranja se sastoji od

pokretanja naredbi *download* i *update*. Naredba *list -o* je opcionalna te daje informaciju o trenutno instaliranoj verziji softvera te dostupnoj verziji pohranjenoj na serveru (Slika 5.3.). Pokretanjem naredbe *download* ponovno se vrši usporedba trenutne i dostupne verzije te se pokreće preuzimanje ažuriranja (Slika 5.4.). Nakon uspješnog preuzimanja, moguće je pokrenuti naredbu *update*.

```
################################################################################
##                      SWUpdater Console Application                         ##
################################################################################
list -o
Current software version:  "C1_feature-ota-update "
Checking available update version...
Available update version:  "C1_v2.0.1-RC18"
```

Slika 5.3. Izgled ispisa naredbe *list*

```
download
Current version: C2_feature-ota-update_v3
Avaliable version: C2_feature-ota-update_v3
Downloading update file...
c-two-update-apalis-imx6.swu => 0.25MB/419.46MB => 0.06%
c-two-update-apalis-imx6.swu => 0.50MB/419.46MB => 0.12%
c-two-update-apalis-imx6.swu => 0.75MB/419.46MB => 0.18%
c-two-update-apalis-imx6.swu => 1.00MB/419.46MB => 0.24%
c-two-update-apalis-imx6.swu => 1.25MB/419.46MB => 0.30%
c-two-update-apalis-imx6.swu => 1.50MB/419.46MB => 0.36%
c-two-update-apalis-imx6.swu => 1.75MB/419.46MB => 0.42%
```

Slika 5.4. Izgled ispisa naredbe *download*

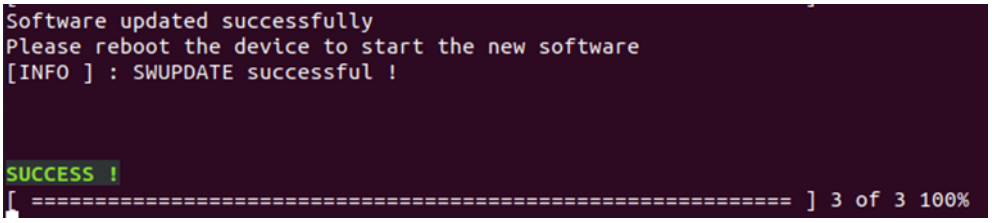
Nakon pokretanja naredbe *update*, aplikacija pronalazi preuzeto ažuriranje, ključeve za dešifriranje pohranjene na uređaju te utvrđuje trenutno pokrenutu kopiju sustava (Slika 5.5). Nakon toga sadržaj preuzetog ažuriranja se dešifrira te kreće instalacija pojedinih artefakata na odgovarajuće particije. U konzoli možemo vidjeti ispis napretka ažuriranja (Slika 5.6.).

```
update
Starting OTA update...
Update file:  "concept-one-update.swu"


Rootfs currently mounted on: "/dev/mmcblk0p4"

Swupdate v2018.3.0

Licensed under GPLv2. See source distribution for detailed copyright notices.

[DEBUG] : SWUPDATE running :  [load_decryption_key] : Read decryption key, initialization v
ector, and salt from file /etc/keys/ra-crypt.

software set: stable mode: copy1

Running on apalis-imx6 Revision 1.0
```

Slika 5.5. Dio ispisa nakon pokretanja *update* naredbe

Slika 5.6. Ispis napretka ažuriranja u konzolnoj aplikaciji

Nakon uspješnog ažuriranja postavljaju se odgovarajuće varijable okruženja u *bootloaderu* kako bi sustav znao koje particije trebaju biti korištene prilikom sljedećeg pokretanja te dobijemo odgovarajuću poruku o uspješnosti (Slika 5.7.). Prilikom sljedećega pokretanja sustava, koriste se particije na koje je instalirano ažuriranje te sustav nastavlja s normalnim radom s ažuriranom verzijom *softvera*.



Slika 5.7. Ispis poruke nakon uspješnog ažuriranja

6. ZAKLJUČAK

Bežična ažuriranja mogu značajno doprinijeti bržem ispravljanju grešaka, instalaciji novih značajki u softveru vozila te manjim novčanim troškovima za proizvođače. Bitno je da takvi sustavi imaju implementirane odgovarajuće sigurnosne mehanizme te da budu robusni na greške kako krajnja sigurnost vozača, vozila i ostalih sudionika u prometu ne bi bila ugrožena. Ovaj rad detaljno uspoređuje dostupne mehanizme za ažuriranje Linux ugradbenog operacijskog sustava. Na temelju usporedbe, SWUpdate mehanizam se pokazao kao najpogodniji za realizaciju ovoga rada zbog iznimne prilagodljivosti. Moguće ga je vrlo dobro, prema specifičnim zahtjevima, prilagoditi za ažuriranje Linux baziranog operacijskog sustava. U praktičnome dijelu rada SWUpdate je korišten za ažuriranje uređaja na kojemu je pokrenut operacijski sustav s aplikacijama informacijsko-zabavnog sustava vozila. Dobiveni sustav ažuriranja nije ograničen samo na ovu primjenu. Sličan sustav, uz odgovarajuće prilagodbe, moguće je koristiti za ažuriranje ostalih jedinica unutar vozila. Kao što je navedeno u uvodnom poglavlju, vozila se sastoje od velikog broja elektroničkih upravljačkih jedinica. Za svaki od njih potrebno je definirati određene zahtjeve i procedure ažuriranja te je potrebno odabrati jedan centralni uređaj koji će vršiti ažuriranje ostalih. Ovaj rad pruža vrlo dobro polazište za daljnji razvoj bežičnog ažuriranja za uređaje unutar vozila, ali i ostale uređaje bazirane na Linux operacijskom sustavu.

LITERATURA

[1] J.R. Quain, **With benefits - and risks - software updates are coming to the car**, 29. lipnja 2018, https://www.digitaltrends.com/cars/over-the-air-software-updates-cars-pros-cons/, [lipanj 2019]

[2] Wikipedia, **Connected car**, 9. lipnja 2019., https://en.wikipedia.org/wiki/Connected_car, [lipanj 2019]

[3] K. Daimi, **Securing Vehicle ECUs Update Over The Air**, AICT 2016, The Twelfth Advanced International Conference on Telecommunications, str. 45-50, 2016.

[4] T. Chowdhury, E. Lesiuta, K. Rikley, **Safe and Secure Automotive Over-the-Air Updates**, str. 172-187, 2018,

[5] Stefano Babic, **SWUpdate**, 2013-2019, https://sbabic.github.io/swupdate/swupdate.html, [lipanj 2019]

[6] Yocto Wiki stranica, **System Update**, 2019, https://wiki.yoctoproject.org/wiki/System_Update, [lipanj 2019]

[7] Mender službena stranica, **Mender pregled**, https://mender.io/overview/solution, [lipanj 2019]

[8] Git repozitorij, **Mender**, 2019, https://github.com/mendersoftware/mender, [lipanj 2019]

[9] Ville Baillie, **OTA updates for Embedded Linux, part 2 - Fundamentals and implementation**, 23. kolovoz 2018, https://www.embedded.com/design/operating-systems/4461020/1/OTA-updates-for-Embedded-Linux--part-2-----A-comparison-of-off-the-shelf-update-systems#, [lipanj 2019]

[10] Jan Luebbe, Enrico Joerns, Juergen Borleis, **RAUC**, 2016-2018, https://rauc.readthedocs.io/en/latest/basic.html, [lipanj 2019]

[11] Qt službena stranica, **Qt**, 2019, https://doc.qt.io/, [lipanj 2019]

[12] Qt službena stranica, **Qt modules**, 2019, https://doc.qt.io/qt-5/qtmodules.html, [lipanj 2019]

[13] Qt službena stranica, **Qt for Beginners**, https://wiki.qt.io/Qt_for_Beginners, [lipanj 2019]

[14] Amazon službena stranica, **Amazon S3**, 2019, https://aws.amazon.com/s3/, [lipanj 2019]

[15] Ramesh Natarajan, **28 Essential AWS S3 CLI Command Examples to Manage Buckets and Objects**, 9. travnja 2019, https://www.thegeekstuff.com/2019/04/aws-s3-cli-examples/, [lipanj 2019]

SAŽETAK

U ovome radu opisani su načini ažuriranja Linux ugradbenog operacijskog sustava. Utvrđeno je koji od njih su najpogodniji za primjenu na informacijsko-zabavnom sustavu vozila te su opisani dostupni mehanizmi koji ih implementiraju. Nakon usporedbe dostupnih mehanizama, SWUpdate je odabran kao najpogodniji za realizaciju praktičnog dijela ovoga rada. U središnjem dijelu rada, dan je opis alata potrebnih za implementaciju sustava, dok je u završnom dijelu opisan način razvoja kompletnog sustava bežičnog ažuriranja za informacijsko-zabavni sustav vozila. Razvijeni sustav omogućava učinkovito preuzimanje podataka potrebnih za ažuriranje, provjeru njihove izvornosti i integriteta, dešifriranje, instalaciju na ugradbeni operacijski sustav te oporavak u slučaju greške.

Ključne riječi: Ažuriranje, AWS, bežično, Linux, SWUpdate, vozilo, Qt

ABSTRACT

This paper describes different methods which can be used to update Linux embedded operating system. The most suitable method for updating operating system of an in-vehicle infotainment system has been chosen and the mechanisms which implement it have been described. After comparing available mechanisms, SWUpdate has been chosen as the most convenient for implementation of the practical part of this paper. Main part contains description of the tools required for implementation, while in final section the way to develop a complete wireless update system for the in-vehicle infotainment is described. The developed system can effectively download update data, verify authenticity and integrity, decrypt, install update on the embedded operating system, and recover in case of failure.

Key words: AWS, Linux, Qt, SWUpdate, update, vehicle, wireless

ŽIVOTOPIS

David Tkalčec rođen je 17. studenog 1995. godine u Virovitici. Završio je Osnovnu školu Petra Preradovića u Pitomači te je nakon završene osnovne škole upisao opći smjer u Gimnaziji Petra Preradovića u Virovitici gdje je prolazio s odličnim uspjehom. Maturirao je 2014. godine, nakon čega je upisao Preddiplomski sveučilišni studij Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Nakon uspješnog završetka preddiplomskog studija računarstva, 2017. godine, dobiva zvanje prvostupnik (baccalaureus) inženjer računarstva (univ. bacc. ing. comp.). Iste godine nastavlja obrazovanje na diplomskom studiju Automobilskog računarstva i komunikacija. Stručnu praksu i realizaciju ovoga rada odrađivao je u tvrtki Rimac Automobili d.o.o. Diplomski studij završava s odličnim uspjehom 2019. godine.